2.5 字符串搜索和替换¶

问题¶

你想在字符串中搜索和匹配指定的文本模式

解决方案¶

对于简单的字面模式，直接使用 `str.replace()` 方法即可，比如：

```
>>> text = 'yeah, but no, but yeah, but no, but yeah'
>>> text.replace('yeah', 'yep')
'yep, but no, but yep, but no, but yep'
>>>
```

对于复杂的模式，请使用 `re` 模块中的 `sub()` 函数。 为了说明这个，假设你想将形式为 `11/27/2012` 的日期字符串改成 `2012-11-27` 。示例如下：

```
>>> text = 'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> import re
>>> re.sub(r'(\d+)/(\d+)/(\d+)', r'\3-\1-\2', text)
'Today is 2012-11-27. PyCon starts 2013-3-13.'
>>>
```

`sub()` 函数中的第一个参数是被匹配的模式，第二个参数是替换模式。反斜杠数字比如 `\3` 指向前面模式的捕获组号。

如果你打算用相同的模式做多次替换，考虑先编译它来提升性能。比如：

```
>>> import re
>>> datepat = re.compile(r'(\d+)/(\d+)/(\d+)')
>>> datepat.sub(r'\3-\1-\2', text)
'Today is 2012-11-27. PyCon starts 2013-3-13.'
>>>
```

如果你使用了命名分组，那么第二个参数请使用 `\g<group_name>` ,如下

```
>>> text = 'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> import re
>>> re.sub(r'(?P<month>\d+)/(?P<day>\d+)/(?P<year>\d+)', r'\g<year>-\g<month>-\g<day>', text)
'Today is 2012-11-27. PyCon starts 2013-3-13.'
>>>
```

对于更加复杂的替换，可以传递一个替换回调函数来代替，比如：

```
>>> from calendar import month_abbr
>>> def change_date(m):
... mon_name = month_abbr[int(m.group(1))]
... return '{} {} {}'.format(m.group(2), mon_name, m.group(3))
...
>>> datepat.sub(change_date, text)
'Today is 27 Nov 2012. PyCon starts 13 Mar 2013.'
>>>
```

一个替换回调函数的参数是一个 `match` 对象，也就是 `match()` 或者 `find()` 返回的对象。 使用 `group()` 方法来提取特定的匹配部分。回调函数最后返回替换字符串。

如果除了替换后的结果外，你还想知道有多少替换发生了，可以使用 `re.subn()` 来代替。比如：

```
>>> newtext, n = datepat.subn(r'\3-\1-\2', text)
>>> newtext
'Today is 2012-11-27. PyCon starts 2013-3-13.'
>>> n
2
>>>
```

讨论¶

关于正则表达式搜索和替换，上面演示的 `sub()` 方法基本已经涵盖了所有。 其实最难的部分就是编写正则表达式模式，这个最好是留给读者自己去练习了。