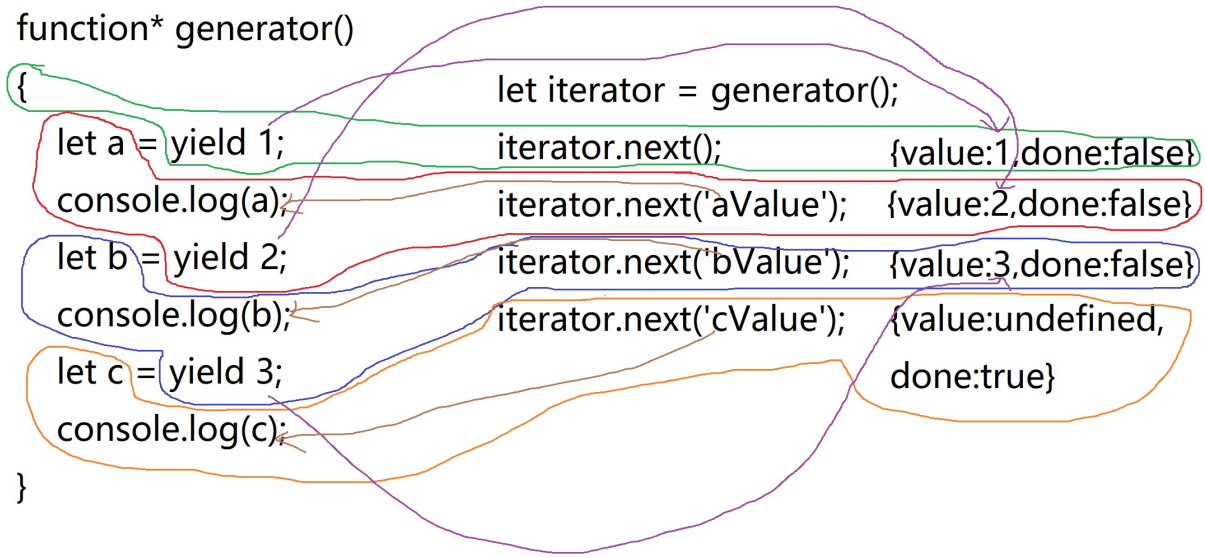
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=285 sentences=533, words=4085

1. Generator

1.1 Generator

- Generator是生成器，一个函数如果加了*，他就会变成一个 生成器函数，他的运行结果会返回一个 迭代器对象
- ES6规范中规定迭代器必须有一个next方法，这个方法会返回一个对象，这个对象具有 done和 value两个属性
  - done表示当前迭代器是否已经执行完，执行完为true，否则为false
  - value表示当前步骤返回的值
- 当调用迭代器的 next方法时，会继续往下执行,遇到 yield关键字都会暂停执行,并会将 yield后面表达式的值作为返回对象的 value




```
function* generator() {
    let a = yield 1;
    console.log(a);
    let b = yield 2;
    console.log(b);
    let c = yield 3;
    console.log(c);
}

let iterator = generator();

let r1 = iterator.next();

console.log(r1);

let r2 = iterator.next('aValue');
console.log(r2);

let r3 = iterator.next('bValue');
console.log(r3);

let r4 = iterator.next('cValue');
console.log(r4);
```

1.3 throw

- throw方法可以在函数体外部抛出错误，然后在函数里面捕获

```
function* generator() {
    try{
        let a = yield 1;
        console.log(a);
        let b = yield 2;
        console.log(b);
    }catch(err){
        console.log(err);
    }
}

let iterator = generator();
let r1 = iterator.next();
console.log(r1);
let r2 =  iterator.throw('出错了');
console.log(r2);
```

1.4 return

- throw方法可以终止当前迭代器

```
function* generator() {
    try{
        let a = yield 1;
        console.log(a);
        let b = yield 2;
        console.log(b);
    }catch(err){
        console.log(err);
    }
}

let iterator = generator();
let r1 = iterator.next();
console.log(r1);
let r2 =  iterator.return();
console.log(r2);
```

1.5 co

```
function co(generator) {
    let it = generator();
    let result;
    function next(arg) {
        result = it.next(arg);
        if(!result.done)
            next(result.value);
    }
    next();
}

function* generator() {
    let a = yield 1;
    console.log(a);
    let b = yield 2;
    console.log(b);
    let c = yield 3;
    console.log(c);
}
co(generator);
```

1.7 generator

1.7.1 使用

```
function* generator() {
    let a = yield 1;
    console.log(a);
    let b = yield 2;
    console.log(b);
    let c = yield 3;
    console.log(c);
}
var iterator = generator();
console.log(iterator.next());
console.log(iterator.next('aValue'));
console.log(iterator.next('bValue'));
console.log(iterator.next('cValue'));
```

1.7.2 实现

- repl (https://babeljs.io/repl)
- runtime.js (https://github.com/facebook/regenerator/blob/master/packages/regenerator-runtime/runtime.js)

1.7.2.1 generator.js

```
let regeneratorRuntime = require('./regeneratorRuntime');
var _marked = regeneratorRuntime.mark(generator);

function generator() {
  var a, b, c;
  return regeneratorRuntime.wrap(function generator$(_context) {
    while (1) {
      switch (_context.prev = _context.next) {
        case 0:
          _context.next = 2;
          return 1;

        case 2:
          a = _context.sent;
          console.log(a);
          _context.next = 6;
          return 2;

        case 6:
          b = _context.sent;
          console.log(b);
          _context.next = 10;
          return 3;

        case 10:
          c = _context.sent;
          console.log(c);

        case 12:
        case "end":
          return _context.stop();
        default:
      }
    }
  }, _marked);
}
var iterator = generator();
console.log(iterator.next());
console.log(iterator.next('aValue'));
console.log(iterator.next('bValue'));
console.log(iterator.next('cValue'));
```

1.7.2.2 regeneratorRuntime.js

```
class Context {
    next = 0;
    done = false;
    stop() {
        this.done = true;
    }
}
exports.mark = function (genFun) {
    return genFun;
};
exports.wrap = (innerFn, outerFn) => {
    let generator = Object.create(outerFn.prototype);
    var context = new Context();
    generator.next = (arg) => {
        context.sent = arg;
        let value = innerFn(context);
        return {
            value,
            done: context.done
        };
    };
    return generator;
}
```

2. redux-saga实战

2.1 初始化项目

```
create-react-app zhufeng-redux-saga
cd zhufeng-redux-saga
cnpm i redux react-redux redux-saga -S
```

2.2 实现计数器

2.2.1 action-types.js

src\store\action-types.js

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

2.2.2 reducer.js

src\store\reducer.js

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

2.2.3 store\index.js

src\store\index.js

```
import {createStore} from 'redux';
import reducer from './reducer';
let store=(createStore)(reducer);
export default store;
```

2.2.4 actions.js

src\store\actions.js

```
import * as actionTypes from './action-types';
const actions = {
    add() {
        return {type:actionTypes.ADD}
    }
}
export default actions;
```

2.2.5 Counter.js

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.add}>+button>
            div>
      )
  }
}
export default connect(
    state => state,
    actions
)(Counter);
```

2.2.6 src\index.js

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'));
```

2.3 saga计数器

2.3.1 src\store\sagas.js

src\store\sagas.js

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

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

2.3.2 action-types.js

src\store\action-types.js

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

2.3.3 actions.js

src\store\actions.js

```
import * as actionTypes from './action-types';
const actions = {
    add() {
+       return {type:actionTypes.ASYNC_ADD}
    }
}
export default actions;
```

2.3.4 src\store\index.js

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;
```

3 实现take和put

- take 监听action，暂停Generator，匹配的action被发起时，恢复执行。take结合fork，可以实现takeEvery和takeLatest的效果
- put 相当于dispatch，分发一个action

3.1 index.js

- index.js (https://gitee.com/zhufengpeixun/redux-saga/blob/master/packages/core/src/index.js)

```
export { default } from './middleware';
```

3.2 middleware.js

- middleware.js (https://gitee.com/zhufengpeixun/redux-saga/blob/master/packages/core/src/internal/middleware.js)

```
import { stdChannel } from './channel';
import { runSaga } from './runSaga';

function sagaMiddlewareFactory() {
  const channel = stdChannel();

  let boundRunSaga;
  function sagaMiddleware({ getState, dispatch }) {
    boundRunSaga = runSaga.bind(null, {
      channel,
      dispatch,
      getState,
    })

    return function (next) {
      return function (action) {
        const result = next(action);
        channel.put(action);
        return result;
      }
    }
  }

  sagaMiddleware.run = (...args) => {
    boundRunSaga(...args)
  }

  return sagaMiddleware;
}

export default sagaMiddlewareFactory;
```

3.3 channel.js

- channel.js (https://gitee.com/zhufengpeixun/redux-saga/blob/master/packages/core/src/internal/channel.js)

```
export function stdChannel() {
  let currentTakers = [];
  function take(cb, matcher) {
    cb['MATCH'] = matcher;
    cb.cancel = ()=>{
      currentTakers = currentTakers.filter(item=>item!==cb);
    }
    currentTakers.push(cb);
  }

  function put(input) {
    const takers = currentTakers;
    for (let i = 0, len = takers.length; i < len; i++) {
      const taker = takers[i]
      if (taker['MATCH'](input)) {
        taker.cancel();
        taker(input);
      }
    }
  }

  return {
    take,
    put
  }
}
```

3.4 runSaga.js

- runSaga.js (https://gitee.com/zhufengpeixun/redux-saga/blob/master/packages/core/src/internal/runSaga.js)

```
import proc from './proc';

export function runSaga(
  { channel, dispatch, getState },
  saga,
  ...args
) {
  const iterator = saga(...args);

  const env = {
    channel,
    dispatch,
    getState,
  };

  proc(env, iterator);
}
```

3.5 effectTypes.js

src\redux-saga\effectTypes.js

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

3.6 effectRunnerMap.js

- effectRunnerMap.js (https://gitee.com/zhufengpeixun/redux-saga/blob/master/packages/core/src/internal/effectRunnerMap.js)

```
import * as effectTypes from './effectTypes'
function runTakeEffect(env, {pattern}, cb) {
  const matcher = input => input.type === pattern;
  env.channel.take(cb, matcher);
}

function runPutEffect(env, { action }, cb) {
  const result = env.dispatch(action);
  cb(result);
}

const effectRunnerMap = {
  [effectTypes.TAKE]: runTakeEffect,
  [effectTypes.PUT]: runPutEffect
};

export default effectRunnerMap;
```

3.6 proc.js

- proc.js (https://gitee.com/zhufengpeixun/redux-saga/blob/master/packages/core/src/internal/proc.js)

```
import effectRunnerMap from './effectRunnerMap';

export default function proc(env, iterator) {
  next();
  function next(arg, isErr) {
    let result;
    if (isErr) {
      result = iterator.throw(arg);
    } else {
      result = iterator.next(arg);
    }
    if (!result.done) {
      runEffect(result.value, next)
    }
  }

  function runEffect(effect, next) {
    if (effect) {
      const effectRunner = effectRunnerMap[effect.type]
      effectRunner(env, effect.payload, next,{runEffect});
    } else {
      next();
    }
  }
}
```

3.7 effects.js

- effects.js (https://gitee.com/zhufengpeixun/redux-saga/blob/master/packages/core/src/effects.js)

```
import * as effectTypes from './effectTypes'
const makeEffect = (type, payload) => ({
  type,
  payload
})

export function take(pattern) {
  return makeEffect(effectTypes.TAKE, { pattern })
}

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

4. 支持产出iterator

4.1 sagas.js

src\store\sagas.js

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

export function* add() {
  yield put({type:actionTypes.ADD});
}

export function* rootSaga() {
     for (let i=0;i<3;i++){
         yield take(actionTypes.ASYNC_ADD);
         yield add();
     }
     console.log('已经达到最大值');
}
```

4.2 is.js

src\redux-saga\is.js

```
export const func = f => typeof f === 'function';
export const iterator = it => it && func(it.next) && func(it.throw);
```

4.3 proc.js

src\redux-saga\proc.js

```
import effectRunnerMap from './effectRunnerMap';
+import * as is  from './is';
+export default function proc(env, iterator,cont) {
  next();
  function next(arg, isErr) {
    let result;
    if (isErr) {
      result = iterator.throw(arg);
    } else {
      result = iterator.next(arg);
    }
    if (!result.done) {
      runEffect(result.value, next)
+    }else{
+      cont&&cont(result.value);
+    }
  }

  function runEffect(effect, next) {
+    if(is.iterator(effect)){
+      proc(env,effect,next);
+    }else if (effect) {
      const effectRunner = effectRunnerMap[effect.type]
+     effectRunner(env, effect.payload, next,{runEffect});
    } else {
      next();
    }
  }
}
```

5. 支持takeEvery

- takeEvery 监听action，每监听到一个action，就执行一次操作
- fork 异步非阻塞调用，无阻塞的执行fn，执行fn时，不会暂停Generator
- 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

5.1 sagas.js

src\store\sagas.js

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

export function* add() {
  yield put({type:actionTypes.ADD});
}

export function* rootSaga() {
+  yield takeEvery(actionTypes.ASYNC_ADD,add);
}
```

5.2 effectTypes.js

src\redux-saga\effectTypes.js

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

5.3 effects.js

src\redux-saga\effects.js import * as effectTypes from './effectTypes'

```
const makeEffect = (type, payload) => ({
  type,
  payload
})

export function take(pattern) {
  return makeEffect(effectTypes.TAKE, { pattern })
}

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

+export function fork(fn) {
+  return makeEffect(effectTypes.FORK, { fn })
+}

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

5.4 effectRunnerMap.js

src\redux-saga\effectRunnerMap.js

```
import * as effectTypes from './effectTypes'
+import proc from "./proc";
function runTakeEffect(env, {pattern}, cb) {
  const matcher = input => input.type
  env.channel.take(cb, matcher);
}

function runPutEffect(env, { action }, cb) {
  const result = env.dispatch(action);
  cb(result);
}

+function runForkEffect(env, { fn }, cb) {
+  const taskIterator = fn();
+  proc(env, taskIterator);
+  cb();
+}

const effectRunnerMap = {
  [effectTypes.TAKE]: runTakeEffect,
  [effectTypes.PUT]: runPutEffect,
+ [effectTypes.FORK]: runForkEffect
};

export default effectRunnerMap;
```

6 支持promise

6.1 sagas.js

src\store\sagas.js

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

export function* rootSaga() {
  yield takeEvery(actionTypes.ASYNC_ADD,add);
}
```

6.2 is.js

src\redux-saga\is.js

```
export const func = f => typeof f
export const iterator = it => it && func(it.next) && func(it.throw);
+export const promise = p => p && func(p.then);
```

6.3 proc.js

src\redux-saga\proc.js

```
import effectRunnerMap from './effectRunnerMap';
+import * as is from './is';
+function resolvePromise(promise, cb) {
+  promise.then(cb, function (error) {
+    cb(error, true);
+  });
+}
export default function proc(env, iterator,cont) {
  next();
  function next(arg, isErr) {
    let result;
    if (isErr) {
      result = iterator.throw(arg);
    } else {
      result = iterator.next(arg);
    }
    if (!result.done) {
      runEffect(result.value, next)
    }else{
      cont&&cont(result.value);
    }
  }

  function runEffect(effect, next) {
+    if (is.promise(effect)) {
+      resolvePromise(effect, next);
+    }else if(is.iterator(effect)){
      proc(env,effect,next);
    }else if (effect) {
      const effectRunner = effectRunnerMap[effect.type]
      effectRunner(env, effect.payload, next,{runEffect});
    } else {
      next();
    }
  }
}
```

7.支持call

- 异步阻塞调用**7.1 sagas.js #**src\store\sagas.js ````diff +import { put, takeEvery,call} from '../redux-saga/effects'; //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 function* rootSaga() { yield takeEvery(actionTypes.ASYNC_ADD,add); }

```
### 7.2 effectTypes.js
src\redux-saga\effectTypes.js
```diff
export const TAKE = 'TAKE';
export const PUT = 'PUT';
export const FORK = 'FORK';
+export const CALL = 'CALL';
```

** 7.3 effects.js #**

src\redux-saga\effects.js

```
const makeEffect = (type, payload) => ({
  type,
  payload
})

export function take(pattern) {
  return makeEffect(effectTypes.TAKE, { pattern })
}

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

export function fork(fn) {
  return makeEffect('FORK', { fn })
}

export function takeEvery(pattern, saga) {
  function* takeEveryHelper() {
    while (true) {
      yield take(pattern);
      yield fork(saga);
    }
  }

  return fork(takeEveryHelper);
}

+export function call(fn, ...args) {
+  return makeEffect(effectTypes.CALL, { fn, args })
+}
```

** 7.4 effectRunnerMap.js #**

src\redux-saga\effectRunnerMap.js

```
import * as effectTypes from './effectTypes'
import proc from "./proc";
import * as is from './is';
function runTakeEffect(env, {pattern}, cb) {
  const matcher = input => input.type
  env.channel.take(cb, matcher);
}

function runPutEffect(env, { action }, cb) {
  const result = env.dispatch(action);
  cb(result);
}

function runForkEffect(env, { fn }, cb) {
  const taskIterator = fn();
  proc(env, taskIterator);
  cb();
}

+function runCallEffect(env, { fn, args }, cb) {
+  const result = fn.apply(null, args);
+  if (is.promise(result)) {
+    return result
+      .then(data => cb(data))
+      .catch(error => cb(error, true));
+  }
+  cb(result);
+}

const effectRunnerMap = {
  [effectTypes.TAKE]: runTakeEffect,
  [effectTypes.PUT]: runPutEffect,
  [effectTypes.FORK]: runForkEffect,
+ [effectTypes.CALL]: runCallEffect
};

export default effectRunnerMap;
```

8 支持cps

** 8.1 sagas.js #**

src\store\sagas.js
```

```
+import { put, takeEvery,call,cps} from '../redux-saga/effects';
//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 function* rootSaga() {
  yield takeEvery(actionTypes.ASYNC_ADD,add);
}
```

** 8.2 is.js#**

src\redux-saga\is.js

```
export const func = f => typeof f
export const iterator = it => it && func(it.next) && func(it.throw);
export const promise = p => p && func(p.then);
+export const undef = v => v === null || v === undefined
```

** 8.3 effectTypes.js#**

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';
```

** 8.4 effects.js#**

src\redux-saga\effects.js

```
import * as effectTypes from './effectTypes'
const makeEffect = (type, payload) => ({
  type,
  payload
})

export function take(pattern) {
  return makeEffect(effectTypes.TAKE, { pattern })
}

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

export function fork(fn) {
  return makeEffect(effectTypes.FORK, { fn })
}

export function takeEvery(pattern, saga) {
  function* takeEveryHelper() {
    while (true) {
      yield take(pattern);
      yield fork(saga);
    }
  }

  return fork(takeEveryHelper);
}

export function call(fn, ...args) {
  return makeEffect(effectTypes.CALL, { fn, args })
}
+export function cps(fn, ...args) {
+  return makeEffect(effectTypes.CPS, { fn, args })
+}
```

** 8.5 effectRunnerMap.js#**

src\redux-saga\effectRunnerMap.js

```
import * as effectTypes from './effectTypes'
import proc from "./proc";
import * as is  from './is';
function runTakeEffect(env, {pattern}, cb) {
  const matcher = input => input.type
  env.channel.take(cb, matcher);
}

function runPutEffect(env, { action }, cb) {
  const result = env.dispatch(action);
  cb(result);
}

function runForkEffect(env, { fn }, cb) {
  const taskIterator = fn();
  proc(env, taskIterator);
  cb();
}

function runCallEffect(env, { fn, args }, cb) {
  const result = fn.apply(null, args);
  if (is.promise(result)) {
    return result
      .then(data => cb(data))
      .catch(error => cb(error, true));
  }
  cb(result);
}

+function runCPSEffect(env, {context,fn,args}, cb) {
+  const cpsCb = (err, res) => {
+    if (is.undef(err)) {
+      cb(res);
+    } else {
+      cb(err, true);
+    }
+  }
+  fn.apply(context, args.concat(cpsCb));
+}

const effectRunnerMap = {
  [effectTypes.TAKE]: runTakeEffect,
  [effectTypes.PUT]: runPutEffect,
  [effectTypes.FORK]: runForkEffect,
  [effectTypes.CALL]: runCallEffect,
+ [effectTypes.CPS]: runCPSEffect
};

export default effectRunnerMap;
```

9.支持all

- all合并多个异步操作,当某个操作失败或者全部操作成功则进行返回
- all中的异步操作是并发也是同步,不用等一个结束,也不用等另一个开始

** 9.1 sagas.js #**

src\store\sagas.js

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

+export function* add1() {
+  for (let i=0;i
+      yield take(actionTypes.ASYNC_ADD);
+      yield put({type:actionTypes.ADD});
+  }
+  return 'add1';
+}
+export function* add2() {
+  for (let i=0;i
+      yield take(actionTypes.ASYNC_ADD);
+      yield put({type:actionTypes.ADD});
+  }
+  return 'add2';
+}

export function* rootSaga() {
+  let result = yield all([add1(),add2()]);
+  console.log('done',result);
}
```

** 9.2 effectTypes.js #**

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';
```

** 9.3 effects.js #**

src\redux-saga\effects.js

```
import * as effectTypes from './effectTypes'
const makeEffect = (type, payload) => ({
  type,
  payload
})

export function take(pattern) {
  return makeEffect(effectTypes.TAKE, { pattern })
}

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

export function fork(fn) {
  return makeEffect(effectTypes.FORK, { fn })
}

export function takeEvery(pattern, saga) {
  function* takeEveryHelper() {
    while (true) {
      yield take(pattern);
      yield fork(saga);
    }
  }

  return fork(takeEveryHelper);
}

export function call(fn, ...args) {
  return makeEffect(effectTypes.CALL, { fn, args })
}
export function cps(fn, ...args) {
  return makeEffect(effectTypes.CPS, { fn, args })
}
+export function all(effects) {
+  return makeEffect(effectTypes.ALL, effects)
+}
```

** 9.4 utils.js#**

src\redux-saga\utils.js

```
export function createAllStyleChildCallbacks(shape, parentCallback) {
    const keys = Object.keys(shape);
    const totalCount = keys.length;
    let completedCount = 0;
    const results = new Array(totalCount);
    const childCallbacks = {}

    function checkEnd() {
      if (completedCount === totalCount) {
        parentCallback(results)
      }
    }

    keys.forEach(key => {
      childCallbacks[key] = (res) => {
        results[key] = res;
        completedCount++
        checkEnd()
      }
    })
    return childCallbacks
  }
```

** 9.5 effectRunnerMap.js#**

src\redux-saga\effectRunnerMap.js

```
import proc from "./proc";
import * as is  from './is';
import * as effectTypes from './effectTypes'
import {createAllStyleChildCallbacks} from './utils';
function runTakeEffect(env, {pattern}, cb) {
  const matcher = input => input.type
  env.channel.take(cb, matcher);
}

function runPutEffect(env, { action }, cb) {
  const result = env.dispatch(action);
  cb(result);
}

function runForkEffect(env, { fn }, cb) {
  const taskIterator = fn();
  proc(env, taskIterator);
  cb();
}

function runCallEffect(env, { fn, args }, cb) {
  const result = fn.apply(null, args);
  if (is.promise(result)) {
    return result
      .then(data => cb(data))
      .catch(error => cb(error, true));
  }
  cb(result);
}

function runCPSEffect(env, {context,fn,args}, cb) {
  const cpsCb = (err, res) => {
    if (is.undef(err)) {
      cb(res);
    } else {
      cb(err, true);
    }
  }
  fn.apply(context, args.concat(cpsCb));
}

+function runAllEffect(env, effects, cb, { runEffect }) {
+  const keys = Object.keys(effects);
+  if (keys.length === 0) {
+    cb([]);
+    return;
+  }
+
+  const childCallbacks = createAllStyleChildCallbacks(effects, cb);
+  keys.forEach(key => {
+    runEffect(effects[key], childCallbacks[key])
+  })
+}

const effectRunnerMap = {
  [effectTypes.TAKE]: runTakeEffect,
  [effectTypes.PUT]: runPutEffect,
  [effectTypes.FORK]: runForkEffect,
  [effectTypes.CALL]: runCallEffect,
  [effectTypes.CPS]: runCPSEffect,
+  [effectTypes.ALL]: runAllEffect
};

export default effectRunnerMap;
```

10.支持cancel

- cancel 指示 middleware 取消之前的 fork 任务，cancel 是一个无阻塞 Effect

** 10.1 sagas.js #**

src\store\sagas.js

```
+import { put, takeEvery,call,cps,all,take,cancel,fork,delay} from '../redux-saga/effects';
//import { put, takeEvery,call,cps,all,take,delay,cancel,fork} 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);
+  yield take(actionTypes.STOP_ADD);
+  yield cancel(task);
+}

+export function* rootSaga() {
+  let result = yield addWatcher();
+  console.log('done',result);
+}
```

** 10.2 effectTypes.js #**

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';
```

** 10.3 effects.js #**

src\redux-saga\effects.js

```
import * as effectTypes from './effectTypes'
const makeEffect = (type, payload) => ({
  type,
  payload
})

export function take(pattern) {
  return makeEffect(effectTypes.TAKE, { pattern })
}

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

export function fork(fn) {
  return makeEffect(effectTypes.FORK, { fn })
}

export function takeEvery(pattern, saga) {
  function* takeEveryHelper() {
    while (true) {
      yield take(pattern);
      yield fork(saga);
    }
  }

  return fork(takeEveryHelper);
}

export function call(fn, ...args) {
  return makeEffect(effectTypes.CALL, { fn, args })
}
export function cps(fn, ...args) {
  return makeEffect(effectTypes.CPS, { fn, args })
}
export function all(effects) {
  return makeEffect(effectTypes.ALL, effects)
}

+export function cancel(task) {
+  return makeEffect(effectTypes.CANCEL, task)
+}

+export default function delayP(ms, val = true) {
+  const promise = new Promise(resolve => {
+    setTimeout(resolve, ms, val);
+  })
+  return promise
+}
+export const delay = call.bind(null, delayP)
```

** 10.4 effectRunnerMap.js #**

src\redux-saga\effectRunnerMap.js

```
import proc from "./proc";
import * as is  from './is';
import * as effectTypes from './effectTypes'
import {createAllStyleChildCallbacks} from './utils';
function runTakeEffect(env, {pattern}, cb) {
  const matcher = input => input.type
  env.channel.take(cb, matcher);
}

function runPutEffect(env, { action }, cb) {
  const result = env.dispatch(action);
  cb(result);
}

function runForkEffect(env, { fn }, cb) {
  const taskIterator = fn();
+  const task = proc(env, taskIterator);
+  cb(task);
}

function runCallEffect(env, { fn, args }, cb) {
  const result = fn.apply(null, args);
  if (is.promise(result)) {
    return result
      .then(data => cb(data))
      .catch(error => cb(error, true));
  }
  cb(result);
}

function runCPSEffect(env, {context,fn,args}, cb) {
  const cpsCb = (err, res) => {
    if (is.undef(err)) {
      cb(res);
    } else {
      cb(err, true);
    }
  }
  fn.apply(context, args.concat(cpsCb));
}

function runAllEffect(env, effects, cb, { runEffect }) {
  const keys = Object.keys(effects);
  if (keys.length
    cb([]);
    return;
  }

  const childCallbacks = createAllStyleChildCallbacks(effects, cb);
  keys.forEach(key => {
    runEffect(effects[key], childCallbacks[key])
  })
}

+function runCancelEffect(env, task, cb) {
+  task.cancel();
+  cb();
+}

const effectRunnerMap = {
  [effectTypes.TAKE]: runTakeEffect,
  [effectTypes.PUT]: runPutEffect,
  [effectTypes.FORK]: runForkEffect,
  [effectTypes.CALL]: runCallEffect,
  [effectTypes.CPS]: runCPSEffect,
  [effectTypes.ALL]: runAllEffect,
+ [effectTypes.CANCEL]: runCancelEffect
};

export default effectRunnerMap;
```

** 10.5 symbols.js #**

src\redux-saga\symbols.js

```
export const  TASK_CANCEL = Symbol('TASK_CANCEL');
```

** 10.6 proc.js #**

src\redux-saga\proc.js

```
import effectRunnerMap from './effectRunnerMap';
+import {TASK_CANCEL} from './symbols';
import * as is  from './is';

function resolvePromise(promise, cb) {
  promise.then(cb, function (error) {
    cb(error, true);
  });
}
export default function proc(env, iterator,cont) {
+ let task = {cancel:()=>next(TASK_CANCEL)};
  next();
  function next(arg, isErr) {
    let result;
    if (isErr) {
      result = iterator.throw(arg);
+    }else if(arg === TASK_CANCEL){
+      result = iterator.return(arg);
+    } else {
      result = iterator.next(arg);
    }
    if (!result.done) {
      runEffect(result.value, next)
    }else{
      cont&&cont(result.value);
    }
  }

  function runEffect(effect, next) {
    if (is.promise(effect)) {
      resolvePromise(effect, next);
    }else if(is.iterator(effect)){
      proc(env,effect,next);
    }else if (effect) {
      const effectRunner = effectRunnerMap[effect.type]
      effectRunner(env, effect.payload, next,{runEffect});
    } else {
      next();
    }
  }
+  return task;
}
```

** 10.7 action-types.js #**

src\store\action-types.js

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

** 10.8 actions.js #**

src\store\actions.js

```
import * as actionTypes from './action-types';
const actions = {
    add() {
        return {type:actionTypes.ASYNC_ADD}
    },
+    stop() {
+        return {type:actionTypes.STOP_ADD}
+    }
}
export default actions;
```

** 10.9 Counter.js #**

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 (

                {this.props.number}
+               stop

      )
  }
}
export default connect(
    state => state,
    actions
)(Counter);
```