链滴

Python-CookBook：31、文本过滤和清理

作者：zhaolixiang
原文链接：https://ld246.com/article/1566205236150
来源网站：链滴

问题

某些无聊的脚本小子在Web页面表单中填入了“pýt̂höñ”这样的文本，我们想以某种方式将其清理。

解决方案

文本过滤和清理所涵盖的范围非常广泛，涉及文本解析和数据处理方面的问题。在非常简单的层次上我们可能会用基本的字符串函数（例如str.upper()和str.lower()）将文本转换为标准形式。简单的替操作可通过str.replace()或re.sub()来完成，它们把重点放在移除或修改特定的字符序列上。也可以利unicodedata.normalize()来规范化文本。

然而我们可能想更进一步。比方说也许想清除整个范围内的字符，或者去掉音符标志。要完成这些任，可以使用常被忽视的str.translate()方法。为了说明其用法，假设有如下这段混乱的字符串：

```
s='pyt̂hon\fis\tawesome\r\n'
print(s)
```

输出：

```
pyt̂honis    awesome
```

第一步是清理空格。要做到这步，先建立一个小型的转换表，然后使用translate()方法：

```
remap={
    ord('\t'):' ',
    ord('\f'):' ',
    ord('\r'):None,
}
a=s.translate(remap)

print(a)
```

输出：

```
pyt̂hon is awesome
```

可以看到，类似\t和\f这样的空格符已经被重新映射成一个单独的空格。回车符\r已经完全被删除掉了。

可以利用这种重新映射的思想进一步构建出更加庞大的转换表。例如，我们把所有的Unicode组合字都去掉：

```
import unicodedata
import sys
cmb_chars=dict.fromkeys(c for c in range(sys.maxunicode) if unicodedata.combining(chr(c)))

s='pyt̂hon\fis\tawesome\r\n'
remap={
    ord('\t'):' ',
    ord('\f'):' ',
    ord('\r'):None,
```

```
}
a=s.translate(remap)
b=unicodedata.normalize('NFD',a)
print(b)

print(b.translate(cmb_chars))
```

输出：

```
pyth□on is awesome

python is awesome
```

在这个例子中，我们使用dict.fromkeys()方法构建了一个将每个Unicode组合字符都映射为None的典。

原始输入会通过unicodedata.normalize()方法转换为分离形式，然后再通过translate()方法删除所的重音符号。我们也可以利用相似的技术来去掉其他类型的字符（例如控制字符）。

下面来看另一个例子。这里有一张转换表将所有的Unicode十进制数字字符映射为它们对应的ASCII本：

```
import unicodedata
import sys

digimap={c:ord('0')+unicodedata.digit(chr(c))
        for c in range(sys.maxunicode)
        if unicodedata.category(chr(c))=='Nd'}

print(len(digimap))

x='\u0661\u0662\u0663'
print(x.translate(digimap))
```

输出：

```
610
123
```

另一种用来清理文本的技术涉及I/O解码和编码函数。大致思路是首先对文本做初步的清理，然后通结合encode()和decode()操作来修改或清理文本。示例如下：

```
import unicodedata

a='pyt̂hon is awesome\n'
b=unicodedata.normalize('NFD',a)
print(b.encode('ascii','ignore').decode('ascii'))
```

输出：

```
python is awesome
```

这里的normalize()方法先对原始文本做分解操作。后续的ASCII编码/解码只是简单地一次性丢弃所有需要的字符。很显然，这种方法只有当我们的最终目标就是ASCII形式的文本时才有用。

讨论

文本过滤和清理的一个主要问题就是运行时的性能。一般来说操作越简单，运行得就越快。对于简单替换操作，用str.replace()通常是最快的方式——即使必须多次调用它也是如此。比方说如果要清理空格符，可以编写如下的代码：

```
def clean_spaces(s):
    s = s.replace('\r', '')
    s = s.replace('\t', ' ')
    s = s.replace('\f', ' ')
return s
```

如果试着调用它，就会发现这比使用translate()或者正则表达式的方法要快得多。

另一方面，如果需要做任何高级的操作，比如字符到字符的重映射或删除，那么translate()方法还是常快的。

从整体来看，我们应该在具体的应用中去进一步揣摩性能方面的问题。不幸的是，想在技术上给出一“放之四海而皆准”的建议是不可能的，所以应该尝试多种不同的方法，然后做性能统计分析。

尽管本节的内容主要关注的是文本，但类似的技术也同样适用于字节对象（byte），这包括简单的替、翻译和正则表达式。