**Python 正则表达式入门（初级篇）**

本文主要为没有使用正则表达式经验的新手入门所写。  
转载请写明出处

**引子**

**首先说 正则表达式是什么？**

正则表达式，又称正规表示式、正规表示法、正规表达式、规则表达式、常规表示法（英语：Regular Expression，在代码中常简写为regex、regexp或RE），计算机科学的一个概念。正则表达式使用单个字符串来描述、匹配一系列匹配某个句法规则的字符串。在很多文本编辑器里，正则表达式通常被用来检索、替换那些匹配某个模式的文本。  
许多程序设计语言都支持利用正则表达式进行字符串操作。例如，在Perl中就内建了一个功能强大的正则表达式引擎。正则表达式这个概念最初是由Unix中的工具软件（例如sed和grep）普及开的。正则表达式通常缩写成“regex”，单数有regexp、regex，复数有regexps、regexes、regexen。  
引用自维基百科https://zh.wikipedia.org/wiki/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F

定义是定义，太正经了就没法用了。我们来举个栗子：假如你在写一个爬虫，你得到了  
一个网页的HTML源码。其中有一段

<html><body><h1>hello world<h1></body></html>

你想要把这个hello world提取出来，但你这时如果只会python 的字符串处理，那么第一反应可能是

s = <html><body><h1>hello world<h1></body></html>

start\_index = s.find('<h1>')

然后从这个位置向下查找到下一个<h1>出现这样做未尝不可，但是很麻烦不是吗。需要考虑多个标签，一不留神就多匹配到东西了，而如果想要非常准确的匹配到，又得多加循环判断，效率太低。

这时候，正则表达式就是首选的帮手。

**干货开始**

**入门级别**

接着说我们刚才那个例子。我们如果拿正则处理这个表达式要怎么做呢？

import re

key = r"<html><body><h1>hello world<h1></body></html>"#这段是你要匹配的文本

p1 = r"(?<=<h1>).+?(?=<h1>)"#这是我们写的正则表达式规则，你现在可以不理解啥意思

pattern1 = re.compile(p1)#我们在编译这段正则表达式

matcher1 = re.search(pattern1,key)#在源文本中搜索符合正则表达式的部分

print matcher1.group(0)#打印出来

你可以尝试运行上面的代码，看看是不是和我们想象的一样（博主是在python2.7环境下）发现代码挺少挺简单？往下看。而且正则表达式实际上要比看起来的那种奇形怪状要简单得多。

首先，从最基础的正则表达式说起。  
假设我们的想法是把一个字符串中的所有"python"给匹配到。我们试一试怎么做

import re

key = r"javapythonhtmlvhdl"#这是源文本

p1 = r"python"#这是我们写的正则表达式

pattern1 = re.compile(p1)#同样是编译

matcher1 = re.search(pattern1,key)#同样是查询

print matcher1.group(0)

看完这段代码，你是不是觉得：卧槽？这就是正则表达式？直接写上去就行？  
确实，正则表达式并不像它表面上那么奇葩，如果不是我们故意改变一些符号的含义时，你看到的就是想要匹配的。  
所以，先把大脑清空，先认为正则表达式就是和想要匹配的字符串长得一样。在之后的练习中我们会逐步进化

**初级**

0.无论是python还是正则表达式都是**区分大小写**的，所以当你在上面那个例子上把"python"换成了"Python"，那就匹配不到你心爱的python了。

1.重新回到第一个例子中那个<h1>hello world<h1>匹配。假如我像这么写，会怎么样？

import re

key = r"<h1>hello world<h1>"#源文本

p1 = r"<h1>.+<h1>"#我们写的正则表达式，下面会将为什么

pattern1 = re.compile(p1)

print pattern1.findall(key)#发没发现，我怎么写成findall了？咋变了呢？

有了入门级的经验，我们知道那两个<h1>就是普普通通的字符，但是中间的是什么鬼？  
*.字符在正则表达式代表着可以代表任何一个字符（包括它本身）*  
findall返回的是所有符合要求的元素列表，包括仅有一个元素时，它还是给你返回的列表。

机智如你可能会突然问：那我如果就只是想匹配"."呢？结果啥都给我返回了咋整？在正则表达式中有一个字符\，其实如果你编程经验较多的话，你就会发现这是好多地方的“转义符”。在正则表达式里，这个符号通常用来把特殊的符号转成普通的，把普通的转成特殊的23333（并不是特殊的“2333”，写完才发现会不会有脑洞大的想歪了）。  
举个栗子，[你真的想匹配"chuxiuhong@hit.edu.cn"这个邮箱](mailto:%E4%BD%A0%E7%9C%9F%E7%9A%84%E6%83%B3%E5%8C%B9%E9%85%8D%22chuxiuhong@hit.edu.cn%22%E8%BF%99%E4%B8%AA%E9%82%AE%E7%AE%B1)（我的邮箱），你可以把正则表达式写成下面这个样子：

import re

key = r"afiouwehrfuichuxiuhong@hit.edu.cnaskdjhfiosueh"

p1 = r"chuxiuhong@hit\.edu\.cn"

pattern1 = re.compile(p1)

print pattern1.findall(key)

发现了吧，我们在.的前面加上了转义符\，但是并不是代表匹配“\.”的意思，而是**只**匹配“.”的意思！  
不知道你细不细心，有没有发现我们第一次用.时，后面还跟了一个+？那这个加号是干什么的呢？  
其实不难想，我们说了“.字符在正则表达式代表着可以代表任何一个字符（包括它本身）”，但是"hello world"可不是一个字符啊。  
*+的作用是将前面一个字符或一个子表达式重复一遍或者多遍。*  
比方说表达式“ab+”那么它能匹配到“abbbbb”，但是不能匹配到"a"，它要求你必须得有个b，多了不限，少了不行。你如果问我有没有那种“有没有都行，有多少都行的表达方式”，回答是有的。  
*\*跟在其他符号后面表达可以匹配到它0次或多次*  
比方说我们在王叶内遇到了链接，可能既有http://开头的，又有https://开头的，我们怎么处理？

import re

key = r"http://www.nsfbuhwe.com and https://www.auhfisna.com"#胡编乱造的网址，别在意

p1 = r"https\*://"#看那个星号！

pattern1 = re.compile(p1)

print pattern1.findall(key)

输出

['http://', 'https://']

2.比方说我们有这么一个字符串"cat hat mat qat"，你会发现前面三个是实际的单词，最后那个是我胡编乱造的（上百度查完是昆士兰英语学院的缩写= =）。如果你本来就知道"at"前面是c、h、m其中之一时这才构成单词，你想把这样的匹配出来。根据已经学到的知识是不是会想到写出来三个正则表达式进行匹配？实际上不需要。因为有一种多字符匹方式  
*[]代表匹配里面的字符中的任意一个*  
还是举个栗子，我们发现啊，有的程序员比较过分，，在<html></html>这对标签上，大小写混用，老害得我们抓不到想要的东西，我们该怎么应对？是写16\*16种正则表达式挨个匹配？no

import re

key = r"lalala<hTml>hello</Html>heiheihei"

p1 = r"<[Hh][Tt][Mm][Ll]>.+?</[Hh][Tt][Mm][Ll]>"

pattern1 = re.compile(p1)

print pattern1.findall(key)

输出

['<hTml>hello</Html>']

我们既然有了范围性的匹配，自然有范围性的排除。  
*[^]代表除了内部包含的字符以外都能匹配*  
还是cat,hat,mat,qat这个例子，我们想匹配除了qat以外的，那么就应该这么写：

import re

key = r"mat cat hat pat"

p1 = r"[^p]at"#这代表除了p以外都匹配

pattern1 = re.compile(p1)

print pattern1.findall(key)

输出  
为了方便我们写简洁的正则表达式，它本身还提供下面这样的写法

| **正则表达式** | **代表的匹配字符** |
| --- | --- |
| [0-9] | 0123456789任意之一 |
| [a-z] | 小写字母任意之一 |
| [A-Z] | 大写字母任意之一 |
| \d | 等同于[0-9] |
| \D | 等同于[^0-9]匹配非数字 |
| \w | 等同于[a-z0-9A-Z\_]匹配大小写字母、数字和下划线 |
| \W | 等同于[^a-z0-9A-Z\_]等同于上一条取非 |

3.介绍到这里，我们可能已经掌握了大致的正则表达式的构造方式，但是我们常常会在实战中遇到一些匹配的不准确的问题。比方说：

import re

key = r"chuxiuhong@hit.edu.cn"

p1 = r"@.+\."#我想匹配到@后面一直到“.”之间的，在这里是hit

pattern1 = re.compile(p1)

print pattern1.findall(key)

输出结果

['@hit.edu.']

呦呵！你咋能多了呢？我理想的结果是@hit.，你咋还给我加量了呢？这是因为正则表达式默认是“贪婪”的，我们之前讲过，“+”代表是字符重复一次或多次。但是我们没有细说这个多次到底是多少次。所以它会尽可能“贪婪”地多给我们匹配字符，在这个例子里也就是匹配到最后一个“.”。  
我们怎么解决这种问题呢？只要在“+”后面加一个“？”就好了。

import re

key = r"chuxiuhong@hit.edu.cn"

p1 = r"@.+?\."#我想匹配到@后面一直到“.”之间的，在这里是hit

pattern1 = re.compile(p1)

print pattern1.findall(key)

输出结果

['@hit.']

加了一个“?”我们就将贪婪的“+”改成了懒惰的“+”。这对于[abc]+,\w\*之类的同样适用。

小测验：上面那个例子可以不使用懒惰匹配，想一种方法得到同样的结果

\*\*个人建议：在你使用"+","\*"的时候，一定先想好到底是用贪婪型还是懒惰型，尤其是当你用到范围较大的项目上时，因为很有可能它就多匹配字符回来给你！！！\*\*

为了能够准确的控制重复次数，正则表达式还提供  
{a,b}(代表a<=匹配次数<=b)

还是举个栗子，我们有sas,saas,saaas，我们想要sas和saas，我们怎么处理呢？

import re

key = r"saas and sas and saaas"

p1 = r"sa{1,2}s"

pattern1 = re.compile(p1)

print pattern1.findall(key)

输出

['saas', 'sas']

如果你省略掉{1,2}中的2，那么就代表至少匹配一次，那么就等价于？  
如果你省略掉{1,2}中的1，那么就代表至多匹配2次。

下面列举一些正则表达式里的元字符及其作用

| **元字符** | **说明** |
| --- | --- |
| . | 代表任意字符 |
| \ |  |
| [ ] | 匹配内部的任一字符或子表达式 |
| [^] | 对字符集和取非 |
| - | 定义一个区间 |
| \ | 对下一字符取非（通常是普通变特殊，特殊变普通） |
| \* | 匹配前面的字符或者子表达式0次或多次 |
| \*? | 惰性匹配上一个 |
| + | 匹配前一个字符或子表达式一次或多次 |
| +? | 惰性匹配上一个 |
| ? | 匹配前一个字符或子表达式0次或1次重复 |
| {n} | 匹配前一个字符或子表达式 |
| {m,n} | 匹配前一个字符或子表达式至少m次至多n次 |
| {n,} | 匹配前一个字符或者子表达式至少n次 |
| {n,}? | 前一个的惰性匹配 |
| ^ | 匹配字符串的开头 |
| \A | 匹配字符串开头 |
| $ | 匹配字符串结束 |
| [\b] | 退格字符 |
| \c | 匹配一个控制字符 |
| \d | 匹配任意数字 |
| \D | 匹配数字以外的字符 |
| \t | 匹配制表符 |
| \w | 匹配任意数字字母下划线 |
| \W | 不匹配数字字母下划线 |

初级篇先写到这里，之后会有中级篇来介绍子表达式，向前向后查找，回溯引用

# Python 正则表达式入门（中级篇）

## 上一篇我们说在这一篇里，我们会介绍子表达式，向前向后查找，回溯引用。到这一篇开始前除了回溯引用在一些场合不可替代以外，大部分情况下的正则表达式你应该都会写了。

## 1.子表达式

子表达式的概念特别好理解。其实它就是将几个字符的组合形式看做一个大的“字符”。不好理解？举个栗子：我们要匹配类似IP地址这种形式的字符（暂且不考虑数值范围的合理性，这个留作学完之后的思考题吧）。形如192.168.1.1这样的地址我们怎么写表达式呢？

答案一 \d+.?\d+.?\d+.?\d+

不好，一个是太繁琐，另一个是连位数都控制不了

答案二 \d+{1,3}.?\d+{1,3}.?\d+{1,3}.?\d+{1,3}

一般般，复杂但是起码能把位数控制在合理范围

答案三 (\d+{1,3}\.){3}\d+{1,3}\.

利用子表达式，将123.这种数字加小数点看做一个整体字符，对其规定重复匹配的次数，既简洁，效果又好。所以只要你将几个字符组合用圆括号括起来，那么你就可以把一个圆括号内的内容当做一个字符，外面可以加我们之前讲过的所有元字符来控制匹配。

## 2.向前向后查找

现在，我们终于来到了向前向后查找这一块。为什么说终于来到这了呢？还记得我们在初级篇最开始的例子吗？

假如你在写一个爬虫，你得到了一个网页的HTML源码。其中有一段html

<html><body><h1>hello world</h1></body></html>

你想要把这个hello world提取出来

import re

key = r"<html><body><h1>hello world</h1></body></html>"#这段是你要匹配的文本

p1 = r"(?<=<h1>).+?(?=</h1>)"#这是我们写的正则表达式规则，你现在可以不理解啥意思

pattern1 = re.compile(p1)#我们在编译这段正则表达式

matcher1 = re.search(pattern1,key)#在源文本中搜索符合正则表达式的部分

print matcher1.group(0)#打印出来

这个正则表达式

p1 = r"(?<=<h1>).+?(?=<h1>)"

看到(?<=<h1>) 和 (?=<h1>)了吗？第一个?<=表示在被匹配字符前必须得有<h1>，后面的?=表示被匹配字符后必须有<h1>

简单来说，就是你要匹配的字符是XX，但必须满足形式是AXXB这样的字符串，那么你就可以这样写正则表达式

p = r"(?<=A)XX(?=B)"

匹配到的字符串就是XX。并且，向前查找向后查找不需要必须同时出现。如果你愿意，可以只写满足一个条件。

所以你也不需要记住哪个是向前查找，哪个是向后查找。只要记住?<=后面跟着的是前缀要求，?=后面跟的是后缀要求。

本质上来说，向前查找和向后查找其实是匹配整个字符串，即AXXB，但返回时仅仅返回一个XX。也就是说，如果你愿意，完全可以避开向前向后查找的方式，直接匹配带有前后缀的字符串，然后做字符串切片处理。

## 3.回溯引用

不同于前面的向前向后查找，这一条有时候你未必绕的过去。在有些情况下，你还必须得用到回溯引用，所以你如果想拥有在实际应用中使用正则表达式，回溯引用是你应该了解和掌握的。

我们还是从最开始的例子来说。  
你原本要匹配<h1></h1>之间的内容，现在你知道HTML有多级标题，你想把每一级的标题内容都提取出来。你也许会这样写：

p = r"<h[1-6]>.\*?</h[1-6]>"

这样一来，你就可以将HTML页面内所有的标题内容全部匹配出来。即<h1></h1>到<h6></h6>的内容都可以被提取出来。但是我们之前说过，写正则表达式困难的不是匹配到想要的内容，而是尽可能的不匹配到不想要的内容。在这个例子中，很有可能你就会被下面这样的用例玩坏。

比方说

<h1>hello world</h3>

发现后面的</h3>了吗？我们不管是怎么写出来这样的标题的，但实实在在的是我们的正则表达式同样会把这里面的hello world匹配出来。这时候就是回溯引用的重要作用。下面就是一个示例：

import re

key = r"<h1>hello world</h3>"

p1 = r"<h([1-6])>.\*?</h\1>"

pattern1 = re.compile(p1)

m1 = re.search(pattern1,key)

print m1.group(0)#这里是会报错的，因为匹配不到，你如果将源字符串改成</h1>

结尾就能看出效果

看到\1了吗？原本那个位置应该是[1-6]，但是我们写的是\1，我们之前说过，转义符\干的活就是把特殊的字符转成一般的字符，把一般的字符转成特殊字符。普普通通的数字1被转移成什么了呢？在这里1表示第一个子表达式，也就是说，它是动态的，是随着前面第一个子表达式的匹配到的东西而变化的。比方说前面的子表达式内是[1-6]，在实际字符串中找到了1，那么后面的\1就是1，如果前面的子表达式在实际字符串中找到了2，那么后面的\1就是2。

类似的，\2,\3,....就代表第二个第三个子表达式。

所以回溯引用是正则表达式内的一个“动态”的正则表达式，让你根据实际的情况变化进行匹配。

中级篇就到这里，其实正则表达式还有很多细节还没有写出来，也有很多元字符我没有交代，但掌握了纲要，懂得原理之后剩下的就类似于查表构造这种活了。

建议看到这的朋友看看《正则表达式必知必会》，初级篇和这篇中有几个例子也是取材于此。

标签: [python](http://www.cnblogs.com/chuxiuhong/tag/python/), [正则表达式](http://www.cnblogs.com/chuxiuhong/tag/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/)