Python 面向对象 Python CGI编程

Python 正则表达式

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。

compile 函数根据一个模式字符串和可选的标志参数生成一个正则表达式对象。该对象拥有一系列方法用于正则表达式匹配和替换。

re 模块也提供了与这些方法功能完全一致的函数，这些函数使用一个模式字符串做为它们的第一个参数。

本章节主要介绍Python中常用的正则表达式处理函数。

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。

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。参见：正则表达式修饰符 - 可选标志

匹配成功re.match方法返回一个匹配的对象，否则返回None。

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。

实例

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import re
```

```
print(re.match('www', 'www.runoob.com').span()) # 在起始位置匹配
print(re.match('com', 'www.runoob.com')) # 不在起始位置匹配
```

以上实例运行输出结果为：

```
(0, 3)
None
```

实例

```
#!/usr/bin/python
import re
line = "Cats are smarter than dogs"
matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)
if matchObj:
print "matchObj.group() : ", matchObj.group()
print "matchObj.group(1) : ", matchObj.group(1)
print "matchObj.group(2) : ", matchObj.group(2)
else:
print "No match!!"
```

以上实例执行结果如下：

```
matchObj.group() :  Cats are smarter than dogs
matchObj.group(1) :  Cats
matchObj.group(2) :  smarter
```

re.search方法

re.search 扫描整个字符串并返回第一个成功的匹配。

函数语法：

```
re.search(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。

匹配成功re.search方法返回一个匹配的对象，否则返回None。

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。

实例

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import re
print(re.search('www', 'www.runoob.com').span()) # 在起始位置匹配
print(re.search('com', 'www.runoob.com').span()) # 不在起始位置匹配
```

以上实例运行输出结果为：

```
(0, 3)
(11, 14)
```

实例

```
#!/usr/bin/python
import re
line = "Cats are smarter than dogs";
searchObj = re.search( r'(.*) are (.*?) .*', line, re.M|re.I)
if searchObj:
print "searchObj.group() : ", searchObj.group()
print "searchObj.group(1) : ", searchObj.group(1)
print "searchObj.group(2) : ", searchObj.group(2)
else:
print "Nothing found!!"
```

以上实例执行结果如下：

```
searchObj.group() :  Cats are smarter than dogs
searchObj.group(1) :  Cats
searchObj.group(2) :  smarter
```

re.match与re.search的区别

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。

实例

```
#!/usr/bin/python
import re
line = "Cats are smarter than dogs";
matchObj = re.match( r'dogs', line, re.M|re.I)
if matchObj:
```

```
print "match --> matchObj.group() : ", matchObj.group()
else:
print "No match!!"
matchObj = re.search( r'dogs', line, re.M|re.I)
if matchObj:
print "search --> matchObj.group() : ", matchObj.group()
else:
print "No match!!"
```

以上实例运行结果如下：

```
No match!!
search --> matchObj.group() :  dogs
```

检索和替换

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, flags=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。

实例

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import re
phone = "2004-959-559 # 这是一个国外电话号码"
# 删除字符串中的 Python注释
num = re.sub(r'#.*$', "", phone)
print "电话号码是: ", num
# 删除非数字(-)的字符串
num = re.sub(r'\D', "", phone)
print "电话号码是 : ", num
```

以上实例执行结果如下：

```
电话号码是:  2004-959-559
电话号码是 :  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以 2：

实例

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import re
# 将匹配的数字乘以 2
def double(matched):
value = int(matched.group('value'))
return str(value * 2)
s = 'A23G4HFD567'
print(re.sub('(?P<value>\d+)', double, s))
```

执行输出结果为：

```
A46G8HFD1134
```

re.compile 函数

compile 函数用于编译正则表达式，生成一个正则表达式（ Pattern ）对象，供 match() 和 search() 这两个函数使用。

语法格式为：

```
re.compile(pattern[, flags])
```

参数：

- `pattern` : 一个字符串形式的正则表达式
- `flags` : 可选，表示匹配模式，比如忽略大小写，多行模式等，具体参数为：
    1. **re.I** 忽略大小写
    2. **re.L** 表示特殊字符集 \w, \W, \b, \B, \s, \S 依赖于当前环境
    3. **re.M** 多行模式
    4. **re.S** 即为 `.` 并且包括换行符在内的任意字符（ `.` 不包括换行符）
    5. **re.U** 表示特殊字符集 \w, \W, \b, \B, \d, \D, \s, \S 依赖于 Unicode 字符属性数据库
    6. **re.X** 为了增加可读性，忽略空格和 `#` 后面的注释

实例

实例

```
>>>import re
>>> pattern = re.compile(r'\d+') # 用于匹配至少一个数字
>>> m = pattern.match('one12twothree34four') # 查找头部，没有匹配
>>> print m
None
```

```
>>> m = pattern.match('one12twothree34four', 2, 10) # 从'e'的位置开始匹配，没有匹配
>>> print m
None
>>> m = pattern.match('one12twothree34four', 3, 10) # 从'1'的位置开始匹配，正好匹配
>>> print m # 返回一个 Match 对象
<_sre.SRE_Match object at 0x10a42aac0>
>>> m.group(0) # 可省略 0
'12'
>>> m.start(0) # 可省略 0
3
>>> m.end(0) # 可省略 0
5
>>> m.span(0) # 可省略 0
(3, 5)
```

在上面，当匹配成功时返回一个 Match 对象，其中：

- `group([group1, …])` 方法用于获得一个或多个分组匹配的字符串，当要获得整个匹配的子串时，可直接使用 `group()` 或 `group(0)`；
- `start([group])` 方法用于获取分组匹配的子串在整个字符串中的起始位置（子串第一个字符的索引），参数默认值为 0；
- `end([group])` 方法用于获取分组匹配的子串在整个字符串中的结束位置（子串最后一个字符的索引+1），参数默认值为 0；
- `span([group])` 方法返回 `(start(group), end(group))`。

再看看一个例子：

实例

```
>>>import re
>>> pattern = re.compile(r'([a-z]+) ([a-z]+)', re.I) # re.I 表示忽略大小写
>>> m = pattern.match('Hello World Wide Web')
>>> print m # 匹配成功，返回一个 Match 对象
<_sre.SRE_Match object at 0x10bea83e8>
>>> m.group(0) # 返回匹配成功的整个子串
'Hello World'
>>> m.span(0) # 返回匹配成功的整个子串的索引
(0, 11)
>>> m.group(1) # 返回第一个分组匹配成功的子串
'Hello'
>>> m.span(1) # 返回第一个分组匹配成功的子串的索引
(0, 5)
>>> m.group(2) # 返回第二个分组匹配成功的子串
'World'
>>> m.span(2) # 返回第二个分组匹配成功的子串
(6, 11)
>>> m.groups() # 等价于 (m.group(1), m.group(2), ...)
('Hello', 'World')
>>> m.group(3) # 不存在第三个分组
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: no such group
```

findall

在字符串中找到正则表达式所匹配的所有子串，并返回一个列表，如果没有找到匹配的，则返回空列表。

注意： match 和 search 是匹配一次 findall 匹配所有。

语法格式为：

```
findall(string[, pos[, endpos]])
```

参数：

- `string`：待匹配的字符串。
- `pos`：可选参数，指定字符串的起始位置，默认为 0。
- `endpos`：可选参数，指定字符串的结束位置，默认为字符串的长度。

查找字符串中的所有数字：

实例

```
# -*- coding:UTF8 -*-
import re
pattern = re.compile(r'\d+') # 查找数字
result1 = pattern.findall('runoob 123 google 456')
result2 = pattern.findall('run88oob123google456', 0, 10)
print(result1)
print(result2)
```

输出结果：

```
['123', '456']
['88', '12']
```

re.finditer

和 findall 类似，在字符串中找到正则表达式所匹配的所有子串，并把它们作为一个迭代器返回。

```
re.finditer(pattern, string, flags=0)
```

参数：

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。参见：正则表达式修饰符 - 可选标志

实例

```
# -*- coding: UTF-8 -*-
import re
it = re.finditer(r"\d+","12a32bc43jf3")
for match in it:
print (match.group() )
```

输出结果：

```
12
32
43
3
```

re.split

split 方法按照能够匹配的子串将字符串分割后返回列表，它的使用形式如下：

```
re.split(pattern, string[, maxsplit=0, flags=0])
```

参数：

参数	描述
pattern	匹配的正则表达式
string	要匹配的字符串。
maxsplit	分隔次数，maxsplit=1 分隔一次，默认为 0，不限制次数。
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。参见：正则表达式修饰符 - 可选标志

实例

```
>>>import re
>>> re.split('\W+', 'runoob, runoob, runoob.')
['runoob', 'runoob', 'runoob', '']
>>> re.split('(\W+)', ' runoob, runoob, runoob.')
['', ' ', 'runoob', ', ', 'runoob', ', ', 'runoob', '.', '']
>>> re.split('\W+', ' runoob, runoob, runoob.', 1)
['', 'runoob, runoob, runoob.']
>>> re.split('a*', 'hello world') # 对于一个找不到匹配的字符串而言，split 不会对其作出分割
['hello world']
```

正则表达式对象

re.RegexObject

re.compile() 返回 RegexObject 对象。

re.MatchObject

group() 返回被 RE 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

正则表达式修饰符 - 可选标志

正则表达式可以包含一些可选标志修饰符来控制匹配的模式。修饰符被指定为一个可选的标志。多个标志可以通过按位 OR(|) 它们来指定。如 re.I | re.M 被设置成 I 和 M 标志：

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。

正则表达式模式

模式字符串使用特殊的语法来表示一个正则表达式：

字母和数字表示他们自身。一个正则表达式模式中的字母和数字匹配同样的字符串。

多数字母和数字前加一个反斜杠时会拥有不同的含义。

标点符号只有被转义时才匹配自身，否则它们表示特殊的含义。

反斜杠本身需要使用反斜杠转义。

由于正则表达式通常都包含反斜杠，所以你最好使用原始字符串来表示它们。模式元素(如 r'\t'，等价于 '\\t')匹配相应的特殊字符。

下表列出了正则表达式模式语法中的特殊元素。如果你使用模式的同时提供了可选的标志参数，某些模式元素的含义会改变。

模式	描述
^	匹配字符串的开头
$	匹配字符串的末尾。
.	匹配任意字符，除了换行符，当re.DOTALL标记被指定时，则可以匹配包括换行符的任意字

	符。
[...]	用来表示一组字符,单独列出：[amk] 匹配 'a'，'m'或'k'
[^...]	不在[]中的字符：[^abc] 匹配除了a,b,c之外的字符。
re*	匹配0个或多个的表达式。
re+	匹配1个或多个的表达式。
re?	匹配0个或1个由前面的正则表达式定义的片段，非贪婪方式
re{ n}	精确匹配 n 个前面表达式。例如， `o{2}` 不能匹配 "Bob" 中的 "o"，但是能匹配 "food" 中的两个 o。
re{ n,}	匹配 n 个前面表达式。例如， o{2,} 不能匹配"Bob"中的"o"，但能匹配 "foooood"中的所有 o。"o{1,}" 等价于 "o+"。"o{0,}" 则等价于 "o*"。
re{ n, m}	匹配 n 到 m 次由前面的正则表达式定义的片段，贪婪方式
a| b	匹配a或b
(re)	匹配括号内的表达式，也表示一个组
(?imx)	正则表达式包含三种可选标志：i, m, 或 x 。只影响括号中的区域。
(?-imx)	正则表达式关闭 i, m, 或 x 可选标志。只影响括号中的区域。
(?: re)	类似 (...), 但是不表示一个组
(?imx: re)	在括号中使用i, m, 或 x 可选标志
(?-imx: re)	在括号中不使用i, m, 或 x 可选标志
(?#...)	注释.
(?= re)	前向肯定界定符。如果所含正则表达式，以 ... 表示，在当前位置成功匹配时成功，否则失败。但一旦所含表达式已经尝试，匹配引擎根本没有提高；模式的剩余部分还要尝试界定符的右边。
(?! re)	前向否定界定符。与肯定界定符相反；当所含表达式不能在字符串当前位置匹配时成功
(?> re)	匹配的独立模式，省去回溯。
\w	匹配字母数字及下划线

\W	匹配非字母数字及下划线
\s	匹配任意空白字符，等价于 [\t\n\r\f].
\S	匹配任意非空字符
\d	匹配任意数字，等价于 [0-9].
\D	匹配任意非数字
\A	匹配字符串开始
\Z	匹配字符串结束，如果是存在换行，只匹配到换行前的结束字符串。
\z	匹配字符串结束
\G	匹配最后匹配完成的位置。
\b	匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。
\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'。
\n, \t, 等.	匹配一个换行符。匹配一个制表符。等
\1...\9	匹配第n个分组的内容。
\10	匹配第n个分组的内容，如果它经匹配。否则指的是八进制字符码的表达式。

正则表达式实例

字符匹配

实例	描述
python	匹配 "python".

字符类

实例	描述
[Pp]ython	匹配 "Python" 或 "python"
rub[ye]	匹配 "ruby" 或 "rube"
[aeiou]	匹配中括号内的任意一个字母
[0-9]	匹配任何数字。类似于 [0123456789]
[a-z]	匹配任何小写字母

[A-Z]	匹配任何大写字母
[a-zA-Z0-9]	匹配任何字母及数字
[^aeiou]	除了aeiou字母以外的所有字符
[^0-9]	匹配除了数字外的字符

特殊字符类

实例	描述
.	匹配除 "\n" 之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用象 '[.\n]' 的模式。
\d	匹配一个数字字符。等价于 [0-9]。
\D	匹配一个非数字字符。等价于 [^0-9]。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]。
\S	匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。
\w	匹配包括下划线的任何单词字符。等价于'[A-Za-z0-9_]'。
\W	匹配任何非单词字符。等价于 '[^A-Za-z0-9_]'。

← Python 面向对象 Python CGI编程 →

2 篇笔记

写笔记

正则表达式实例：

```
#!/usr/bin/python
import re
line = "Cats are smarter than dogs"
matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)
if matchObj:
    print "matchObj.group() : ", matchObj.group()
    print "matchObj.group(1) : ", matchObj.group(1)
    print "matchObj.group(2) : ", matchObj.group(2)
else:
    print "No match!!"
```

正则表达式：

```
r'(.*) are (.*?) .*'
```

解析:

首先，这是一个字符串，前面的一个 `r` 表示字符串为非转义的原始字符串，让编译器忽略反斜杠，也就是忽略转义字符。但是这个字符串里没有反斜杠，所以这个 `r` 可有可无。

- **(.*)** 第一个匹配分组，`.*` 代表匹配除换行符之外的所有字符。
- **(.*?)** 第二个匹配分组，`.*?` 后面多个问号，代表非贪婪模式，也就是说只匹配符合条件的最少字符
- 后面的一个 `.*` 没有括号包围，所以不是分组，匹配效果和第一个一样，但是不计入匹配结果中。

matchObj.group() 等同于 matchObj.group(0)，表示匹配到的完整文本字符

matchObj.group(1) 得到第一组匹配结果，也就是(.*)匹配到的

matchObj.group(2) 得到第二组匹配结果，也就是(.*?)匹配到的

因为只有匹配结果中只有两组，所以如果填 3 时会报错。

jim 2年前 (2017-04-24)

`'(?P...)'` 分组匹配

例：身份证 **1102231990xxxxxxxx**

```
import re
s = '1102231990xxxxxxxx'
res = re.search('(?P<province>\d{3})(?P<city>\d{3})(?P<born_year>\d{4})',s)
print(res.groupdict())
```

此分组取出结果为：

```
{'province': '110', 'city': '223', 'born_year': '1990'}
```

直接将匹配结果直接转为字典模式，方便使用。

CrazyDemo 8个月前 (07-23)