Generator(生成器)

1、简介：可以把它理解成一个函数的内部状态的遍历器，每调用一次，函数的内部状态发生一次改变。

2、写法：一是，function 关键字与函数名之间有一个星号；

二是，函数体内部使用 yield 表达式，定义不同的内部状态

三，ES6 没有规定，function 关键字与函数名之间的星号，写在哪个位置

```
function * foo() {···}
function *foo() {···}
function* foo() {···}
function*foo(y) {···}
```

3、next 方法，使得指针移向下一个状态。每次调用 next 方法，内部指针就从函数头部或上一次停下来的地方开始执行，直到遇到下一个 yield 表达式（或 return 语句）为止

（1）yield 表达式：只有调用 next 方法才会遍历下一个内部状态，所以其实提供了一种可以暂停执行的函数。yield 表达式就是暂停标志。（再次调用 next 方法时，再继续往下执行，直到遇到下一个 yield 表达式。）

4、例：如下，代码定义了一个 Generator 函数 f，它内部有两个 yield 表达式（hello 和 world），即该函数有三个状态：hello，world 和 return 语句（结束执行）。

（1） 必须调用遍历器对象的 next 方法，使得指针移向下一个状态。也就是说，每次调用 next 方法，内部指针就从函数头部或上一次停下来的地方开始执行，直到遇到下一个 yield 表达式（或 return 语句）为止。

```
function* f() {
    yield 'hello';
    yield 'world';
    return 'ending';
}
let hw = f();

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

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

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

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

（2） 代码一共调用了四次 next 方法。

第一次调用，Generator 函数开始执行，直到遇到第一个 yield 表达式为止。next 方法返回一个对象，它的 value 属性就是当前 yield 表达式的值 hello，done 属性的值 false，表示遍历还没有结束。

第二次调用，Generator 函数从上次 yield 表达式停下的地方，一直执行到下一个 yield 表达式。next 方法返回的对象的 value 属性就是当前 yield 表达式的值 world，done 属性的值 false，表示遍历还没有结束。

第三次调用，Generator 函数从上次 yield 表达式停下的地方，一直执行到 return 语句（如果没有 return 语句，就执行到函数结束）。next 方法返回的对象的 value 属性，就是紧跟在 return 语句后面的表达式的值（如果没有 return 语句，则 value 属性的值为 undefined），done 属性的值 true，表示遍历已经结束。

第四次调用，此时 Generator 函数已经运行完毕，next 方法返回对象的 value 属性为 undefined，done 属性为 true。以后再调用 next 方法，返回的都是这个值。

注：调用 Generator 函数，返回一个遍历器对象，代表 Generator 函数的内部指针。以后，每次调用遍历器对象的 next 方法，就会返回一个有着 value 和 done 两个属性的对象。value 属性表示当前的内部状态的值，是 yield 表达式后面那个表达式的值；done 属性是一个布尔值，表示是否遍历结束。

慕课网 imooc.com