# ES6-Generator函数和async函数

2017年07月05日 15:47:37

阅读数：525

# **ES6-异步函数**

Generator是一个状态机，它封装了多个内部状态。执行它将生成一个遍历器对象。

## **Generator有2个特征：**

声明时使用function\*

函数内部使用yield声明内部状态

调用函数后，函数并不执行，而且返回的也不是函数运行结果，而是一个指向内部状态的对象。

function\* helloWorldGenerator() {

yield 'hello';

yield 'world';

return 'ending';

}

var hw = helloWorldGenerator();

hw.next()// { value: 'hello', done: false }

hw.next()// { value: 'world', done: false }

hw.next()// { value: 'ending', done: true }

hw.next()// { value: undefined, done: true }

yield相当于暂停执行标识，next()相当于恢复执行标识，每调用一次next(),Generator函数才执行一次，直到遇到下一个yield或者return，当遇到yield或ruturn时，会在此暂停，并将其代表的值作为value返回。done是遍历标识符，表示是否完成遍历。

## **yield表达式**

* yield表达式只能用在Generator函数中
* yield表达式如果用在另一个表达式之中，必须放在园括号中
* yield表达式用作函数参数或放在赋值表达式的右边可以不加括号
* yield\* 表达式用来在一个Generator函数执行另一个Generator函数

function\* foo(a,b){

yield a;

yield b;

return a+b;

}

function\* demo() {

console.log('Hello' + yield); // SyntaxError

console.log('Hello' + yield 123); // SyntaxError

console.log('Hello' + (yield)); // OK

console.log('Hello' + (yield 123)); // OK

foo(yield 'a',yield 'b')//OK

let input = yield;

}

## **next方法**

yield表达式本身没有返回值，next方法可以带一个参数，该参数就被当作了yield表达式的对应变量。

function\* foo(x) {

var y = 2 \* (yield (x + 1));

var z = yield (y / 3);

return (x + y + z);

}

var a = foo(5);

a.next() // Object{value:6, done:false},第一次会将传入的参数赋值给x

a.next() // Object{value:NaN, done:false}，由于yield并没有返回值，所以y为undefined

a.next() // Object{value:NaN, done:true}

var b = foo(5);

b.next() // { value:6, done:false }

b.next(12) // { value:8, done:false }

b.next(13) // { value:42, done:true }

Generator返回的并不是一个对象，而是一个指针对象，调用next方法会将指针指向下一个阶段。每一个yield都表示一个阶段，跟下一个阶段并不影响。

## **Generator.prototype.return()**

Generator函数返回的遍历器对象，还有一个return方法，可以返回给定的值，并且终结遍历Generator函数。

function\* gen() {

yield 1;

yield 2;

yield 3;

}

var g = gen();

g.next() // { value: 1, done: false }

g.return('foo') // { value: "foo", done: true }

g.next() // { value: undefined, done: true }

Generator并不能自动执行，所以它需要一个自动执行器，常用的就是co函数库

# **async**

async函数是Generator函数的语法糖

区别如下:

* 将\*替换为了async且async提到了function的前面,表示函数中有异步操作
* 将yield替换为了await，表示紧跟后面的表达式需要等待结果
* async函数自带了执行器，不需要像Generator一样需要next方法才能执行
* yield命令后面只能是 Thunk 函数或 Promise 对象，而async函数的await命令后面，可以是Promise 对象和原始类型的值（数值、字符串和布尔值，但这时等同于同步操作）
* async返回的是一个Promise对象，Generator函数返回的是一个遍历器对象

var fs = require('fs');

var readFile = function (fileName) {

return new Promise(function (resolve, reject) {

fs.readFile(fileName, function(error, data) {

if (error) reject(error);

resolve(data);

});

});

};

var gen = function\* () {

var f1 = yield readFile('/etc/fstab');

var f2 = yield readFile('/etc/shells');

console.log(f1.toString());

console.log(f2.toString());

};

//async写法var asyncReadFile = async function () {

var f1 = await readFile('/etc/fstab');

var f2 = await readFile('/etc/shells');

console.log(f1.toString());

console.log(f2.toString());

};

async返回一个promise对象，所以return返回的值将作为then方法的参数。

async函数必须等内部所有的await命令后面的Promise对象执行完成才会执行then方法指定的回调函数。

# **await**

await返回一个Promise对象，如果不是，会被转换成一个立即执行的Promise对象,await将等待当前行执行完成才会继续之后的内容

async function f() {

await Promise.reject('出错了')

.catch(e => console.log(e));

return await Promise.resolve('hello world');

}

f().then(v => console.log(v))// 出错了// hello world