

# Python3 迭代器与生成器

## Python3 迭代器与生成器

迭代器

创建一个迭代器

StopIteration

生成器

## 迭代器

迭代是Python最强大的功能之一，是访问集合元素的一种方式。

迭代器是一个可以记住遍历的位置的对象。

**迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束。迭代器只能往前不会后退。**

迭代器有两个基本的方法：`iter()` 和 `next()`。

**字符串，列表或元组对象都可用于创建迭代器：**

```
list=[1,2,3,4]
it = iter(list)    # 创建迭代器对象
print (next(it))   # 输出迭代器的下一个元素
#输出 1
print (next(it))
#输出 2
```

迭代器对象可以使用常规for语句进行遍历：

```
#!/usr/bin/python3

list = [1, 2, 3, 4]
it = iter(list)    # 创建迭代器对象
for x in it:
    print (x, end=" ")
```

执行以上程序，输出结果如下：

```
1 2 3 4
```

也可以使用 `next()` 函数：

```
#!/usr/bin/python3

import sys          # 引入 sys 模块

list = [1, 2, 3, 4]
it = iter(list)     # 创建迭代器对象

while True:
    try:
        print (next(it))
    except StopIteration:
        sys.exit()
```

执行以上程序，输出结果如下：

```
1
2
3
4
```

## 创建一个迭代器

把一个类作为一个迭代器使用需要在类中实现两个方法 `__iter__()` 与 `__next__()`。

如果你已经了解的面向对象编程，都支持类都有一个构造函数，Python 的构造函数为 `__init__()`，它会在对象初始化的时候执行。更多内容查阅：Python3 面向对象

`__iter__()` 方法返回一个特殊的迭代器对象，这个迭代器对象实现了 `__next__()` 方法并通过 `StopIteration` 异常标识迭代的完成。

`__next__()` 方法（Python 2 里是 `next()`）会返回下一个迭代器对象。

创建一个返回数字的迭代器，初始值为 1，逐步递增 1：

```
class MyNumbers:
    def __iter__(self):
        self.a = 1
        return self

    def __next__(self):
        x = self.a
        self.a += 1
        return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
```

```
print(next(myiter))
print(next(myiter))
```

执行输出结果为：

```
1
2
3
4
5
```

## StopIteration

StopIteration 异常用于标识迭代的完成，防止出现无限循环的情况，在 `__next__()` 方法中我们可以设置在完成指定循环次数后触发 StopIteration 异常来结束迭代。

在 20 次迭代后停止执行：

```
class MyNumbers:
    def __iter__(self):
        self.a = 1
        return self

    def __next__(self):
        if self.a <= 20:
            x = self.a
            self.a += 1
            return x
        else:
            raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
    print(x)
```

执行输出结果为：

```
1
2
3
4
5
6
7
8
9
10
11
```

12  
13  
14  
15  
16  
17  
18  
19  
20

## 生成器

在 Python 中，使用了 **yield** 的函数被称为**生成器 (generator)**。

跟普通函数不同的是，**生成器是一个返回迭代器的函数，只能用于迭代操作，更简单点理解生成器就是一个迭代器。**

**在调用生成器运行的过程中，每次遇到 yield 时函数会暂停并保存当前所有的运行信息，返回 yield 的值，并在下一次执行 next() 方法时从当前位置继续运行。**

调用一个生成器函数，返回的是一个迭代器对象。

以下实例使用 yield 实现斐波那契数列：

```
#!/usr/bin/python3

import sys

def fibonacci(n):          # 生成器函数 - 斐波那契
    a, b, counter = 0, 1, 0
    while True:
        if (counter > n):
            return
        yield a
        a, b = b, a + b
        counter += 1

f = fibonacci(10)          # f 是一个迭代器，由生成器返回生成

while True:
    try:
        print (next(f), end=" ")
    except StopIteration:
        sys.exit()
```

执行以上程序，输出结果如下：

0 1 1 2 3 5 8 13 21 34 55