4.13 创建数据处理管道¶

问题¶

你想以数据管道(类似Unix管道)的方式迭代处理数据。 比如，你有个大量的数据需要处理，但是不能将它们一次性放入内存中。

解决方案¶

生成器函数是一个实现管道机制的好办法。 为了演示，假定你要处理一个非常大的日志文件目录：

```
foo/
   access-log-012007.gz
   access-log-022007.gz
   access-log-032007.gz
   ...
   access-log-012008
bar/
   access-log-092007.bz2
   ...
   access-log-022008
```

假设每个日志文件包含这样的数据：

```
124.115.6.12 - - [10/Jul/2012:00:18:50 -0500] "GET /robots.txt ..." 200 71
210.212.209.67 - - [10/Jul/2012:00:18:51 -0500] "GET /ply/ ..." 200 11875
210.212.209.67 - - [10/Jul/2012:00:18:51 -0500] "GET /favicon.ico ..." 404 369
61.135.216.105 - - [10/Jul/2012:00:20:04 -0500] "GET /blog/atom.xml ..." 304 -
...
```

为了处理这些文件，你可以定义一个由多个执行特定任务独立任务的简单生成器函数组成的容器。就像这样：

```
import os
import fnmatch
import gzip
import bz2
import re

def gen_find(filepat, top):
    '''
    Find all filenames in a directory tree that match a shell wildcard pattern
    '''
    for path, dirlist, filelist in os.walk(top):
        for name in fnmatch.filter(filelist, filepat):
            yield os.path.join(path,name)

def gen_opener(filenames):
    '''
    Open a sequence of filenames one at a time producing a file object.
    The file is closed immediately when proceeding to the next iteration.
    '''
    for filename in filenames:
        if filename.endswith('.gz'):
            f = gzip.open(filename, 'rt')
        elif filename.endswith('.bz2'):
```

```
            f = bz2.open(filename, 'rt')
        else:
            f = open(filename, 'rt')
        yield f
        f.close()

def gen_concatenate(iterators):
    '''
    Chain a sequence of iterators together into a single sequence.
    '''
    for it in iterators:
        yield from it

def gen_grep(pattern, lines):
    '''
    Look for a regex pattern in a sequence of lines
    '''
    pat = re.compile(pattern)
    for line in lines:
        if pat.search(line):
            yield line
```

现在你可以很容易的将这些函数连起来创建一个处理管道。 比如，为了查找包含单词python的所有日志行，你可以这样做：

```
lognames = gen_find('access-log*', 'www')
files = gen_opener(lognames)
lines = gen_concatenate(files)
pylines = gen_grep('(?i)python', lines)
for line in pylines:
    print(line)
```

如果将来的时候你想扩展管道，你甚至可以在生成器表达式中包装数据。 比如，下面这个版本计算出传输的字节数并计算其总和。

```
lognames = gen_find('access-log*', 'www')
files = gen_opener(lognames)
lines = gen_concatenate(files)
pylines = gen_grep('(?i)python', lines)
bytecolumn = (line.rsplit(None,1)[1] for line in pylines)
bytes = (int(x) for x in bytecolumn if x != '-')
print('Total', sum(bytes))
```

讨论¶

以管道方式处理数据可以用来解决各类其他问题，包括解析，读取实时数据，定时轮询等。

为了理解上述代码，重点是要明白 `yield` 语句作为数据的生产者而 `for` 循环语句作为数据的消费者。 当这些生成器被连在一起后，每个 `yield` 会将一个单独的数据元素传递给迭代处理管道的下一阶段。 在例子最后部分， `sum()` 函数是最终的程序驱动者，每次从生成器管道中提取出一个元素。

这种方式一个非常好的特点是每个生成器函数很小并且都是独立的。这样的话就很容易编写和维护它们了。 很多时候，这些函数如果比较通用的话可以在其他场景重复使用。 并且最终将这些组件组合起来的代码看上去非常简单，也很容易理解。

使用这种方式的内存效率也不得不提。上述代码即便是在一个超大型文件目录中也能工作的很好。 事实上，由于使用了迭代方式处理，代码运行过程中只需要很小很小的内存。

在调用 `gen_concatenate()` 函数的时候你可能会有些不太明白。 这个函数的目的是将输入序列拼接成一个很长的行序列。 `itertools.chain()` 函数同样有类似的功能，但是它需要将所有可迭代对象最为参数传入。 在上面这个例子中，你可能会写类似这样的语句 `lines = itertools.chain(*files)` ， 这将导致 `gen_opener()` 生成器被提前全部消费掉。 但由于 `gen_opener()` 生成器每次生成一个打开过的文件， 等到下一个迭代步骤时文件就关闭了，因此 `chain()` 在这里不能这样使用。 上面的方案可以避免这种情况。

`gen_concatenate()` 函数中出现过 `yield from` 语句，它将 `yield` 操作代理到父生成器上去。 语句 `yield from it` 简单的返回生成器 `it` 所产生的所有值。 关于这个我们在4.14小节会有更进一步的描述。

最后还有一点需要注意的是，管道方式并不是万能的。 有时候你想立即处理所有数据。 然而，即便是这种情况，使用生成器管道也可以将这类问题从逻辑上变为工作流的处理方式。

David Beazley 在他的 Generator Tricks for Systems Programmers 教程中对于这种技术有非常深入的讲解。可以参考这个教程获取更多的信息。