4.6 带有外部状态的生成器函数¶

问题¶

你想定义一个生成器函数，但是它会调用某个你想暴露给用户使用的外部状态值。

解决方案¶

如果你想让你的生成器暴露外部状态给用户， 别忘了你可以简单的将它实现为一个类，然后把生成器函数放到 `__iter__()` 方法中过去。比如：

```
from collections import deque

class linehistory:
    def __init__(self, lines, histlen=3):
        self.lines = lines
        self.history = deque(maxlen=histlen)

    def __iter__(self):
        for lineno, line in enumerate(self.lines, 1):
            self.history.append((lineno, line))
            yield line

    def clear(self):
        self.history.clear()
```

为了使用这个类，你可以将它当做是一个普通的生成器函数。 然而，由于可以创建一个实例对象，于是你可以访问内部属性值， 比如 `history` 属性或者是 `clear()` 方法。代码示例如下：

```
with open('somefile.txt') as f:
    lines = linehistory(f)
    for line in lines:
        if 'python' in line:
            for lineno, hline in lines.history:
                print('{}:{}'.format(lineno, hline), end='')
```

讨论¶

关于生成器，很容易掉进函数无所不能的陷阱。 如果生成器函数需要跟你的程序其他部分打交道的话(比如暴露属性值，允许通过方法调用来控制等等)， 可能会导致你的代码异常的复杂。 如果是这种情况的话，可以考虑使用上面介绍的定义类的方式。 在 `__iter__()` 方法中定义你的生成器不会改变你任何的算法逻辑。 由于它是类的一部分，所以允许你定义各种属性和方法来供用户使用。

一个需要注意的小地方是，如果你在迭代操作时不使用for循环语句，那么你得先调用 `iter()` 函数。比如：

```
>>> f = open('somefile.txt')
>>> lines = linehistory(f)
>>> next(lines)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: 'linehistory' object is not an iterator

>>> # Call iter() first, then start iterating
>>> it = iter(lines)
>>> next(it)
'hello world\n'
>>> next(it)
'this is a test\n'
>>>
```