**ES6之Generator函数**

在学习ES6的过程中，个人认为Generator稍微会有些难以理解，其实Generator函数是ES6提供的一种异步编程解决方案，语法行为与传统函数完全不同。

我们从表面上看，Generator函数是一个普通函数，但是有两个特征：   
 1. function关键字与函数名之间有一个星号；   
 2. 函数体内部使用yield语句，定义不同的内部状态（yield语句在英语里的意思就是“产出”）。

**Generator的使用：（举个栗子）**

"use strict";

function\* fibonacci() {

yield 1;

yield 2;

}

var it = fibonacci();

console.log(it); // "Generator { }"

console.log(it.next()); // 1

console.log(it.next()); // 2

console.log(it.next()); //undefined

这个玩意儿如果运行的话，会返回一个Iterator实例， 然后再执行Iterator实例的**next()**方法， 那么这个函数才开始真正运行， 并把yield后面的值**包装成固定对象并返回**，直到运行到函数结尾， 最后再返回**undefined**；

**Yield**

Generator函数返回的Iterator运行的过程中，如果碰到了yield， 就会把yield后面的值返回， 此时函数相当于停止了， 下次再执行next()方法的时候， 函数又会从上次退出去的地方重新开始执行；

如果把**yield**和**return**一起使用的话， 那么return的值也会作为最后的返回值， 如果return语句后面还有yield， 那么这些yield不生效：

**Yield\***

yield\*这种语句让我们可以在Generator函数里面再套一个Generator， 当然你要在一个Generator里面调用另外的Generator需要使用: **yield\* 函数()** 这种语法， 都是套路啊：

例如：

function\* foo() {

yield 0;

yield 1;

}

function\* bar() {

yield 'x';

yield\* foo();

yield 'y';

}

for (let v of bar()){

console.log(v);//x,0,1,y

};

**Next()方法：**

Generator函数返回的Iterator执行next()方法以后， 返回值的结构为：

{

value : "value", //**value**为返回的值

done : false //**done**的值为一个布尔值， 如果Interator未遍历完毕， 他会返回false， 否则返回true；

}

**Next()方法的参数：**

如果给next方法传参数， 那么这个参数将会作为上一次**yield语句**的返回值 ，这个特性在异步处理中是非常重要的， 因为在执行异步代码以后， 有时候**需要**上一个异步的结果， 作为下次异步的参数， 如此循环：：

<script>

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 }

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

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

</script>

**Return 方法：**

如果执行Iterator的return()方法， 那么这个迭代器的返回会被强制设置为迭代完毕， 执行return()方法的参数就是这个Iterator的返回值，此时done的状态也为true：

<script>

function\* gen() {

yield 0;

yield 1;

yield 2;

yield 3;

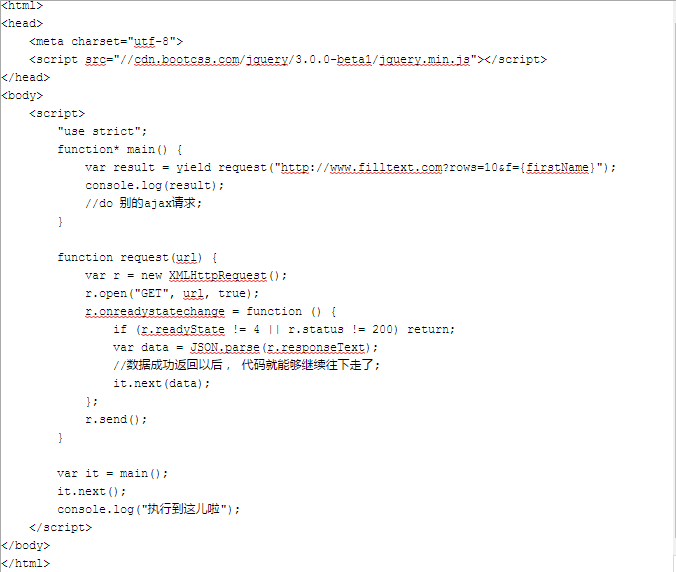
};

let g = gen();

console.log(g.return("heheda")); //输出：{ value: 'heheda', done: true }

</script>

**实际使用：**



以上代码中的**console.log("执行到这儿啦");**先被执行了， 然后才出现了ajax的返回结果， 也就说明了Generator函数是异步的了；

