---

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=90 sentences=225, words=1752

---

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 做左右的工作，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. 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/sagas.js

```
import {put,take} from 'redux-saga/effects';
import * as types from './store/action-types';

export function* rootSaga() {
    for (let i=0;i<3;i++){
        yield take(types.INCREMENT_ASYNC);
        yield put({type:types.INCREMENT});
    }
    console.log('已经达到最大值');
}
```

src/components/Counter.js

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

src/store/index.js

```
import {createStore, applyMiddleware} from 'redux';
import reducer from './reducer';
import createSagaMiddleware from 'redux-saga';
import {rootSaga} from '../saga';
let sagaMiddleware=createSagaMiddleware();
let store=applyMiddleware(sagaMiddleware)(createStore)(reducer);
sagaMiddleware.run(rootSaga);
window.store=store;
export default store;
```

src/store/actions.js

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

src/store/action-types.js

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

rc/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;
    }
}
```

5. 实现take

- runSaga.js (https://github.com/redux-saga/redux-saga/blob/master/packages/core/src/internal/runSaga.js)**5.1 index.js**redux-saga/index.js

```
export default function createSagaMiddleware() {
  function createChannel() {
      let listener={};
      function subscribe(actionType,cb) {
          listener[actionType]=cb;
      }
      function publish(action) {
          if (listener[action.type]) {
              let temp=listener[action.type];
              delete listener[action.type];
              temp(action);
          }
      }
      return {subscribe,publish}
  }
  let channel=createChannel();
  function sagaMiddleware({getState,dispatch}) {
      function run(generator) {
          let it=generator();
          function next(action) {
              let {value:effect,done} = it.next(action);
              if (!done) {
                  switch (effect.type) {
                      case 'TAKE':
                          channel.subscribe(effect.actionType,next);
                          break;
                      case 'PUT':
                          dispatch(effect.action);
                          next();
                          break;
                      default:
                  }
              }
          }
          next();
      }
      sagaMiddleware.run=run;
      return function (next) {
          return function (action) {
              channel.publish(action);
              next(action);
          }
      }
  }
  return sagaMiddleware;
}
```

src\redux-saga\effects.js

```
export function take(actionType) {
    return {
        type:'take',
        actionType
    }
}

export function put(action) {
    return {
        type: 'put',
        action
    }
}
```

6.支持产出terator

遍历器（Iterator）就是这样一种机制。它是一种接口，为各种不同的数据结构提供统一的访问机制。任何数据结构只要部署 Iterator 接口，就可以完成遍历操作（即依次处理该数据结构的所有成员）。 Iterator 的作用有三个：

- 一是为各种数据结构，提供一个统一的、简便的访问接口
- 二是使得数据结构的成员能够按某种次序排列
- 三是 ES6 创造了一种新的遍历命令for...of循环，Iterator 接口主要供for...of消费

原生具备 Iterator 接口的数据结构如下。

- Array
- Map
- Set
- String
- TypedArray
- 函数的 arguments 对象
- NodeList 对象

```
var arr = [1,2];
let iterator = arr[Symbol.iterator]();
console.log(iterator.next());
console.log(iterator.next());
console.log(iterator.next());
```

```
let obj = {
    name:'zhufeng',
    age:10,
    [Symbol.iterator]() {
        const self = this;
        let values = Object.values(self);
        let index = 0;
        return {
            next() {
                return {value:values[index++],done:index>values.length?true:false};
            }
        };
    }
}
let iterator = obj[Symbol.iterator]();
console.log(iterator.next());
console.log(iterator.next());
console.log(iterator.next());
```

```
import {put,take} from '../redux-saga/effects';
import * as types from './action-types';

+export function* increment() {
+    yield put({type:types.INCREMENT});
+}

export function* rootSaga() {
    for (let i=0;i+        yield increment();
    }
    console.log('已经达到最大值');
}
```

```
export default function createSagaMiddleware() {
    function createChannel() {
        let listener={};
        function subscribe(actionType,cb) {
            listener[actionType]=cb;
        }
        function publish(action) {
            if (listener[action.type]) {
                let temp=listener[action.type];
                delete listener[action.type];
                temp(action);
            }
        }
        return {subscribe,publish};
    }
    let channel=createChannel();
    function sagaMiddleware({getState,dispatch}) {
        function run(generator) {
+            let it= typeof generator == 'function'? generator():generator;
            function next(action) {
                let {value:effect,done} = it.next(action);
                if (!done) {
+                    if (typeof effect[Symbol.iterator]=='function') {
+                        run(effect);
+                        next();
+                    } else {
                       switch (effect.type) {
                           case 'TAKE':
                               channel.subscribe(effect.actionType,next);
                               break;
                           case 'PUT':
                               dispatch(effect.action);
                               next();
                               break;
                           default:
                       }
                   }
                }
            }
            next();
        }
        sagaMiddleware.run=run;
        return function (next) {
            return function (action) {
                channel.publish(action);
                next(action);
            }
        }
    }
    return sagaMiddleware;
}
```

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

```
const task = yield fork(otherSaga, ...args)
```

```
import {put,takeEvery} from '../redux-saga/effects';
import * as types from './action-types';
export function* increment() {
    yield put({type:types.INCREMENT});
}

export function* rootSaga() {
+    yield takeEvery(types.INCREMENT_ASYNC,increment);
}
```

src\redux-saga\effects.js

```
export function take(actionType) {
    return {
        type:'TAKE',
        actionType
    }
}

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

export function fork(task) {
    return {
        type: 'FORK',
        task
    }
}

export function* takeEvery(actionType,task) {
    yield fork(function* () {
        while (true) {
            yield take(actionType);
            yield task();
        }
    });
}
```

```
                        switch (effect.type) {
                            case 'TAKE':
                                channel.subscribe(effect.actionType,next);
                                break;
                            case 'PUT':
                                dispatch(effect.action);
                                next();
                                break;
+                            case 'FORK':
+                                run(effect.task);
+                                next();
+                                break;
                            default:
                                break;
                        }
```

8. 支持promise

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

export function* rootSaga() {
    yield takeEvery(types.INCREMENT_ASYNC,increment);
}
```

```
function next(action) {
                let {value:effect,done} = it.next(action);
                if (!done) {
                    if (typeof effect[Symbol.iterator]=='function') {
                        run(effect);
                        next();
+                    } else if(effect.then){
+                        effect.then(next);
+                    }else {
                        switch (effect.type) {
                            case 'TAKE':
                                channel.subscribe(effect.actionType,next);
                                break;
```

9. 支持 call

```
export function* increment() {

    yield call(delay,1000);
    yield put({type:types.INCREMENT});
}
```

```
export function call(fn,...args) {
    return {
        type: 'CALL',
        fn,
        args
    }
}
```

```
    case 'CALL':
        effect.fn(...effect.args).then(next);
    break;
```

10. 支持 delay

```
+import {put,takeEvery,delay} from '../redux-saga/effects';
import * as types from './action-types';

export function* increment() {
+    yield delay(1000);
    yield put({type:types.INCREMENT});
}

export function* rootSaga() {
    yield takeEvery(types.INCREMENT_ASYNC,increment);
}
```

```
const innerDelay=ms => new Promise((resolve,reject) => {
    setTimeout(() => {
        resolve();
    },ms);
});
export function delay(...args) {
    return call(innerDelay,...args);
}
```

11. 支持 cps

```
import {put,takeEvery,call,cps} from '../redux-saga/effects';
import * as types from './action-types';

+const delay = (ms,callback)=>{
+    setTimeout(() => {
+        callback('ok');
+    },ms);
+}

+export function* increment() {
+    let data = yield cps(delay,1000);
+    console.log(data);
+    yield put({type:types.INCREMENT});
+}

export function* rootSaga() {
    yield takeEvery(types.INCREMENT_ASYNC,increment);
}
```

```
export function cps(fn,...args) {
    return {
        type: 'CPS',
        fn,
        args
    }
}
```

```
    case 'CALL':
        effect.fn(...effect.args).then(next);
        break;
+    case 'CPS':
+        effect.fn(...effect.args,next);
+        break;
    default:
        break;
```

12. 支持all

src\store\saga.js

```
import {put,takeEvery,delay,all} from '../redux-saga/effects';
import * as types from './action-types';

export function* increment() {
    yield delay(1000);
    yield put({type:types.INCREMENT});
}
export function* incrementWatcher() {
    yield takeEvery(types.INCREMENT_ASYNC,increment);
}
export function* logger() {
    console.log('action');
}
export function* loggerWatcher() {
    yield takeEvery(types.INCREMENT_ASYNC,logger);
}

export function* rootSaga() {
    yield all([incrementWatcher(),loggerWatcher()]);
    console.log('done');
}
```

```
export function all(fns) {
    return {
        type: 'ALL',
        fns
    }
}
```

```
export default function createSagaMiddleware() {
    function createChannel() {
        let listener={};
        function subscribe(actionType,cb) {
            listener[actionType]=cb;
        }
        function publish(action) {
            if (listener[action.type]) {
                let temp=listener[action.type];
                delete listener[action.type];
                temp(action);
            }
        }
        return {subscribe,publish};
    }
    let channel=createChannel();
+    function times(cb,total) {
+        let count=0;
+        return function () {
+            if (++count === total) {
+                cb();
+            }
+        }
+    }
    function sagaMiddleware({getState,dispatch}) {
+        function run(generator,callback) {
            let it= typeof generator == 'function'? generator():generator;
            function next(action) {
                let {value:effect,done} = it.next(action);
                if (!done) {
                    if (typeof effect[Symbol.iterator]=='function') {
                        run(effect);
                        next();
                    } else if(effect.then){
                        effect.then(next);
                    }else {
                        switch (effect.type) {
                            case 'TAKE':
                                channel.subscribe(effect.actionType,next);
                                break;
                            case 'PUT':
                                dispatch(effect.action);
                                next();
                                break;
                            case 'FORK':
                                run(effect.task);
                                next();
                                break;
                            case 'CALL':
                                effect.fn(...effect.args).then(next);
                                break;
+                            case 'ALL':
+                                let fns=effect.fns;
+                                let done=times(next,fns.length);
+                                for (let i=0;i
+                                    let fn=fns[i];
+                                    run(fn,done);
+                                }
+                                break;
                            case 'CPS':
                                effect.fn(...effect.args,next);
                                break;
                            default:
                                break;
                        }
                    }
+                } else {
+                    callback&&callback();
+                }
            }
            next();
        }
        sagaMiddleware.run=run;
        return function (next) {
            return function (action) {
                channel.publish(action);
                next(action);
            }
        }
    }
    return sagaMiddleware;
}
```

13. 取消任务

src\store\saga.js

```
import {put,take,delay,all,fork,cancel} from '../redux-saga/effects';
import * as types from './action-types';

export function* increment() {
    while(true){
        yield delay(1000);
        yield put({type:types.INCREMENT});
    }
}

export function* incrementWatcher() {
    const task = yield fork(increment);
    yield take(types.STOP_INCREMENT);
    yield cancel(task);
}

export function* rootSaga() {
    yield all([incrementWatcher()]);
    console.log('done');
}
```

src\components\Counter.js

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

src\redux-saga\effects.js

```
export function cancel(task) {
    return {
        type: 'CANCEL',
        task
    }
}
```

src\redux-saga\index.js

```
function sagaMiddleware({getState,dispatch}) {
        function run(generator,callback) {
            let it= typeof generator == 'function'? generator():generator;
            function next(action) {
                let {value:effect,done} = it.next(action);
                if (!done) {
                    if (typeof effect[Symbol.iterator]=='function') {
                        run(effect);
                        next();
                    } else if(effect.then){
                        effect.then(next);
                    }else {
                        switch (effect.type) {
                            case 'TAKE':
                                channel.subscribe(effect.actionType,next);
                                break;
                            case 'PUT':
                                dispatch(effect.action);
                                next();
                                break;
+                               case 'FORK':
+                                   let newTask = effect.task();
+                                   run(newTask);
+                                   next(newTask);
+                                   break;
                            case 'CALL':
                                effect.fn(...effect.args).then(next);
                                break;
                            case 'ALL':
                                let fns=effect.fns;
                                let done=times(next,fns.length);
                                for (let i=0;i+                                case 'CANCEL':
+                                   effect.task.return('over');
+                                   break;
                            default:
                                break;
                        }
                    }
                } else {
                    callback&&callback();
                }
            }
            next();
        }
        sagaMiddleware.run=run;
        return function (next) {
            return function (action) {
                channel.publish(action);
                next(action);
            }
        }
    }
```

14. 资源