Generator 函数

简单来说，generator 函数就是一个分段式函数。

格式 : function\* haha(){

yield 'hello';

yield 'world';

......

return 'ending'; } ------ 最后一个是return

Yield 为产出的意思，也是该函数的一个暂停点

Generator 函数本身不会执行，需要加.next() ，

如：haha() ---------不输出

haha.next() -------’hello’

haha.next() -------’world’

....

haha.next() -------’ending’

以上列出的是value,还有一个参数done, 表示函数是否执行完毕

Generator 函数可以不用yield表达式，这时就变成了一个单纯的暂缓执行函数。

function\* f() {

console.log('执行了！')

}

f.next() ------执行

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}

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

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

var b = foo(5);

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

// 由于next方法的参数表示上一个yield表达式的返回值，所以在第一次使用next方法时，传递参数是无效的。

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

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

上面代码第一次调用b的next方法时，返回x+1的值6；

第二次调用next方法，将上一次yield表达式的值设为12，因此y等于24，返回y / 3的值8；

第三次调用next方法，将上一次yield表达式的值设为13，因此z等于13，这时x等于5，y等于24，所以return语句的值等于42。