

Python的regex模块——更强大的正则表达式引擎

Python自带了正则表达式引擎（内置的re模块），但是不支持一些高级特性，比如下面这几个：

- 固化分组 Atomic grouping
- 占有优先量词 Possessive quantifiers
- 可变长度的逆序环视 Variable-length lookbehind
- 递归匹配 Recursive patterns
- （起始/继续）位置锚\G Search anchor

幸好，在2009年，Matthew Barnett写了一个更强大正则表达式引擎——[regex模块](#)，这是一个Python的第三方模块。

除了上面这几个高级特性，还有很多有趣、有用的东西，本文大致介绍一下，很多内容取自regex的文档。

无论是编程还是文本处理，regex模块都是一件利器。

用一个指标可以大致了解它的复杂性，re模块有4270行C语言代码，而regex模块有24513行C语言代码。

这个模块以后可能被收进Python标准库。目前（2015年）它还在不断发展，经常发布bug修正版，不过感觉用在一般生产环境应该没什么问题。

目录

- 一、安装regex
- 二、一些有趣的特性
- 三、模糊匹配
- 四、两种工作模式
- 五、Version 0模式和re模块不兼容之处

一、安装regex

regex支持Python 2.5+和Python 3.1+，可以用pip命令安装：

```
pip install regex
```

PyPy 2.6+也可以使用这个模块。

regex基本兼容re模块，现有的程序可以很容易切换到regex模块：

```
import regex as re
```

二、一些有趣的特性

完整的Unicode支持

1, 支持最新的Unicode标准, 这一点经常比Python本身还及时。

2, 支持Unicode代码属性, 包括scripts和blocks。

如: `\p{Cyrillic}`表示西里尔字符 (scripts), `\p{InCyrillic}`表示西里尔区块 (blocks)。

3, 支持完整的Unicode字符大小写匹配, 详见[此文](#)。

如: `ss`可匹配`ß`; `cliff` (这里的ff是一个字符) 可匹配`CLIFF` (FF是两个字符)。

不需要的话可以关闭此特性。不支持Unicode组合字符与单一字符的大小写匹配, 所以感觉这个特性不太实用。

4, `regex.WORD`标志开启后:

作用1: `\b`、`\B`采用Unicode的分界规则, 详见[此文](#)。

如: 开启后`\b.+?\b`可搜索到`3.4`; 关闭后小数点`.`成为分界符, 于是只能搜到`['3', '.', '4']`。

作用2: 采用Unicode的换行符。除了传统的`\r`、`\n`, Unicode还有一些换行符, 开启后作用于`.MULTILINE`和`.DOTALL`模式。

5, `\X`匹配Unicode的单个字形 (grapheme)。

Unicode有时用多个字符组合在一起表示一个字形, 详见[此文](#)。

`\X`匹配一个字形, 如: `^\X$`可以匹配`'\u0041\u0308'`。

单词起始位置、单词结束位置

`\b`是单词分界位置, 但不能区分是起始还是结束位置。

`regex`用`\m`表示单词起始位置, 用`\M`表示单词结束位置。

(?|...|...)

重置分支匹配中的捕获组编号。

```
>>> regex.match(r"(?|(first)|(second))", "first").groups()
('first',)
>>> regex.match(r"(?|(first)|(second))", "second").groups()
('second',)
```

两次匹配都是把捕获到的内容放到编号为1捕获组中, 在某些情况很方便。

(?flags-flags:...)

局部范围的flag控制。在`re`模块, flag只能作用于整个表达式, 现在可以作用于局部范围了:

```
>>> regex.search(r"<B>(?:i:good)</B>", "<B>GOOD</B>")
<regex.Match object; span=(0, 11), match='<B>GOOD</B>'>
```

在这个例子里, 忽略大小写模式只作用于标签之间的单词。

`(?:i:)`是打开忽略大小写, `(?:-i:)`则是关闭忽略大小写。

如果有多个flag挨着写既可，如(?is-f:)，减号左边的是打开，减号右边的是关闭。

除了局部范围的flag，还有全局范围的flag控制，如 (?si-f)good

re模块也支持这个，可以参见Python文档。

把flags写进表达式、而不是以函数参数的方式声明，方便直观且不易出错。

(?(DEFINE)...)

定义可重复使用的子句

```
>>> regex.search(r'(? (DEFINE) (?P<quant>\d+)(?P<item>\w+))(?&quant) (?&item)', '5 elephants')
<regex.Match object; span=(0, 11), match='5 elephants'>
```

此例中，定义之后，(?&quant)表示\d+，(?&item)表示\w+。如果子句很复杂，能省不少事。

partial matches

部分匹配。可用于验证用户输入，当输入不合法字符时，立刻给出提示。

可以pickle编译后的正则表达式对象

如果正则表达式很复杂或数量很多，编译需要较长时间。

这时可以把编译好的正则式用pickle存储到文件里，下次使用直接pickle.load()就不必再次编译了。

除了以上这些，还有很多新特性（匹配控制、便利方法等等），这里就不介绍了，请自行查阅文档。

三、模糊匹配

regex有模糊匹配（fuzzy matching）功能，能针对字符进行模糊匹配，提供了3种模糊匹配：

- i，模糊插入
- d，模糊删除
- s，模糊替换

以及e，包括以上三种模糊

举个例子：

```
>>> regex.findall('(?:hello){s<=2}', 'hallo')
['hallo']
```

(?:hello){s<=2}的意思是：匹配hello，其中最多容许有两个字符的错误。

于是可以成功匹配hallo。

这里只简单介绍一下模糊匹配，详情还是参见文档吧。

四、两种工作模式

regex有**Version 0**和**Version 1**两个工作模式，其中的**Version 0**基本兼容现有的re模块，以下是区别：

	Version 0 （基本兼容re模块）	Version 1
启用方法	设置.VERSION0或.V0标志，或者在表达式里写上(?V0)。	设置.VERSION1或.V1标志，或者在表达式里写上(?V1)。
零宽匹配	像re模块那样处理： .split 不能在零宽匹配处切割字符串。 .sub 在匹配零宽之后向前传动一个位置。	像Perl和PCRE那样处理： .split 可以在零宽匹配处切割字符串。 .sub 采用正确的行为。
内联flag	内联flag只能作用于整个表达式，不可关闭。	内联flag可以作用于局部表达式，可以关闭。
字符组	只支持简单的字符组。	字符组里可以有嵌套的集合，也可以做集合运算（并集、交集、差集、对称差集）。
大小写匹配	默认支持普通的Unicode字符大小写，如Й可匹配й。 这与Python3里re模块的默认行为相同。	默认支持完整的Unicode字符大小写，如ss可匹配ß。 可惜不支持Unicode组合字符与单一字符的大小写匹配，所以感觉这个特性不太实用。可以在表达式里写上(?-f)关闭此特性。


如果什么设置都不做，会采用regex.DEFAULT_VERSION指定的模式。在目前， regex.DEFAULT_VERSION的默认值是regex.V0。

如果想默认使用V1模式，这样就可以了：

```
import regex
regex.DEFAULT_VERSION = regex.V1
```

V1模式默认开启.FULLCASE（完整的忽略大小写匹配）。通常用不上这个，所以在忽略大小写匹配时用(?-f)关闭.FULLCASE即可，这样速度更快一点，例如： (?i-f)tag

其中零宽匹配的替换操作差异比如明显。绝大多数正则引擎采用的是Perl流派的作法，于是Version 1也朝着Perl的方向改过去了。



```
>>> # Version 0 behaviour (like re)
>>> regex.sub('(?V0).*', 'x', 'test')
'x'
```

```
>>> regex.sub('(?V0).*?', '|', 'test')
'|t|e|s|t|'

>>> # Version 1 behaviour (like Perl)
>>> regex.sub('(?V1).*', 'x', 'test')
'xx'
>>> regex.sub('(?V1).*?', '|', 'test')
'|'||'||'||'|'
```

re模块对零宽匹配的实现可能是有误的（见[issue1647489](#)）；

而V0零宽匹配的搜索和替换会出现不一致的行为（搜索采用V1的方式，替换采用re模块的方式）；

在Python 3.7+环境下，re和regex模块的行为同时做了改变，据称是采用了“正确”的方式处理零宽匹配：**总是返回第一个匹配（字符串或零宽），但是如果是零宽并且之前匹配的还是零宽则忽略。**这和3.6-的re模块、regex的V0模式、V1模式都略有不同。

说着挺吓人的，在实际使用中3.6- re、3.7+ re、V0、V1之间极少出现不兼容的现象。

五、Version 0模式和re模块不兼容之处

上面说了“Version 0基本兼容re模块”，说说不兼容的地方：

1、对零宽匹配的处理。

regex修复了re模块的搜索bug（见[issue1647489](#)），但是也带来了不兼容的问题。

在re中，用".*?"搜索"test"返回：['', '', '', '', '']，也就是：最前、字母之间的3个位置、最后，总共5个位置。

在regex中，则返回：['', 't', '', 'e', '', 's', '', 't', '']

在实际使用中，这个问题几乎不会造成不兼容的情况，所以基本可以忽略此差异。

2、\s的范围。

在re中，\s在**这一带**的范围是0x09 ~ 0x0D，0x1C ~ 0x1E。

在regex中，\s采用的是Unicode 6.3+标准的\p{Whitespace}，在**这一带**的范围有所缩小，只有：0x09 ~ 0x0D。

十六进制	十进制	英文说明	中文说明
0x09	9	HT (horizontal tab)	水平制表符
0x0A	10	LF (NL line feed, new line)	换行键
0x0B	11	VT (vertical tab)	垂直制表符
0x0C	12	FF (NP form feed, new page)	换页键
0x0D	13	CR (carriage return)	

			回车键
...
0x1C	28	FS (file separator)	文件分割符
0x1D	29	GS (group separator)	分组符
0x1E	30	RS (record separator)	记录分离符

除此之外，可能还有未知的不兼容之处。