Python正则表达式指南 - AstralWind - 博客园

星期六, 1月 7 2012, 6:27 上午

< 2010年7月 >

日	一	二	三	四	五	六
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

我的标签

Java(3)
JDK(1)
动态代理(1)
Calendar(1)
Hibernate(1)
映射(1)

随笔分类

Java(4) (rss)
Java web(1) (rss)
Other(1) (rss)
Python(13) (rss)
随笔(2) (rss)
学习笔记(3) (rss)

随笔档案

2011年7月 (3)
2011年6月 (2)
2011年3月 (2)
2011年1月 (1)
2010年12月 (2)
2010年7月 (1)
2010年6月 (2)
2009年12月 (6)

最新评论

1. Re:Python字符编码详解
楼主讲真不错
简单清晰
很多年都搞不清楚编码问题，这个清楚了，多谢
--YuxuanWang
2. Re:Python正则表达式指南
谢谢楼主分享
--pyt123
3. Re:Python函数式编程指南（四）：生成

Python正则表达式指南

本文介绍了Python对于正则表达式的支持，包括正则表达式基础以及Python正则表达式标准库的完整介绍及使用示例。本文的内容不包括如何编写高效的正则表达式、如何优化正则表达式，这些主题请查看其他教程。

注意：本文基于Python2.4完成；如果看到不明白的词汇请记得百度谷歌或维基，whatever。

1. 正则表达式基础

1.1. 简单介绍

正则表达式并不是Python的一部分。正则表达式是用于处理字符串的强大工具，拥有自己独特的语法以及一个独立的处理引擎，效率上可能不如str自带的方法，但功能十分强大。得益于这一点，在提供了正则表达式的语言里，正则表达式的语法都是一样的，区别只在于不同的编程语言实现支持的语法数量不同；但不用担心，不被支持的语法通常是不常用的部分。如果已经在其他语言里使用过正则表达式，只需要简单看一看就可以上手了。

下图展示了使用正则表达式进行匹配的流程：

正则表达式的大致匹配过程是：依次拿出表达式和文本中的字符比较，如果每一个字符都能匹配，则匹配成功；一旦有匹配不成功的字符则匹配失败。如果表达式中有量词或边界，这个过程会稍微有一些不同，但也是很好理解的，看下图中的示例以及自己多使用几次就能明白。

下图列出了Python支持的正则表达式元字符和语法：

器
临睡前，不小心看到了Python函数式编程4篇，一口气读完了，以前对Lambda都很是恐惧的我，在作者风趣、简明的语言引诱下读完了，也终于理解了函数编程的一些概念。

--有一个Python爱好者

4. Re:Python正则表达式指南
好哇，我每次写正则表达式时，都来这看。脑子不行，记不住规则，只能随写随查

--会长

5. Re:Python正则表达式指南
@你能记住几个
ms office

--AstralWind

阅读排行榜

1. Python正则表达式指南(21728)
2. Python线程指南(6039)
3. Python自省（反射）指南(4441)
4. Python装饰器与面向切面编程(3872)
5. Python字符编码详解(2763)

评论排行榜

1. Python正则表达式指南(19)
2. 继承自java.util.Calendar的200年农历(11)
3. Python装饰器与面向切面编程(10)
4. 使用自定义主题让Windows Live Writer在本地预览语法高亮效果(9)
5. Python字符编码详解(7)

1.2. 数量词的贪婪模式与非贪婪模式

正则表达式通常用于在文本中查找匹配的字符串。Python里数量词默认是贪婪的（在少数语言里也可能是默认非贪婪），总是尝试匹配尽可能多的字符；非贪婪的则相反，总是尝试匹配尽可能少的字符。例如：正则表达式"ab*"如果用于查找"abbbc"，将找到"abbb"。而如果使用非贪婪的数量词"ab*?"，将找到"a"。

1.3. 反斜杠的困扰

与大多数编程语言相同，正则表达式里使用"\"作为转义字符，这就可能造成反斜杠困扰。假如你需要匹配文本中的字符"\"，

那么使用编程语言表示的正则表达式里将需要4个反斜杠"\\\\"：前两个和后两个分别用于在编程语言里转义成反斜杠，转换成两个反斜杠后再在正则表达式里转义成一个反斜杠。Python里的原生字符串很好地解决了这个问题，这个例子中的正则表达式可以使用r"\\"表示。同样，匹配一个数字的"\\d"可以写成r"\d"。有了原生字符串，你再也不用担心是不是漏写了反斜杠，写出来的表达式也更直观。

1.4. 匹配模式

正则表达式提供了一些可用的匹配模式，比如忽略大小写、多行匹配等，这部分内容将在Pattern类的工厂方法re.compile(pattern[, flags])中一起介绍。

2. re模块

2.1. 开始使用re

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。

```
# encoding: UTF-8
import re

# 将正则表达式编译成Pattern对象
pattern = re.compile(r'hello')

# 使用Pattern匹配文本，获得匹配结果，无法匹配时将返回None
match = pattern.match('hello world!')

if match:
    # 使用Match获得分组信息
    print match.group()

### 输出 ###
# hello
```

re.compile(strPattern[, flag]):

这个方法是Pattern类的工厂方法，用于将字符串形式的正则表达式编译为Pattern对象。 第二个参数flag是匹配模式，取值可以使用按位或运算符'|'表示同时生效，比如re.I | re.M。另外，你也可以在regex字符串中指定模式，比如re.compile('pattern', re.I | re.M)与re.compile('(?im)pattern')是等价的。
可选值有：

- re.**I**(re.IGNORECASE): 忽略大小写（括号内是完整写法，下同）
- **M**(MULTILINE): 多行模式，改变'^'和'$'的行为（参见上图）
- **S**(DOTALL): 点任意匹配模式，改变'.'的行为
- **L**(LOCALE): 使预定字符类 \w \W \b \B \s \S 取决于当前区域设定
- **U**(UNICODE): 使预定字符类 \w \W \b \B \s \S \d \D 取决于unicode定义的字符属性
- **X**(VERBOSE): 详细模式。这个模式下正则表达式可以是多行，忽略空白字符，并可以加入注释。以下两个正则表达式是等价的：

```
a = re.compile(r"""\d +  # the integral part
                   \.    # the decimal point
                   \d *  # some fractional digits""", re.X)
b = re.compile(r"\d+\.\d*")
```

re提供了众多模块方法用于完成正则表达式的功能。这些方法可以使用Pattern实例的相应方法替代，唯一的好处是少写一行re.compile()代码，但同时也无法复用编译后的Pattern对象。这些方法将在Pattern类的实例方法部分一起介绍。如上面这个例子可以简写为：

```
m = re.match(r'hello', 'hello world!')
print m.group()
```

re模块还提供了一个方法`escape(string)`，用于将`string`中的正则表达式元字符如`*/+/?`等之前加上转义符再返回，在需要大量匹配元字符时有那么一点用。

2.2. Match

Match对象是一次匹配的结果，包含了很多关于此次匹配的信息，可以使用Match提供的可读属性或方法来获取这些信息。

属性：

- **string**: 匹配时使用的文本。

- **re**: 匹配时使用的Pattern对象。
- **pos**: 文本中正则表达式开始搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。
- **endpos**: 文本中正则表达式结束搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。
- **lastindex**: 最后一个被捕获的分组在文本中的索引。如果没有被捕获的分组，将为None。
- **lastgroup**: 最后一个被捕获的分组的别名。如果这个分组没有别名或者没有被捕获的分组，将为None。

方法：

- **group([group1, ...]):**
  获得一个或多个分组截获的字符串；指定多个参数时将以元组形式返回。group1可以使用编号也可以使用别名；编号0代表整个匹配的子串；不填写参数时，返回group(0)；没有截获字符串的组返回None；截获了多次的组返回最后一次截获的子串。
- **groups([default]):**
  以元组形式返回全部分组截获的字符串。相当于调用group(1,2,...last)。default表示没有截获字符串的组以这个值替代，默认为None。
- **groupdict([default]):**
  返回以有别名的组的别名为键、以该组截获的子串为值的字典，没有别名的组不包含在内。default含义同上。
- **start([group]):**
  返回指定的组截获的子串在string中的起始索引（子串第一个字符的索引）。group默认值为0。
- **end([group]):**
  返回指定的组截获的子串在string中的结束索引（子串最后一个字符的索引+1）。group默认值为0。
- **span([group]):**
  返回(start(group), end(group))。
- **expand(template):**
  将匹配到的分组代入template中然后返回。template中可以使用\id或\g<id>、\g<name>引用分组，但不能使用编号0。\id与\g<id>是等价的；但\10将被认为是第10个分组，如果你想表达\1之后是字符'0'，只能使用\g<1>0。

```python
import re
m = re.match(r'(\w+) (\w+)(?P<sign>.*)', 'hello world!')

print "m.string:", m.string
print "m.re:", m.re
print "m.pos:", m.pos
print "m.endpos:", m.endpos
print "m.lastindex:", m.lastindex
print "m.lastgroup:", m.lastgroup

print "m.group(1,2):", m.group(1, 2)
print "m.groups():", m.groups()
print "m.groupdict():", m.groupdict()
print "m.start(2):", m.start(2)
print "m.end(2):", m.end(2)
print "m.span(2):", m.span(2)
print r"m.expand(r'\2 \1\3'):", m.expand(r'\2 \1\3')

### output ###
# m.string: hello world!
# m.re: <_sre.SRE_Pattern object at 0x016E1A38>
# m.pos: 0
# m.endpos: 12
# m.lastindex: 3
# m.lastgroup: sign
# m.group(1,2): ('hello', 'world')
# m.groups(): ('hello', 'world', '!')
# m.groupdict(): {'sign': '!'}
# m.start(2): 6
# m.end(2): 11
# m.span(2): (6, 11)
# m.expand(r'\2 \1\3'): world hello!
```

2.3. Pattern

Pattern对象是一个编译好的正则表达式，通过Pattern提供的一系列方法可以对文本进行匹配查找。

Pattern不能直接实例化，必须使用re.compile()进行构造。

Pattern提供了几个可读属性用于获取表达式的相关信息：

- pattern: 编译时用的表达式字符串。
- flags: 编译时用的匹配模式。数字形式。
- groups: 表达式中分组的数量。
- groupindex: 以表达式中有别名的组的别名为键、以该组对应的编号为值的字典，没有别名的组不包含在内。

```
import re
p = re.compile(r'(\w+) (\w+)(?P<sign>.*)', re.DOTALL)

print "p.pattern:", p.pattern
print "p.flags:", p.flags
print "p.groups:", p.groups
print "p.groupindex:", p.groupindex

### output ###
# p.pattern: (\w+) (\w+)(?P<sign>.*)
# p.flags: 16
# p.groups: 3
# p.groupindex: {'sign': 3}
```

实例方法[| re模块方法] :

- **match(string[, pos[, endpos]]) | re.match(pattern, string[, flags]):**
  这个方法将从string的pos下标处起尝试匹配pattern；如果pattern结束时仍可匹配，则返回一个Match对象；如果匹配过程中pattern无法匹配，或者匹配未结束就已到达endpos，则返回None。
  pos和endpos的默认值分别为0和len(string)；re.match()无法指定这两个参数，参数flags用于编译pattern时指定匹配模式。
  注意：这个方法并不是完全匹配。当pattern结束时若string还有剩余字符，仍然视为成功。想要完全匹配，可以在表达式末尾加上边界匹配符'$'。
  示例参见2.1小节。

- **search(string[, pos[, endpos]]) | re.search(pattern, string[, flags]):**
  这个方法用于查找字符串中可以匹配成功的子串。从string的pos下标处起尝试匹配pattern，如果pattern结束时仍可匹配，则返回一个Match对象；若无法匹配，则将pos加1后重新尝试匹配；直到pos=endpos时仍无法匹配则返回None。
  pos和endpos的默认值分别为0和len(string))；re.search()无法指定这两个参数，参数flags用于编译pattern时指定匹配模式。

```
# encoding: UTF-8
import re

# 将正则表达式编译成Pattern对象
pattern = re.compile(r'world')

# 使用search()查找匹配的子串，不存在能匹配的子串时将返回None
# 这个例子中使用match()无法成功匹配
match = pattern.search('hello world!')

if match:
    # 使用Match获得分组信息
    print match.group()

### 输出 ###
# world
```

- **split(string[, maxsplit]) | re.split(pattern, string[, maxsplit]):**
  按照能够匹配的子串将string分割后返回列表。maxsplit用于指定最大分割次数，不指定将全部分割。

```
import re

p = re.compile(r'\d+')
print p.split('one1two2three3four4')
```

```
5
6  ### output ###
7  # ['one', 'two', 'three', 'four', '']
```

- **findall(string[, pos[, endpos]]) | re.findall(pattern, string[, flags]):**
  搜索string，以列表形式返回全部能匹配的子串。

```
1  import re
2
3  p = re.compile(r'\d+')
4  print p.findall('one1two2three3four4')
5
6  ### output ###
7  # ['1', '2', '3', '4']
```

- **finditer(string[, pos[, endpos]]) | re.finditer(pattern, string[, flags]):**
  搜索string，返回一个顺序访问每一个匹配结果（Match对象）的迭代器。

```
1  import re
2
3  p = re.compile(r'\d+')
4  for m in p.finditer('one1two2three3four4'):
5      print m.group(),
6
7  ### output ###
8  # 1 2 3 4
```

- **sub(repl, string[, count]) | re.sub(pattern, repl, string[, count]):**
  使用repl替换string中每一个匹配的子串后返回替换后的字符串。
  当repl是一个字符串时，可以使用\id或\g<id>、\g<name>引用分组，但不能使用编号0。
  当repl是一个方法时，这个方法应当只接受一个参数（Match对象），并返回一个字符串用于替换（返回的字符串中不能再引用分组）。
  count用于指定最多替换次数，不指定时全部替换。

```
1  import re
2
3  p = re.compile(r'(\w+) (\w+)')
4  s = 'i say, hello world!'
5
6  print p.sub(r'\2 \1', s)
7
8  def func(m):
9      return m.group(1).title() + ' ' + m.group(2).title()
10
11 print p.sub(func, s)
12
13 ### output ###
14 # say i, world hello!
15 # I Say, Hello World!
```

- **subn(repl, string[, count]) |re.sub(pattern, repl, string[, count]):**
  返回 (sub(repl, string[, count]), 替换次数)。

```
1  import re
2
3  p = re.compile(r'(\w+) (\w+)')
4  s = 'i say, hello world!'
5
6  print p.subn(r'\2 \1', s)
7
8  def func(m):
9      return m.group(1).title() + ' ' + m.group(2).title()
```

```
10
11  print p.subn(func, s)
12
13  ### output ###
14  # ('say i, world hello!', 2)
15  # ('I Say, Hello World!', 2)
```

以上就是Python对于正则表达式的支持。熟练掌握正则表达式是每一个程序员必须具备的技能，这年头没有不与字符串打交道的程序了。笔者也处于初级阶段，与君共勉，^_^

另外，图中的特殊构造部分没有举出例子，用到这些的正则表达式是具有一定难度的。有兴趣可以思考一下，如何匹配不是以abc开头的单词，^_^

全文结束

分类: 学习笔记, Python

AstralWind
关注 - 1
粉丝 - 40
+加关注

« 博主前一篇：Python线程指南
» 博主后一篇：Python字符编码详解

posted @ 2010-07-04 23:56 AstralWind 阅读(21728) 评论(19) 编辑 收藏

发表评论

#1楼 2010-07-06 14:08 Lucker 回复 引用 查看

精彩，好文!

#2楼 2010-12-17 11:20 iTech 回复 引用 查看

超级好文，转载了

#3楼 2010-12-17 11:24 iTech 回复 引用 查看

此文实在是太好了，我转载了，点击率超级高啊，打破了历史水平啊！高的我都不好意思啊！我只是想收藏此精彩文章，没有其他的意思。

#4楼[楼主] 2010-12-17 21:35 AstralWind 回复 引用 查看

@iTech
对大家有帮助就好，也没打算靠版权挣钱呵呵~

#5楼 2011-05-06 10:59 青怪 回复 引用 查看

很好的文章，学习了。谢谢！

#6楼 2011-06-28 10:37 Sackula[未注册用户] 回复 引用

宇宙无敌好文~~~~~~~~

#7楼 2011-07-20 15:35 vijay 回复 引用 查看

不错，收藏并转载

#8楼 2011-07-31 03:50 dawb[未注册用户] 回复 引用

非常好，写的深入浅出

#9楼 2011-07-31 09:48 lidashuang 回复 引用 查看

非常不错，博主第一图是用什么工具做的？

#10楼[楼主] 2011-07-31 12:54 AstralWind 回复 引用 查看

@lidashuang
PowerPoint ：）

#11楼 2011-08-14 10:24 junjie020[未注册用户] 回复 引用

您好！
有一个问题关于python的条件表达式的

我发现，如果在条件表达式里面，使用别名，python是不行的（我使用的版本是2.7.2和3.2.1）
如 ：
reobj = re.match(r"(?P<N1>8)(?(?P=N1)8)", "88")

这样是无法匹配的，如果使用：
reobj = re.match(r"(8)(?(1)8", "88")
这样是可以的，不知道楼主是否有试过这样使用别名来引用条件表达式呢？还是我有没有写错的？

希望指教！

#12楼[楼主] 2011-08-15 09:43 AstralWind 回复 引用 查看

@junjie020
你好！抱歉回复晚了，是这样，你第一个正则

```
1 r"(?P<N1>8)(?(?P=N1)8)"
```

应该写成

```
1 r"(?P<N1>8)(?(N1)8)"
```

这样就没有问题了。(?(?P=N1)8)这是个不合法的正则表达式，无法编译。

#13楼 2011-10-03 22:24 shirne 回复 引用 查看

python的正则很强大，学习收藏了

#14楼 2011-11-07 11:59 eason@pku 回复 引用 查看

不错.

#15楼 2011-12-09 15:03 qqzhao 回复 引用 查看

不错...

#16楼 2011-12-19 17:16 你能记住几个 回复 引用 查看

您好，能不能分享下文中的图片和表格是用什么软件画的，很好看啊

#17楼[楼主] 2011-12-19 18:46 AstralWind 回复 引用 查看

@你能记住几个
ms office

#18楼 2011-12-25 17:47 会长 回复 引用 查看

好哇，我每次写正则表达式时，都来这看。脑子不行，记不住规则，只能随写随查

#19楼 2011-12-26 17:02 pyt123 回复 引用 查看

谢谢楼主分享

刷新评论列表 刷新页面 返回页首

发表评论

昵称： [登录] [注册]

主页：

邮箱：(仅博主可见)

评论内容：

登录 注册

[使用Ctrl+Enter键快速提交评论]

简洁阅读版式

最新IT新闻:

· 福布斯：CastleVille或有助于支撑Zynga股价
· 诺基亚和Burton合作 推基于传感器的滑雪应用
· 持续集成之“Everything is code”
· 苹果Genius Bar获得成功的7个原因
· 消息称诺基亚将自主生产Lumia 900手机
» 更多新闻...

最新知识库文章:

· 什么是闭包，我的理解
· 什么是闭包(Closure)？
· 设计师的品牌意识
· 如何成为“10倍效率”开发者
· 快速排序（Quicksort）的Javascript实现
» 更多知识库文章...