4.16 迭代器代替while无限循环¶

问题¶

你在代码中使用 `while` 循环来迭代处理数据，因为它需要调用某个函数或者和一般迭代模式不同的测试条件。 能不能用迭代器来重写这个循环呢？

解决方案¶

一个常见的IO操作程序可能会想下面这样：

```
CHUNKSIZE = 8192

def reader(s):
    while True:
        data = s.recv(CHUNKSIZE)
        if data == b'':
            break
        process_data(data)
```

这种代码通常可以使用 `iter()` 来代替，如下所示：

```
def reader2(s):
    for chunk in iter(lambda: s.recv(CHUNKSIZE), b''):
        pass
        # process_data(data)
```

如果你怀疑它到底能不能正常工作，可以试验下一个简单的例子。比如：

```
>>> import sys
>>> f = open('/etc/passwd')
>>> for chunk in iter(lambda: f.read(10), ''):
...     n = sys.stdout.write(chunk)
...
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
_uucp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico
...
>>>
```

讨论¶

`iter` 函数一个鲜为人知的特性是它接受一个可选的 `callable` 对象和一个标记(结尾)值作为输入参数。 当以这种方式使用的时候，它会创建一个迭代器， 这个迭代器会不断调用 `callable` 对象直到返回值和标记值相等为止。

这种特殊的方法对于一些特定的会被重复调用的函数很有效果，比如涉及到I/O调用的函数。 举例来讲，如果你想从套接字或文件中以数据块的方式读取数据，通常你得要不断重复的执行 `read()` 或 `recv()`， 并在后面紧跟一个文件结尾测试来决定是否终止。这节中的方案使用一个简单的 `iter()` 调用就可以将两者结合起来了。 其中 `lambda` 函数参数是为了创建一个无参的 `callable` 对象，并为 `recv` 或 `read()` 方法提供了 `size` 参数。