**第25章 Python正则表达式**

**第一、二部分**

正则表达式是一个特殊的字符序列，它能帮助你方便的检查一个字符串是否与某种模式匹配。Python 自1.5版本起增加了re 模块，它提供 Perl 风格的正则表达式模式。

re 模块使 Python 语言拥有全部的正则表达式功能。

Python通过re模块提供对正则表达式的支持。使用re的一般步骤是先将正则表达式的字符串形式编译为Pattern实例，然后使用Pattern实例处理文本并获得匹配结果（一个Match实例），最后使用Match实例获得信息，进行其他的操作。

使用re模块进行正则运算的步骤：

*#1、引入re模块***import** re  
*#2、正则表达式的字符串*str=**"a.b"***#3、编译Pattern实例*pattern=re.compile(str,flags=re.I)  
*#4、获取得匹配结果*matchs=re.match(pattern,**"acb"**)  
*#5、其他操作*print(matchs.group())

**一、正则表达式中常用的字符含义**

1、普通字符和11个元字符：

|  |  |  |  |
| --- | --- | --- | --- |
| 普通字符 | 匹配自身 | abc | abc |
| . | 匹配任意除换行符"\n"外的字符(在DOTALL模式中也能匹配换行符 | a.c | abc |
| \ | 转义字符，使后一个字符改变原来的意思 | a\.c;a\\c | a.c;a\c |
| \* | 匹配前一个字符0或多次 | abc\* | ab;abccc |
| + | 匹配前一个字符1次或无限次 | abc+ | abc;abccc |
| ? | 匹配一个字符0次或1次 | abc? | ab;abc |
| ^ | 匹配字符串开头。在多行模式中匹配每一行的开头 | ^abc | abc |
| $ | 匹配字符串末尾，在多行模式中匹配每一行的末尾 | abc$ | abc |
| | | 或。匹配|左右表达式任意一个，从左到右匹配，如果|没有包括在()中，则它的范围是整个正则表达式 | abc|def | abc  def |
| {} | {m}匹配前一个字符m次，{m,n}匹配前一个字符m至n次，若省略n，则匹配m至无限次 | ab{1,2}c | abc  abbc |
| [] | 字符集。对应的位置可以是字符集中任意字符。字符集中的字符可以逐个列出，也可以给出范围，如[abc]或[a-c]。[^abc]表示取反，即非abc。 所有特殊字符在字符集中都失去其原有的特殊含义。用\反斜杠转义恢复特殊字符的特殊含义。 | a[bcd]e | abe  ace  ade |
| () | 被括起来的表达式将作为分组，从表达式左边开始没遇到一个  分组的左括号“（”，编号+1. 分组表达式作为一个整体，可以后接数量词。表达式中的|仅在该组中有效。 | (abc){2} a(123|456)c | abcabc  a456c |

*#encoding=utf-8***import** re  
rel=**r'a\\c'**p=re.compile(rel)  
**print** re.search(rel,**"a\c"**).group(0)

a\c

**第三部分**

**二、预定义字符集（可以写在字符集[...]中）**

|  |  |  |  |
| --- | --- | --- | --- |
| \d | 数字:[0-9] | a\dc | a1c |
| \D | 非数字:[^\d] | a\Dc | abc |
| \s | 匹配任何空白字符:[<空格>\t\r\n\f\v] | a\sc | a c |
| \S | 非空白字符:[^\s] | a\Sc | abc |
| \w | 匹配包括下划线在内的任何字字符:[A-Za-z0-9\_] | a\wc | abc |
| \W | 匹配非字母字符，即匹配特殊字符 | a\Wc | a c |
| \A | 仅匹配字符串开头,同^ | \Aabc | abc |
| \Z | 仅匹配字符串结尾，同$ | abc\Z | abc |
| \b | \b 称为单词边界（word boundary）符.  例如只想匹配 My cat is bad.中的cat 可以使用 \bcat\b  。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。 | \babc\b | 空格abc空格 |
| \B | [^\b],B 匹配非单词边界。“er\B”能匹配“verb”中的“er”，但不能匹配“never”中的“er”。 | er\B | verb |

*#encoding=utf-8***import** re  
rel=**r'\babc\b'**p=re.compile(rel)  
**print** re.search(p,**" abc "**).group(0)

abc

**第四部分**

**三、python中常见匹配函数**

***1、re.match() 函数***

re.match 尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话，match()就返回none。从首字母开始开始匹配，string如果包含pattern子串，则匹配成功，返回Match对象，失败则返回None。

函数语法：

re.match(pattern, string, flags=0)

函数参数说明：

|  |  |
| --- | --- |
| 参数 | 描述 |
| pattern | 匹配的正则表达式 |
| string | 要匹配的字符串。 |
| flags | 标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。 |

|  |  |
| --- | --- |
| 修饰符 | 描述 |
| re.I | 使匹配对大小写不敏感 |
| re.L | 做本地化识别（locale-aware）匹配 |
| re.M | 多行匹配，影响 ^ 和 $ |
| re.S | 使 . 匹配包括换行在内的所有字符 |
| re.U | 根据Unicode字符集解析字符。这个标志影响 \w, \W, \b, \B. |
| re.X | 该标志通过给予你更灵活的格式以便你将正则表达式写得更易于理解。 |

match和search一旦匹配成功，就是一个match object对象，而match object对象有以下方法：

 group() 返回被 RE 匹配的字符串,返回re整体匹配的字符串，可以一次输入多个组号，对应组号匹配的字符串

 start() 返回匹配开始的位置

 end() 返回匹配结束的位置

 span() 返回一个元组包含匹配 (开始,结束) 的位置

实例1：

import re

print(re.match('rupeng', 'rupeng.com').span())

结果：

(0, 6)

实例2：

import re

#re.I：使匹配对大小写不敏感

print(re.match('Rupeng', 'rupeng.com',re.I).span())

执行结果：

(0, 6)

#多行匹配

*#encoding=utf-8***import** re  
n=**'''12 drummers drumming,  
11 pipers piping, 10 lords a-leaping'''**p=re.compile(**'^\d+'**)  
p\_multi=re.compile(**'^\d+'**,re.M) *#设置 MULTILINE 标志*print(re.findall(p,n)) *#['12']*print(re.findall(p\_multi,n)) *# ['12', '11']*

我们可以使用group(num) 或 groups() 匹配对象函数来获取匹配表达式。

|  |  |
| --- | --- |
| 匹配对象方法 | 描述 |
| group(num=0) | 匹配的整个表达式的字符串，group() 可以一次输入多个组号，在这种情况下它将返回一个包含那些组所对应值的元组。 |
| groups() | 返回一个包含所有小组字符串的元组，从 1 到 所含的小组号。 |

group和groups是两个不同的函数。

一般，m.group(N) 返回第N组括号匹配的字符。

而m.group() == m.group(0) == 所有匹配的字符，与括号无关，这个是API规定的。

m.groups() 返回所有括号匹配的字符，以tuple格式。

m.groups() == (m.group(1), m.group(2), ...)

*#encoding="utf-8"***import** re  
**print** re.match(**"(abc0)(abc1)(abc2)"**, **"abc0abc1abc2"**).group()  
**print** re.match(**"(abc0)(abc1)(abc2)"**, **"abc0abc1abc2"**).group(0)  
**print** re.match(**"(abc0)(abc1)(abc2)"**, **"abc0abc1abc2"**).group(1)  
**print** re.match(**"(abc0)(abc1)(abc2)"**, **"abc0abc1abc2"**).group(2)  
**print** re.match(**"(abc0)(abc1)(abc2)"**, **"abc0abc1abc2"**).group(3)  
**print** re.match(**"(abc0)(abc1)(abc2)"**, **"abc0abc1abc2"**).groups()  
**for** g **in** re.match(**"(abc0)(abc1)(abc2)"**, **"abc0abc1abc2"**).groups():  
 **print** g

执行结果：

abc0abc1abc2

abc0abc1abc2

abc0

abc1

abc2

('abc0', 'abc1', 'abc2')

abc0

abc1

abc2

实例3：

import re

patter=re.compile("Rupeng")

m=re.match(patter, 'Rupeng.com')

print(m.group(0))

执行结果：

Rupeng

**第五部分**

2、re.search

语法：

re.search(pattern, string[, flags])

若string中包含pattern子串，则返回Match对象，否则返回None，注意，如果string中存在多个pattern子串，只返回第一个。

import re

patter=re.compile("Rupeng")

m=re.search(patter, 'we2323Rupeng.com')

print(m.span())

print(m.group(0))

执行结果：

(6, 12)

Rupeng

re.match只匹配字符串的开始，如果字符串开始不符合正则表达式，则匹配失败，函数返回None；而re.search匹配整个字符串，直到找到一个匹配。

3、re.findall

语法：

re.findall(pattern, string[, flags])

返回string中所有与pattern相匹配的全部字串，返回形式为列表。

import re

patter=re.compile("Rupeng")

m=re.findall(patter, 'Rupengwe2323Rupeng.comRupeng')

for l in m:

print l

执行结果：

Rupeng

Rupeng

Rupeng

4、re.finditer

re.finditer(pattern, string[, flags])

返回string中所有与pattern相匹配的全部字串，返回形式为迭代器。

若匹配成功，match()/search()返回的是Match对象，finditer()返回的也是Match对象的迭代器，获取匹配结果需要调用Match对象的group()、groups或group(index)方法。

import re

patter=re.compile("Rupeng")

m=re.finditer(patter, 'Rupengwe2323Rupeng.comRupeng')

for l in m:

print l.span()

print l.group(0)

执行结果：

(0, 6)

Rupeng

(12, 18)

Rupeng

(22, 28)

Rupeng

import re

s = '2342werwrew23432werwre'

p = r'(\d\*)([a-zA-Z]\*)(\d\*)([a-zA-Z]\*)'

m = re.search(p,s)

print(m.group())

print(m.group(0))

print(m.group(1))

print(m.group(2))

print(m.group(3))

print(m.group(4))

print(m.groups())

执行结果：

2342werwrew23432werwre

2342werwrew23432werwre

2342

werwrew

23432

werwre

('2342', 'werwrew', '23432', 'werwre')

**第六部分**

5、re.sub

Python 的re模块提供了re.sub用于替换字符串中的匹配项。

语法：

re.sub(pattern, repl, string, count=0)

参数：

 pattern : 正则中的模式字符串。

 repl : 替换的字符串，也可为一个函数。

 string : 要被查找替换的原始字符串。

 count : 模式匹配后替换的最大次数，默认 0 表示替换所有的匹配。

#coding=utf-8

import re

phone = "2004-959-559 # 这是一个电话号码"

# 删除注释

num = re.sub(r'#.\*$', "", phone)

print "电话号码 : ", num

# 移除非数字的内容

num = re.sub(r'\D', "", phone)

print "电话号码 : ", num

执行结果：

电话号码 : 2004-959-559

电话号码 : 2004959559

repl 参数是一个函数

以下实例中将字符串中的匹配的数字乘于 2：

import re

# 将匹配的数字乘以2

def double(matched):

value = int(matched.group('a'))

return str(value \* 2)

s = 'A38G4HFD56q7'

print(re.sub('(?P<a>\d+)', double, s))

执行输出结果为：

A76G8HFD112q14

？P表示组的名字，<组名>。可以通过matched.group(组名')获取改组，或者通过下标值获取。

The syntax for a named group is one of the Python-specific extensions: (?P<name>...). *name* is, obviously, the name of the group. Named groups also behave exactly like capturing groups, and additionally associate a name with a group. The[match object](http://blog.csdn.net/huxian370/article/details/51344311) methods that deal with capturing groups all accept either integers that refer to the group by number or strings that contain the desired group’s name. Named groups are still given numbers, so you can retrieve information about a group in two ways:

1. >>> p = re.compile(r'(?P<word>\b\w+\b)')

2. >>> m = p.search( '(((( Lots of punctuation )))' )

3. >>> m.group('word')

4. 'Lots'

5. >>> m.group(1)

6. 'Lots'

补充知识点：正则表达式compile的几种模式

在正则表达式中

Pattern Pattern.compile(String regex, int flag)

flag的取值范围如下： 编译标志效果

Pattern.CANON\_EQ 当且仅当两个字符的"正规分解(canonical decomposition)"都完全相同的情况下，才认定匹配。比如用了这个标志之后，表达式"a/u030A"会匹配"?"。默认情况下，不考虑"规范相等性(canonical equivalence)"。

Pattern.CASE\_INSENSITIVE

(?i) 默认情况下，大小写不明感的匹配只适用于US-ASCII字符集。这个标志能让表达式忽略大小写进行匹配。要想对Unicode字符进行大小不明感的匹配，只要将UNICODE\_CASE与这个标志合起来就行了。

Pattern.COMMENTS

(?x) 在这种模式下，匹配时会忽略(正则表达式里的)空格字符(注：不是指表达式里的"//s"，而是指表达式里的空格，tab，回车之类)。注释从#开始，一直到这行结束。可以通过嵌入式的标志来启用Unix行模式。

Pattern.DOTALL

(?s) 在这种模式下，表达式'.'可以匹配任意字符，包括表示一行的结束符。默认情况下，表达式'.'不匹配行的结束符。

Pattern.MULTILINE

(?m) 在这种模式下，'^'和'$'分别匹配一行的开始和结束。此外，'^'仍然匹配字符串的开始，'$'也匹配字符串的结束。默认情况下，这两个表达式仅仅匹配字符串的开始和结束。

Pattern.UNICODE\_CASE

(?u) 在这个模式下，如果你还启用了CASE\_INSENSITIVE标志，那么它会对Unicode字符进行大小写不明感的匹配。默认情况下，大小写不明感的匹配只适用于US-ASCII字符集。

Pattern.UNIX\_LINES

(?d) 在这个模式下，只有'/n'才被认作一行的中止，并且与'.'，'^'，以及'$'进行匹配。

在这些标志里面，Pattern.CASE\_INSENSITIVE，Pattern.MULTILINE，以及Pattern.COMMENTS是最有用的(其中Pattern.COMMENTS还能帮我们把思路理清楚，并且/或者做文档)。注意，你可以用在表达式里插记号的方式来启用绝大多数的模式。这些记号就在上面那张表的各个标志的下面。你希望模式从哪里开始启动，就在哪里插记号。

可以用"OR" ('|')运算符把这些标志合使用