2.14 合并拼接字符串¶

问题¶

你想将几个小的字符串合并为一个大的字符串

解决方案¶

如果你想要合并的字符串是在一个序列或者 iterable 中，那么最快的方式就是使用 join() 方法。比如：

```
>>> parts = ['Is', 'Chicago', 'Not', 'Chicago?']
>>> ' '.join(parts)
'Is Chicago Not Chicago?'
>>> ','.join(parts)
'Is,Chicago,Not,Chicago?'
>>> ''.join(parts)
'IsChicagoNotChicago?'
>>>
```

初看起来，这种语法看上去会比较怪，但是 join() 被指定为字符串的一个方法。 这样做的部分原因是你想去连接的对象可能来自各种不同的数据序列(比如列表，元组，字典，文件，集合或生成器等)， 如果在所有这些对象上都定义一个 join() 方法明显是冗余的。 因此你只需要指定你想要的分割字符串并调用他的 join() 方法去将文本片段组合起来。

如果你仅仅只是合并少数几个字符串，使用加号(+)通常已经足够了：

```
>>> a = 'Is Chicago'
>>> b = 'Not Chicago?'
>>> a + ' ' + b
'Is Chicago Not Chicago?'
>>>
```

加号(+)操作符在作为一些复杂字符串格式化的替代方案的时候通常也工作的很好，比如：

```
>>> print('{} {}'.format(a,b))
Is Chicago Not Chicago?
>>> print(a + ' ' + b)
Is Chicago Not Chicago?
>>>
```

如果你想在源码中将两个字面字符串合并起来，你只需要简单的将它们放到一起，不需要用加号(+)。比如：

```
>>> a = 'Hello' 'World'
>>> a
'HelloWorld'
>>>
```

讨论¶

字符串合并可能看上去并不需要用一整节来讨论。 但是不应该小看这个问题，程序员通常在字符串格式化的时候因为选择不当而给应用程序带来严重性能损失。

最重要的需要引起注意的是，当我们使用加号(+)操作符去连接大量的字符串的时候是非常低效率的， 因为加号连接会引起内存复制以及垃圾回收操作。 特别的，你永远都不应像下面这样写字符串连接代码：

```
s = ''
for p in parts:
    s += p
```

这种写法会比使用 join() 方法运行的要慢一些，因为每一次执行+=操作的时候会创建一个新的字符串对象。 你最好是先收集所有的字符串片段然后再将它们连接起来。

一个相对比较聪明的技巧是利用生成器表达式(参考1.19小节)转换数据为字符串的同时合并字符串，比如：

```
>>> data = ['ACME', 50, 91.1]
>>> ','.join(str(d) for d in data)
'ACME,50,91.1'
>>>
```

同样还得注意不必要的字符串连接操作。有时候程序员在没有必要做连接操作的时候仍然多此一举。比如在打印的时候：

```
print(a + ':' + b + ':' + c) # Ugly
print(':'.join([a, b, c])) # Still ugly
print(a, b, c, sep=':') # Better
```

当混合使用I/O操作和字符串连接操作的时候，有时候需要仔细研究你的程序。 比如，考虑下面的两端代码片段：

```
# Version 1 (string concatenation)
f.write(chunk1 + chunk2)

# Version 2 (separate I/O operations)
f.write(chunk1)
f.write(chunk2)
```

如果两个字符串很小，那么第一个版本性能会更好些，因为I/O系统调用天生就慢。 另外一方面，如果两个字符串很大，那么第二个版本可能会更加高效， 因为它避免了创建一个很大的临时结果并且要复制大量的内存块数据。 还是那句话，有时候是需要根据你的应用程序特点来决定应该使用哪种方案。

最后谈一下，如果你准备编写构建大量小字符串的输出代码， 你最好考虑下使用生成器函数，利用yield语句产生输出片段。比如：

```
def sample():
    yield 'Is'
    yield 'Chicago'
    yield 'Not'
    yield 'Chicago?'
```

这种方法一个有趣的方面是它并没有对输出片段到底要怎样组织做出假设。 例如，你可以简单的使用 `join()` 方法将这些片段合并起来：

```
text = ''.join(sample())
```

或者你也可以将字符串片段重定向到I/O：

```
for part in sample():
    f.write(part)
```

再或者你还可以写出一些结合I/O操作的混合方案：

```
def combine(source, maxsize):
    parts = []
    size = 0
    for part in source:
        parts.append(part)
        size += len(part)
        if size > maxsize:
            yield ''.join(parts)
            parts = []
            size = 0
    yield ''.join(parts)

# 结合文件操作
with open('filename', 'w') as f:
    for part in combine(sample(), 32768):
        f.write(part)
```

这里的关键点在于原始的生成器函数并不需要知道使用细节，它只负责生成字符串片段就行了。