2.12 审查清理文本字符串¶

问题¶

一些无聊的幼稚黑客在你的网站页面表单中输入文本”pýt͡hön”，然后你想将这些字符清理掉。

解决方案¶

文本清理问题会涉及到包括文本解析与数据处理等一系列问题。 在非常简单的情形下，你可能会选择使用字符串函数(比如 `str.upper()` 和 `str.lower()`)将文本转为标准格式。 使用 `str.replace()` 或者 `re.sub()` 的简单替换操作能删除或者改变指定的字符序列。 你同样还可以使用2.9小节的 `unicodedata.normalize()` 函数将unicode文本标准化。

然后，有时候你可能还想在清理操作上更进一步。比如，你可能想消除整个区间上的字符或者去除变音符。 为了这样做，你可以使用经常会被忽视的 `str.translate()` 方法。 为了演示，假设你现在有下面这个凌乱的字符串：

```
>>> s = 'pýt͡höñ\fis\tawesome\r\n'
>>> s
'pýt͡höñ\x0cis\tawesome\r\n'
>>>
```

第一步是清理空白字符。为了这样做，先创建一个小的转换表格然后使用 `translate()` 方法：

```
>>> remap = {
...     ord('\t') : ' ',
...     ord('\f') : ' ',
...     ord('\r') : None # Deleted
... }
>>> a = s.translate(remap)
>>> a
'pýt͡höñ is awesome\n'
>>>
```

正如你看的那样，空白字符 `\t` 和 `\f` 已经被重新映射到一个空格。回车字符r直接被删除。

你可以以这个表格为基础进一步构建更大的表格。比如，让我们删除所有的和音符：

```
>>> import unicodedata
>>> import sys
>>> cmb_chrs = dict.fromkeys(c for c in range(sys.maxunicode)
...                          if unicodedata.combining(chr(c)))
...
>>> b = unicodedata.normalize('NFD', a)
>>> b
'pýt͡höñ is awesome\n'
>>> b.translate(cmb_chrs)
'python is awesome\n'
>>>
```

上面例子中，通过使用 `dict.fromkeys()` 方法构造一个字典，每个Unicode和音符作为键，对应的值全部为 `None` 。

然后使用 `unicodedata.normalize()` 将原始输入标准化为分解形式字符。 然后再调用 `translate` 函数删除所有重音符。 同样的技术也可以被用来删除其他类型的字符(比如控制字符等)。

作为另一个例子，这里构造一个将所有Unicode数字字符映射到对应的ASCII字符上的表格：

```
>>> digitmap = { c: ord('0') + unicodedata.digit(chr(c))
...         for c in range(sys.maxunicode)
...         if unicodedata.category(chr(c)) == 'Nd' }
...
>>> len(digitmap)
460
>>> # Arabic digits
>>> x = '\u0661\u0662\u0663'
>>> x.translate(digitmap)
```

```
'123'
>>>
```

另一种清理文本的技术涉及到I/O解码与编码函数。这里的思路是先对文本做一些初步的清理， 然后再结合 `encode()` 或者 `decode()` 操作来清除或修改它。比如：

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

这里的标准化操作将原来的文本分解为单独的和音符。接下来的ASCII编码/解码只是简单的一下子丢弃掉那些字符。当然，这种方法仅仅只在最后的目标就是获取到文本对应ACSII表示的时候生效。

讨论¶

文本字符清理一个最主要的问题应该是运行的性能。一般来讲，代码越简单运行越快。 对于简单的替换操作， `str.replace()` 方法通常是最快的，甚至在你需要多次调用的时候。 比如，为了清理空白字符，你可以这样做：

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

如果你去测试的话，你就会发现这种方式会比使用 `translate()` 或者正则表达式要快很多。

另一方面，如果你需要执行任何复杂字符对字符的重新映射或者删除操作的话， `translate()` 方法会非常的快。

从大的方面来讲，对于你的应用程序来说性能是你不得不去自己研究的东西。 不幸的是，我们不可能给你建议一个特定的技术，使它能够适应所有的情况。 因此实际情况中需要你自己去尝试不同的方法并评估它。

尽管这一节集中讨论的是文本，但是类似的技术也可以适用于字节，包括简单的替换，转换和正则表达式。