参考：<http://cuiqingcai.com/977.html>

[Python爬虫入门七之正则表达式](http://cuiqingcai.com/977.html)

[Python](http://cuiqingcai.com/category/technique/python)  [崔庆才](http://cuiqingcai.com/author/cqcre)  3年前 (2015-02-15)  187985浏览  [44评论](http://cuiqingcai.com/977.html#comments)

在前面我们已经搞定了怎样获取页面的内容，不过还差一步，这么多杂乱的代码夹杂文字我们怎样把它提取出来整理呢？下面就开始介绍一个十分强大的工具，正则表达式！

**1.了解正则表达式**

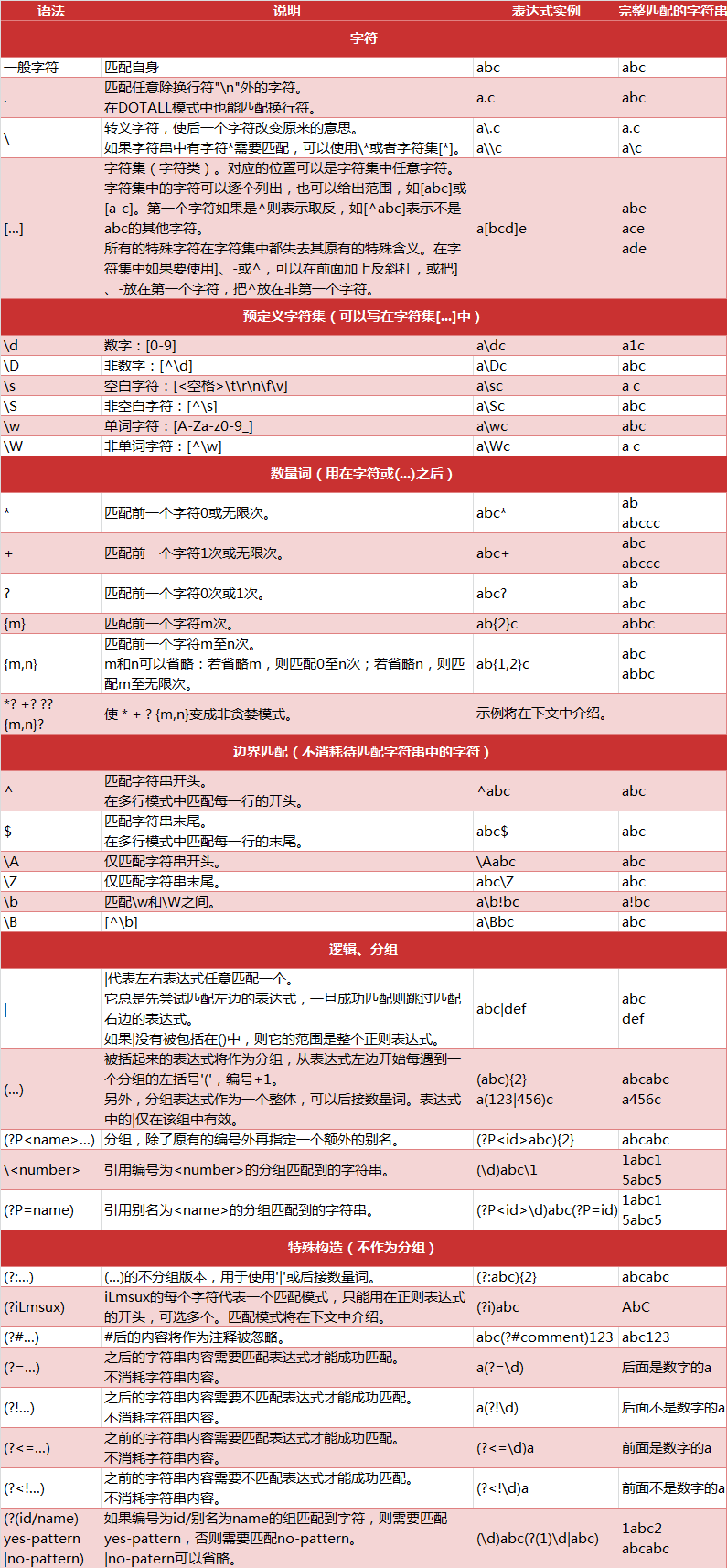
正则表达式是对字符串操作的一种逻辑公式，就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。

正则表达式是用来匹配字符串非常强大的工具，在其他编程语言中同样有正则表达式的概念，Python同样不例外，利用了正则表达式，我们想要从返回的页面内容提取出我们想要的内容就易如反掌了。

正则表达式的大致匹配过程是：  
1.依次拿出表达式和文本中的字符比较，  
2.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。  
3.如果表达式中有量词或边界，这个过程会稍微有一些不同。

**2.正则表达式的语法规则**

下面是Python中正则表达式的一些匹配规则，图片资料来自CSDN

[](http://qiniu.cuiqingcai.com/wp-content/uploads/2015/02/20130515113723855.png)

**3.正则表达式相关注解**

**（1）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”。

注：我们一般使用非贪婪模式来提取。

**（2）反斜杠问题**

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，妈妈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勒。

**4.Python Re模块**

Python 自带了re模块，它提供了对正则表达式的支持。主要用到的方法列举如下

|  |
| --- |
| #返回pattern对象  re.compile(string[,flag])  #以下为匹配所用函数  re.match(pattern, string[, flags])  re.search(pattern, string[, flags])  re.split(pattern, string[, maxsplit])  re.findall(pattern, string[, flags])  re.finditer(pattern, string[, flags])  re.sub(pattern, repl, string[, count])  re.subn(pattern, repl, string[, count]) |

在介绍这几个方法之前，我们先来介绍一下pattern的概念，pattern可以理解为一个匹配模式，那么我们怎么获得这个匹配模式呢？很简单，我们需要利用re.compile方法就可以。例如

|  |  |
| --- | --- |
| 1 | pattern = re.compile(r'hello') |

在参数中我们传入了原生字符串对象，通过compile方法编译生成一个pattern对象，然后我们利用这个对象来进行进一步的匹配。

另外大家可能注意到了另一个参数 flags，在这里解释一下这个参数的含义：

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。

可选值有：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | • re.I(全拼：IGNORECASE): 忽略大小写（括号内是完整写法，下同）  • re.M(全拼：MULTILINE): 多行模式，改变'^'和'$'的行为（参见上图）  • re.S(全拼：DOTALL): 点任意匹配模式，改变'.'的行为  • re.L(全拼：LOCALE): 使预定字符类 \w \W \b \B \s \S 取决于当前区域设定  • re.U(全拼：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  • re.X(全拼：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。 |

在刚才所说的另外几个方法例如 re.match 里我们就需要用到这个pattern了，下面我们一一介绍。

注：以下七个方法中的flags同样是代表匹配模式的意思，如果在pattern生成时已经指明了flags，那么在下面的方法中就不需要传入这个参数了。

**（1）re.match(pattern, string[, flags])**

这个方法将会从string（我们要匹配的字符串）的开头开始，尝试匹配pattern，一直向后匹配，如果遇到无法匹配的字符，立即返回None，如果匹配未结束已经到达string的末尾，也会返回None。两个结果均表示匹配失败，否则匹配pattern成功，同时匹配终止，不再对string向后匹配。下面我们通过一个例子理解一下

|  |
| --- |
| \_\_author\_\_ = 'CQC'  # -\*- coding: utf-8 -\*-    #导入re模块  import re    # 将正则表达式编译成Pattern对象，注意hello前面的r的意思是“原生字符串”  pattern = re.compile(r'hello')    # 使用re.match匹配文本，获得匹配结果，无法匹配时将返回None  result1 = re.match(pattern,'hello')  result2 = re.match(pattern,'helloo CQC!')  result3 = re.match(pattern,'helo CQC!')  result4 = re.match(pattern,'hello CQC!')    #如果1匹配成功  if result1:      # 使用Match获得分组信息      print result1.group()  else:      print '1匹配失败！'    #如果2匹配成功  if result2:      # 使用Match获得分组信息      print result2.group()  else:      print '2匹配失败！'    #如果3匹配成功  if result3:      # 使用Match获得分组信息      print result3.group()  else:      print '3匹配失败！'    #如果4匹配成功  if result4:      # 使用Match获得分组信息      print result4.group()  else:      print '4匹配失败！' |

运行结果

|  |  |
| --- | --- |
| 1  2  3  4 | hello  hello  3匹配失败！  hello |

匹配分析

1.第一个匹配，pattern正则表达式为’hello’，我们匹配的目标字符串string也为hello，从头至尾完全匹配，匹配成功。

2.第二个匹配，string为helloo CQC，从string头开始匹配pattern完全可以匹配，pattern匹配结束，同时匹配终止，后面的o CQC不再匹配，返回匹配成功的信息。

3.第三个匹配，string为helo CQC，从string头开始匹配pattern，发现到 ‘o’ 时无法完成匹配，匹配终止，返回None

4.第四个匹配，同第二个匹配原理，即使遇到了空格符也不会受影响。

我们还看到最后打印出了result.group()，这个是什么意思呢？下面我们说一下关于match对象的的属性和方法  
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：  
1.string: 匹配时使用的文本。  
2.re: 匹配时使用的Pattern对象。  
3.pos: 文本中正则表达式开始搜索的索引。值与Pattern.match()和Pattern.search()方法的同名参数相同。  
4.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。  
5.lastindex: 最后一个被捕获的分组（指的是正则表达式中存在括号分好的组）在文本中的索引。如果没有被捕获的分组，将为None。  
6.lastgroup: 最后一个被捕获的分组的别名（分组命名(?<name>abc)）。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：  
1.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。  
2.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。

3.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。

4.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。  
5.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。  
6.span([group]):  
返回(start(group), end(group))。  
7.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、\g引用分组，但不能使用编号0。\id与\g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\g0。

下面我们用一个例子来体会一下

|  |  |
| --- | --- |
| 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  32  33  34  35  36 | # -\*- coding: utf-8 -\*-  #一个简单的match实例    import re  # 匹配如下内容：单词+空格+单词+任意字符  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():", m.group()  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'\g \g\g'):", m.expand(r'\2 \1\3')    ### output ###  # m.string: hello world!  # m.re:  # 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）re.search(pattern, string[, flags])**

search方法与match方法极其类似，区别在于match()函数只检测re是不是在string的开始位置匹配，search()会扫描整个string查找匹配，match（）只有在0位置匹配成功的话才有返回，如果不是开始位置匹配成功的话，match()就返回None。同样，search方法的返回对象同match()返回对象的方法和属性。我们用一个例子感受一下

|  |
| --- |
| #导入re模块  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 = re.search(pattern,'hello world!')  if match:      # 使用Match获得分组信息      print match.group()  ### 输出 ###  # world |

**（3）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。我们通过下面的例子感受一下。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import re    pattern = re.compile(r'\d+')  print re.split(pattern,'one1two2three3four4')    ### 输出 ###  # ['one', 'two', 'three', 'four', ''] |

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

搜索string，以列表形式返回全部能匹配的子串。我们通过这个例子来感受一下

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import re    pattern = re.compile(r'\d+')  print re.findall(pattern,'one1two2three3four4')    ### 输出 ###  # ['1', '2', '3', '4'] |

**（5）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。我们通过下面的例子来感受一下

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

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

**（7）re.subn(pattern, repl, string[, count])**

返回 (sub(repl, string[, count]), 替换次数)。

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

**5.Python Re模块的另一种使用方式**

在上面我们介绍了7个工具方法，例如match，search等等，不过调用方式都是 re.match，re.search的方式，其实还有另外一种调用方式，可以通过pattern.match，pattern.search调用，这样调用便不用将pattern作为第一个参数传入了。

函数API列表

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | match(string[, pos[, endpos]]) | re.match(pattern, string[, flags])  search(string[, pos[, endpos]]) | re.search(pattern, string[, flags])  split(string[, maxsplit]) | re.split(pattern, string[, maxsplit])  findall(string[, pos[, endpos]]) | re.findall(pattern, string[, flags])  finditer(string[, pos[, endpos]]) | re.finditer(pattern, string[, flags])  sub(repl, string[, count]) | re.sub(pattern, repl, string[, count])  subn(repl, string[, count]) |re.sub(pattern, repl, string[, count]) |

具体的调用方法不必详说了，原理都类似，只是参数的变化不同。小伙伴们尝试一下吧~

小伙伴们加油，即使这一节看得云里雾里的也没关系，接下来我们会通过一些实战例子来帮助大家熟练掌握正则表达式的。

参考文章：此文章部分内容出自 [CNBlogs](http://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html)

转载请注明：[静觅](http://cuiqingcai.com/) » [Python爬虫入门七之正则表达式](http://cuiqingcai.com/977.html)