博学谷
www.boxuegu.com

第十二章 Generator函数

一样的在线教育，不一样的教学品质

目录 Contents

Generator函数基本使用

Generator函数也称为生成器函数，可以用来生成迭代器，也就是前面所学习的遍历器；即可以通过for...of来遍历Generator函数；且Generator函数提供了一种异步编程的解决方案。

生成器函数和普通函数不一样，普通函数是一旦调用就会执行完毕，但是**生成器函数中间可以暂停。**

```
function* go() {
        console.log(1);
        let a = yield 'a';
        console.log(2);
        let b = yield a;
        console.log(3);
        return b;
    }
    let it = go();
    let r1 = it.next();
    console.log(r1);
    let r2 = it.next('b的值');
    console.log(r2);
    let r3 = it.next()
    console.log(r3)
    let r4 = it.next('c的值');
    console.log(r4);
```

目录 Contents

next方法参数

给next方法添加相应的参数，该参数会被当作上一条yield语句的返回值。

示例：判断其对应的输出结果；

```
function* test(num) {
            let x = 3 * (yield(num + 1));
            let y = yield(x / 3);
            return (x + y + num);
        }
        let n = test(6);
        console.log(n.next());
        console.log(n.next());
        console.log(n.next());
```

输出结果如下：

```
{value: 7, done: false}
{value: NaN, done: false}
{value: NaN, done: true}
```

next方法参数

将程序修改成如下的形式：

```
function* test(num) {
    let x = 3 * (yield(num + 1));
    let y = yield(x / 3);
    return (x + y + num);
}
let n = test(6);
console.log(n.next());
console.log(n.next(3));
console.log(n.next(3));
```

输出结果如下：

```
{value: 7, done: false}
{value: 3, done: false}
{value: 18, done: true}
```

注意：

由于next方法的参数表示上一条yield语句的返回值，所以**第一次使用next方法时不能带参数**。

即**第一次使用next方法时是用来启动遍历器对象的**。

目录 Contents

for...of循环

for...of循环可以自动遍历Generator函数，且此时不再需要调用next方法。

```
function* test() {
            yield 1;
            yield 2;
            yield 3;
            yield 4;
            yield 5;
            return 6;
        }
        for (let v of test()) {
            console.log(v);
        }
```

注意：

一旦next( )方法返回的对象的done属性为true，for...of循环就会终止，且不包含该返回对象；

所以，上面的return语句不在for...of循环中。

目录 Contents

yield* 语句

如果在Generator函数内部调用一个Generator函数，默认情况下是没有效果的。

```
function* test() {
            yield 'a';
            yield 'b';
        }
        function* test1() {
            yield 'x';
            test();
            yield 'y';
        }
        for (let v of test1()) {
            console.log(v);
        }
```

yield* 语句

要解决内部调用无效的问题，用 yield* 语句，用来在一个Generator函数中执行另外一个Generator函数。

```
function* test() {
            yield 'a';
            yield 'b';
        }
        function* test1() {
            yield 'x';
            yield* test();
            yield 'y';
        }
        for (let v of test1()) {
            console.log(v);
        }
```

目录 Contents

关于Generator函数中的this问题

下面的代码，是否有错误？

```
function* Person() {
            yield this.name = 'zs';
            yield this.age = 18;
        }
        let person = new Person();
        console.log(person.name);
```

原因：Person既是构造函数，又是Generator函数，所以，使用new命令无法创建Person的对象。

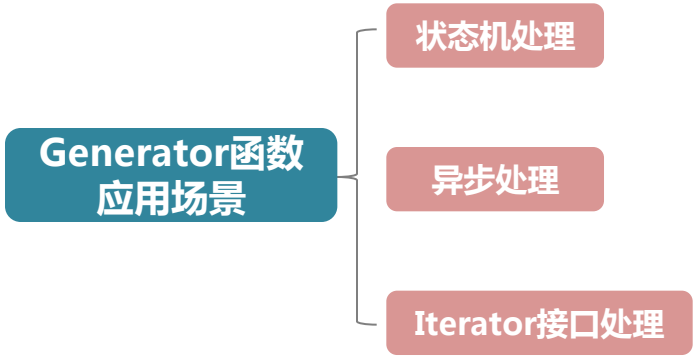
如何解决上述问题呢？

创建一个空对象；使用bind方法绑定Generator函数内部的this；空对象就是Generator函数的实例对象。

```
function* Person() {
            yield this.name = 'zs';
            yield this.age = 18;
        }
        let person = {}
        let obj = Person.bind(person)();
        console.log(obj.next());
```

目录 Contents

Generator函数应用场景



1.状态机处理

案例: 单击按钮实现图片切换

传统做法：

```
let button = document.getElementById('btn') //找到按钮
       let mm = document.getElementById('mv') //找到img标签
       let flag = 0
       button.onclick = function() {
           //将img标签的src属性的值，换成另外一张图片的地址.
           if (flag === 0) {
               mm.src = 'images/b.jpg';
               flag = 1;
           } else {
               mm.src = 'images/a.jpg';
               flag = 0;
           }
       }
```

1.状态机处理

案例: 单击按钮实现图片切换

使用 Generator函数处理：

```
let button = document.getElementById('btn') //找到按钮
       let mm = document.getElementById('mv') //找到img标签
       let it = f(0);
       button.onclick = function() {
           it.next();
       }

       function* f(flag) {
           while (true) {
               mm.src = 'images/b.jpg';
               yield flag;
               mm.src = 'images/a.jpg';
               yield flag;

           }
       }
```

2.异步处理

Generator函数提供了一种异步处理的解决方案，而AJAX是典型的异步操作。

```
function* main() {
  let result = yield request("http://xxx.com/api");
  let resp = JSON.parse(result);
    console.log(resp.value);
}

function request(url) {
  makeAjaxCall(url, function(response){
    it.next(response);
  });
}

let it = main();
it.next();
```

3.Iterator接口处理

由于JavaScript对象没有遍历的接口，无法使用for...of进行遍历；可以通过Generator函数添加接口。

```
let user = {
            name: 'zs',
            age: 18
        }

        function* test(obj) {
            let keys = Reflect.ownKeys(obj);
            for (let key of keys) {
                yield [key, obj[key]];
            }
        }
        for (let item of test(user)) {
            console.log(item);
        }
```

一样的在线教育，不一样的教学品质