2.4 字符串匹配和搜索¶

问题¶

你想匹配或者搜索特定模式的文本

解决方案¶

如果你想匹配的是字面字符串，那么你通常只需要调用基本字符串方法就行， 比如 `str.find()`,`str.endswith()`, `str.startswith()` 或者类似的方法：

```
>>> text = 'yeah, but no, but yeah, but no, but yeah'
>>> # Exact match
>>> text == 'yeah'
False
>>> # Match at start or end
>>> text.startswith('yeah')
True
>>> text.endswith('no')
False
>>> # Search for the location of the first occurrence
>>> text.find('no')
10
>>>
```

对于复杂的匹配需要使用正则表达式和 `re` 模块。 为了解释正则表达式的基本原理，假设你想匹配数字格式的日期字符串比如 `11/27/2012`，你可以这样做：

```
>>> text1 = '11/27/2012'
>>> text2 = 'Nov 27, 2012'
>>>
>>> import re
>>> # Simple matching: \d+ means match one or more digits
>>> if re.match(r'\d+/\d+/\d+', text1):
... print('yes')
... else:
... print('no')
...
yes
>>> if re.match(r'\d+/\d+/\d+', text2):
... print('yes')
... else:
... print('no')
...
no
>>>
```

如果你想使用同一个模式去做多次匹配，你应该先将模式字符串预编译为模式对象。比如：

```
>>> datepat = re.compile(r'\d+/\d+/\d+')
>>> if datepat.match(text1):
... print('yes')
... else:
... print('no')
...
yes
>>> if datepat.match(text2):
... print('yes')
... else:
... print('no')
...
no
>>>
```

`match()` 总是从字符串开始去匹配，如果你想查找字符串任意部分的模式出现位置， 使用 `findall()` 方法去代替。比

如：

```
>>> text = 'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> datepat.findall(text)
['11/27/2012', '3/13/2013']
>>>
```

在定义正则式的时候，通常会利用括号去捕获分组。比如：

```
>>> datepat = re.compile(r'(\d+)/(\d+)/(\d+)')
>>>
```

捕获分组可以使得后面的处理更加简单，因为可以分别将每个组的内容提取出来。比如：

```
>>> m = datepat.match('11/27/2012')
>>> m
<_sre.SRE_Match object at 0x1005d2750>
>>> # Extract the contents of each group
>>> m.group(0)
'11/27/2012'
>>> m.group(1)
'11'
>>> m.group(2)
'27'
>>> m.group(3)
'2012'
>>> m.groups()
('11', '27', '2012')
>>> month, day, year = m.groups()
>>>
>>> # Find all matches (notice splitting into tuples)
>>> text
'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> datepat.findall(text)
[('11', '27', '2012'), ('3', '13', '2013')]
>>> for month, day, year in datepat.findall(text):
...     print('{}-{}-{}'.format(year, month, day))
...
2012-11-27
2013-3-13
>>>
```

`findall()` 方法会搜索文本并以列表形式返回所有的匹配。 如果你想以迭代方式返回匹配，可以使用 `finditer()` 方法来代替，比如：

```
>>> for m in datepat.finditer(text):
...     print(m.groups())
...
('11', '27', '2012')
('3', '13', '2013')
>>>
```

讨论¶

关于正则表达式理论的教程已经超出了本书的范围。 不过，这一节阐述了使用re模块进行匹配和搜索文本的最基本方法。 核心步骤就是先使用 `re.compile()` 编译正则表达式字符串， 然后使用 `match()` , `findall()` 或者 `finditer()` 等方法。

当写正则式字符串的时候，相对普遍的做法是使用原始字符串比如 `r'(\d+)/(\d+)/(\d+)'` 。 这种字符串将不去解析反斜杠，这在正则表达式中是很有用的。 如果不这样做的话，你必须使用两个反斜杠，类似 `'(\\d+)/(\\d+)/(\\d+)'` 。

需要注意的是 `match()` 方法仅仅检查字符串的开始部分。它的匹配结果有可能并不是你期望的那样。比如：

```
>>> m = datepat.match('11/27/2012abcdef')
>>> m
<_sre.SRE_Match object at 0x1005d27e8>
```

```
>>> m.group()
'11/27/2012'
>>>
```

如果你想精确匹配，确保你的正则表达式以$结尾，就像这么这样：

```
>>> datepat = re.compile(r'(\d+)/(\d+)/(\d+)$')
>>> datepat.match('11/27/2012abcdef')
>>> datepat.match('11/27/2012')
<_sre.SRE_Match object at 0x1005d2750>
>>>
```

最后，如果你仅仅是做一次简单的文本匹配/搜索操作的话，可以略过编译部分，直接使用 `re` 模块级别的函数。比如：

```
>>> re.findall(r'(\d+)/(\d+)/(\d+)', text)
[('11', '27', '2012'), ('3', '13', '2013')]
>>>
```

但是需要注意的是，如果你打算做大量的匹配和搜索操作的话，最好先编译正则表达式，然后再重复使用它。 模块级别的函数会将最近编译过的模式缓存起来，因此并不会消耗太多的性能， 但是如果使用预编译模式的话，你将会减少查找和一些额外的处理损耗。