2.20 字节字符串上的字符串操作¶

问题¶

你想在字节字符串上执行普通的文本操作(比如移除，搜索和替换)。

解决方案¶

字节字符串同样也支持大部分和文本字符串一样的内置操作。比如：

```
>>> data = b'Hello World'
>>> data[0:5]
b'Hello'
>>> data.startswith(b'Hello')
True
>>> data.split()
[b'Hello', b'World']
>>> data.replace(b'Hello', b'Hello Cruel')
b'Hello Cruel World'
>>>
```

这些操作同样也适用于字节数组。比如：

```
>>> data = bytearray(b'Hello World')
>>> data[0:5]
bytearray(b'Hello')
>>> data.startswith(b'Hello')
True
>>> data.split()
[bytearray(b'Hello'), bytearray(b'World')]
>>> data.replace(b'Hello', b'Hello Cruel')
bytearray(b'Hello Cruel World')
>>>
```

你可以使用正则表达式匹配字节字符串，但是正则表达式本身必须也是字节串。比如：

```
>>>
>>> data = b'FOO:BAR,SPAM'
>>> import re
>>> re.split('[:,]',data)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "/usr/local/lib/python3.3/re.py", line 191, in split
return _compile(pattern, flags).split(string, maxsplit)
TypeError: can't use a string pattern on a bytes-like object
>>> re.split(b'[:,]',data) # Notice: pattern as bytes
[b'FOO', b'BAR', b'SPAM']
>>>
```

讨论¶

大多数情况下，在文本字符串上的操作均可用于字节字符串。 然而，这里也有一些需要注意的不同点。首先，字节字符串的索引操作返回整数而不是单独字符。比如：

```
>>> a = 'Hello World' # Text string
>>> a[0]
'H'
>>> a[1]
'e'
>>> b = b'Hello World' # Byte string
>>> b[0]
72
>>> b[1]
101
>>>
```

这种语义上的区别会对于处理面向字节的字符数据有影响。

第二点，字节字符串不会提供一个美观的字符串表示，也不能很好的打印出来，除非它们先被解码为一个文本字符串。比如：

```
>>> s = b'Hello World'
>>> print(s)
b'Hello World' # Observe b'...'
>>> print(s.decode('ascii'))
Hello World
>>>
```

类似的，也不存在任何适用于字节字符串的格式化操作：

```
>>> b'%10s %10d %10.2f' % (b'ACME', 100, 490.1)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for %: 'bytes' and 'tuple'
>>> b'{} {} {}'.format(b'ACME', 100, 490.1)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: 'bytes' object has no attribute 'format'
>>>
```

如果你想格式化字节字符串，你得先使用标准的文本字符串，然后将其编码为字节字符串。比如：

```
>>> '{:10s} {:10d} {:10.2f}'.format('ACME', 100, 490.1).encode('ascii')
b'ACME 100 490.10'
>>>
```

最后需要注意的是，使用字节字符串可能会改变一些操作的语义，特别是那些跟文件系统有关的操作。 比如，如果你使用一个编码为字节的文件名，而不是一个普通的文本字符串，会禁用文件名的编码/解码。比如：

```
>>> # Write a UTF-8 filename
>>> with open('jalape\xf1o.txt', 'w') as f:
...     f.write('spicy')
...
>>> # Get a directory listing
>>> import os
>>> os.listdir('.') # Text string (names are decoded)
['jalapeño.txt']
>>> os.listdir(b'.') # Byte string (names left as bytes)
[b'jalapen\xcc\x83o.txt']
>>>
```

注意例子中的最后部分给目录名传递一个字节字符串是怎样导致结果中文件名以未解码字节返回的。 在目录中的文件名包含原始的UTF-8编码。 参考5.15小节获取更多文件名相关的内容。

最后提一点，一些程序员为了提升程序执行的速度会倾向于使用字节字符串而不是文本字符串。 尽管操作字节字符串确实会比文本更加高效(因为处理文本固有的Unicode相关开销)。 这样做通常会导致非常杂乱的代码。你会经常发现字节字符串并不能和Python的其他部分工作的很好， 并且你还得手动处理所有的编码/解码操作。 坦白讲，如果你在处理文本的话，就直接在程序中使用普通的文本字符串而不是字节字符串。不做死就不会死！