实用Python程序设计

郭 炜

微信公众号

微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕！

正则表达式

正则表达式

- 可以用来判断某个字符串是否符合某种模式，比如判断某个字符串是不是邮箱地址，网址，电话号码，身份证号……

- 可以用来在文本中寻找并抽取符合某种模式的字符串，比如电子邮件地址，电话号码，网址，身份证号……，找出三国演义中所有孔明提到曹操的场合都说了些啥。

正则表达式的基本概念和构成

加拿大班芙国家公园

正则表达式

正则表达式是个某些字符有特殊含义字符串，
表示一种字符串的模式（格式），如：

```
"abc"                匹配 "abc"
"b.?p.*k"            匹配 "bapk", "bpabk" ...
"\d{3}[a-zA-Z]+.(\d{2}|N/A)\s\1" 匹配 ????
```

可以用相关函数求给定字符串和正则表达式的匹配情况

正则表达式中的功能字符

字符/组合	匹配的模式	正则表达式	匹配的字符串
.	除'\n'外的任意一个字符，包括汉字（多行匹配方式下也能匹配'\n'）	'a.b'	'acb' 'adb' 'a(b' .....
*	量词。表示左边的字符可出现0次或任意多次	'a*b'	'b' 'ab' 'aaaab' .....
?	量词。表示左边的字符必须出现0次或1次	'ka?b'	'kb' 'kab'
+	量词。表示左边的字符必须出现1次或更多次	'ka+b'	'kab' 'kaaab' .....
{m}	量词。m是整数。表示左边的字符必须且只能出现m次	'ka{3}a'	'kaaaa'
{m,n}	量词。m,n是整数。表示左边的字符必须出现至少m次，最多n次。n也可以不写，表示出现次数没有上限	'ka{1,3}b'	'kab' 'kaab' 'kaaab'

字符/组合	匹配的模式	正则表达式	匹配的字符串
\d	一个数字字符，等价于[0-9]	'a\db'	'a2b' 'a3b' .....
\D	一个非数字的字符，等价于[^\d],[^0-9]	'a\Db'	'acb' .....
\s	一个空白字符，如空格，\t,\r,\n等	'a\sb'	'a b' 'a\nb' .....
\S	一个非空白字符	'a\Sb'	'akb' .....
\w	一个单词字符：包括汉字或大小写英文字母，数字，下划线，或其它语言的文字	'a\wb'	'a_b' 'a中b' .....
\W	一个不是单词字符的字符	'a\Wb'	'a?b' .....
|	A|B 表示能匹配A或能匹配B均算能匹配	'ab|c'	'ab' 'c'

```
print("\\s\s\S\w\W\d\D") #python字符串中，\s等不是转义字符，都是两个字符
#>>\s\s\S\w\W\d\D
```

正则表达式中的特殊字符

正则表达式中常见的特殊字符有以下几个：

. + ? * $ [ ] ( ) ^ { } \

如果要在正则表达式中表示这几个字符本身，就应该在其前面加'\'。

正则表达式中的特殊字符

正则表达式	匹配的字符串
'a\$b'	'a$b'
'a*b'	'a*b'
'a\[\]b'	'a[]b'
'a\.*b'	'ab' 'a.b' 'a..b' .....
'a\\\\b'	'a\\b' （注意：此字符串长度为3，中间那个字符是'\'，即r'a\b'）
r'a\\b'	r'a\b' （r'a\b'等价于'a\\b'）

```
print("\*\$\.\+\[\]\(\)\?\^\{\}") #这些都不是转义字符，都是两个字符
#>>\*\$\.\+\[\]\(\)\?\^\{\}
```

范围符号[ ]
和
量词

圣彼得堡炮兵博物馆

范围符号[XXX]的用法

用以表示“此处必须出现一个某某范围内的字符”，或者“此处必须出现一个字符，但不可以是某某范围内的字符”

[XXX]的用法

[a2c]	匹配'a','2'，'c'之一	's[a2c]k'	'sak' 's2k' 'sck'
[a-zA-Z]	匹配任一英文字母	'b[a-zA-Z]k'	'bak' 'bUk' .....
[\da-z\?]	匹配一个数字或小写英文字母或'?'	'b[\da-z\?]k'	'b0k' 'bck' 'b?k' .....
[^abc]	匹配一个非'a','b','c'之一的字符	'b[^abc]k'	匹配所有能匹配'b.k'的字符串，除了： 'bak' 'bbk' 'bck'
[^a-f0-3]	匹配一个非英文字母'a'到'f'，也非数字'0'到'3'的字符	略	略

匹配汉字

汉字的unicode编码范围是 4e00-9fa5 （16进制），因此 [\u4e00-\u9fa5] 即表示一个汉字

```
print('\u4e00\u4e01\u4e88\u9fa5') #>>一丁予龥
```

量词的用法

'.+'	匹配任意长度不为0且不含'\n'的字符串。'+'表示左边的'.'代表的任意字符出现1次或更多次。不要求出现的字符都必须一样。
'.*'	匹配任意不含'\n'的字符串，包括空串
'[\dac]+'	匹配长度不为0的由数字或'a',c'构成的串，如'451a','a21c78ca'
'\w{5}'	匹配长度为5的由字母或数字或汉字构成的串，如'高大abc','33我a1'

正则表达式示例

`[1-9]\d*`	正整数
`-[1-9]\d*`	负整数
`-?[1-9]\d*|0`	整数
`[1-9]\d*|0`	非负整数
`-?([1-9]\d*\.\d*[1-9]|0\.\d*[1-9]|0)`	左右都没有多余0的小数
`\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*`	邮箱

邮箱示例： a.b.c.ddef+sfd@pku.edu.cn

正则表达式的函数

加拿大班芙国家公园

使用正则表达式

- 使用正则表达式要：

```
import re
```

re.match函数

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

•从字符串string的起始位置匹配一个模式pattern

•flags　标志位，用于控制模式串的匹配方式，如：是否区分大小写，多行匹配等等,如 re.M | re.I　表示忽略大小写，且多行匹配

•成功则返回一个匹配对象，否则返回None

re.match函数

```
import re
def match(pattern,string):
    x = re.match(pattern,string)
    if x != None:
        print(x.group()) #x.group是匹配上的字符串
    else:
        print("None")
match("a c","a cdkgh")          #>>a c
match("abc","kabc")             #>>None  虽然有abc，但不是在起始位置
match("a\tb*c","a\tbbcde")      #>>a    bbc  b出现0次或任意多次，然后跟c
match("ab*c","ac")              #>>ac
match("a\d+c","ac")             #>>None  b出现1次或更多次，然后跟c
match("a\d{2}c","a34c")         #>>a34c
match("a\d{2,}c","a3474884c")   #>>a3474884c
match(".{2}bc","cbcd")          #>>None  bc前面要有2个字符
```

re.match函数

```
match(".{2}bc","bcbcdbc")        #>>bcbc
match("ab.*","ab")               #>>ab     b后面可以没字符或任意字符
match("ab.*","abcd")             #>>abcd
match("\d?b.*","1bcd")
match("\d?b.*","bbcd")
match("a?bc.*","abbbcd")
match("a.b.*","abcd")
match("a.b.*","aeb")
match("a.?b.*","aebcdf")         #>>aebcdf
#a和b之间没字符或有任意一个字符均可
match("a.+b.*","aegsfb")
match("a.+b.*","abc")
match("a高.+k","a高大kcd")       #>>a高大b
```

re.match函数

```
match(".{2}bc","bcbcdbc")      #>>bcbc
match("ab.*","ab")             #>>ab     b后面可以没字符或任意字符
match("ab.*","abcd")           #>>abcd
match("\d?b.*","1bcd")         #>>1bcd         数字应出现0次或1次
match("\d?b.*","bbcd")         #>>bbcd
match("a?bc.*","abbbcd")       #>>None         b太多了
match("a.b.*","abcd")          #>>None       a和b之间必须要有一个字符
match("a.b.*","aeb")           #>>aeb
match("a.?b.*","aebcdf")       #>>aebcdf
#a和b之间没字符或有任意一个字符均可
match("a.+b.*","aegsfb")       #>>aegsfb
match("a.+b.*","abc")          #>>None
match("a高.+k","a高大kcd")     #>>a高大b
```

re.search函数

- **re.search(pattern, string, flags = 0)**

- 查找字符串中可以匹配成功的子串。

- 若匹配成功，则返回匹配对象；若无法匹配，则返回None。

re.search函数

```
import re
def search(pattern,string):
    x = re.search(pattern,string)
    if x != None:
        print(x.group(),x.span())      #输出子串及起止位置
    else:
        print("None")
search("a.+bc*","dbaegsfbcef")         #>>aegsfbc (2, 9)
search("a.+bc*","bcdbaegsfbccc")       #>>aegsfbccc (4, 13)
search("a.?bc*d","dabccdc")            #>>abccd (1, 6)
search("aa","baaaa")                   #>>aa (1, 3)
search("\([1-9]+\)","ab123(0456)(789)45ab")     #>>(789) (11, 16)
search("[1-9]\d+","ab01203d45")        #>>1203 (3, 7)
```

re.findall函数

- **re.findall(pattern, string, flags = 0)**

- 查找字符串中所有和模式匹配的子串(不重叠)放入列表。一个子串都找不到就返回空表 []

re.finall函数

```
import re
print(re.findall('\d+',"this is 334 what me 774gw"))  #>>['334','774']
print(re.findall('[a-zA-Z]+',"A dog has 4 legs.这是true"))
#>>['A', 'dog', 'has', 'legs', 'true']
print(re.findall('\d+',"this is good."))      #>>[]
print(re.findall("aaa","baaaa"))              #>>['aaa']
```

re.finditer函数

- **re.finditer (pattern, string, flags = 0)**

- 查找字符串中所有和模式匹配的子串(不重叠)，每个子串对应于一个匹配对象，返回匹配对象的序列(准确说是“可调用迭代器”）

- 如果找不到匹配子串，返回值也不为None或空表。假设返回值为r,则list(r) 为[]

re.finditer函数

```
import re
s = '233[32]88ab<433>(21)'
m = '\[\d+\]|<\d+>'                    # | 表示 '或'
for x in re.finditer(m,s):             #x是匹配对象
    print(x.group(),x.span())
i  = 0
for y in re.finditer(m,"aaaaa"):
    i += 1                  #不会被执行
```

输出:

```
[32] (3, 7)
<433> (11, 16)
```

边界符号

荷兰阿姆斯特丹库肯霍立夫公园

边界符号

\A	表示字符串的左边界，即要求从此往左边不能有任何字符
\Z	表示字符串的右边界，即要求从此往右边不能有任何字符
^	与\A同。但多行匹配模式下还可以表示一行文字的左边界
$	与\Z同。但多行匹配模式下还可以表示一行文字的右边界

边界符号本身不会和任何字符匹配。

Python字符串 '\A'，'\Z' 都是两个字符，而不是像'\n'那样的一个字符。

```
print("\A\Z")   #>>\A\Z
```

边界符号

```
import re
def search(pattern,string):
    x = re.search(pattern,string)
    if x != None:
        print(x.group())
    else:
        print("None")
m = "\Ahow are"
search(m,"ahow are you")     #>>None
search(m,"how are you")      #>>how are
m = "are you\Z"
search(m,"how are you?")     #>>None
search(m,"how are you")      #>>are you
search("a.+bc*","dbaegsfbcef")       #>>aegsfbc
search("a.+bc*\Z","dbaegsfbcef")     #>>None
search("a.+bc*\Z","dbaegsfbccc")     #>>aegsfbccc
```

边界符号

\b　　表示此处应为单词的左边界或右边界，即不可是单词字符
\B　　表示此处不允许是单词的左边界或右边界，即必须是单词字符
边界符号本身不会和任何字符匹配。

正则表达式的边界符号'\b' 是两个字符。但是在Python字符串中，'\b'和'\t','\n'类似，是一个字符(Backspace)。因此在正则表达式中使用边界符号\b，要写 '\\b' 。如果写 '\\\b'，则连续的两个'\'被看作是一个普通的'\'，不会和后面的'b'一起被当作字符组合，变成边界符号'\b'。

```
print("\\b")  #>>\b
```

Python字符串 '\B' 是两个字符

边界符号

```
pt = "ka\\b.*"
search(pt,"ka")          #>>ka
search(pt,"kax")         #>>None
search(pt,"ka?d")        #>>ka?d
pt = ".*\\bka\\b"
search(pt,"ka")          #>>ka
search(pt,"ska?")        #>>None
search(pt,"b?ka?")       #>>b?ka
```

边界符号

```
m = r"\bA.*N\b T"                #等价于 m = "\\bA.*N\\b T"
search(m,"Ass$NC TK")            #>>None
search(m,"this Ass$N TK")        #>>Ass$N T
m = "\BA.*N\B\w T"
search(m,"this Ass$N TK")        #>>None
search(m,"thisAss$NM TK")        #>>Ass$NM T
search(m,"Ass$NM TK")            #>>None
search(m,"xAss$NM TK")           #>>Ass$NM T
pt = "\\b高兴"
search(pt,"我高兴")              #None
search(pt,"我 高兴")             #高兴
```

匹配选项

波兰华沙老城

正则表达式匹配选项

- **re.search(pattern, string, flags = 0) flags就是匹配选项**

- 匹配函数如search,match可以通过可选的选项标志来控制匹配的模式。多个标志可以通过按位 OR(|) 来指定。如 re.I | re.M 。下表是一部分标志

修饰符	描述
re.I	使匹配对大小写不敏感
re.M	多行匹配，使 ^能匹配行的开头，$能匹配行的结尾 ，同 re.MULTILINE
re.S	使 . 匹配包括换行在内的所有字符 (重要!)，无此标志则 .不匹配 \n 同re.DOTALL

正则表达式匹配选项

```
import re
def search(pattern,string ,flags = 0):
    #flags参数如果省略不写，其值就是 0
    x = re.search(pattern,string,flags)
    if x != None:
        print(x.group())
    else:
        print("None")
```

正则表达式匹配选项

```
search("a.+b.*","acBc")                    #>>None
search("a.+b.*","acBc",re.I)               #>>acBc
m = "^h[a-z]w are"
g = re.findall(m,"how are you\nhew are me")
print(g)                                   #>>['how are']
g = re.findall(m,"how are you\nhew ARE me",re.M|re.I)
#多行匹配模式且不分大小写
print(g)                          #>>['how are', 'hew ARE']
m = "are you$"
search(m,"how are you\nThis")                        #>>None
search(m,"how are you\nThis",re.M)                   #>>are you
search("a.+b","a\ncdb")                              #>>None
search("a.+b","a\ncdbe",re.S)
```

分组

圣彼得堡彼得霍夫宫

分组(...)

➢括号中的表达式是一个分组。多个分组按左括号从左到右从1开始依次编号

```
import re
x = re.search('[a-z]+(\d+)[a-z]+',"ab 123d hello553world47")
print(x.group(1))                #>>553
m = "(((ab*)c)d)e"
r = re.match(m,"abcdefg")
print(r.group(0))                #>>abcde      group(0)等价于group()
print(r.group(1))                #>>abcd
print(r.group(2))                #>>abc
print(r.group(3))                #>>ab
print(r.groups())                #>>('abcd', 'abc', 'ab')
```

分组(...)

➢括号中的表达式是一个分组。多个分组按左括号从左到右从1开始依次编号

```
import re
m = "(ab*)(c(d))e"
r = re.match(m,"abcdefg")
print(r.groups())          #>>('ab', 'cd', 'd')
print(r.group(0))          #>>abcde
print(r.group(1))          #>>ab
print(r.group(2))          #>>cd
print(r.group(3))          #>>d
```

分组(...)

- 在分组的右边可以通过分组的编号引用该分组所匹配的子串

```
import re
m = r'(((ab*)c)d)e\3'        #r表示字符串里的'\'不再转义
#要求 ab*cde后面跟着3号分组在本次匹配中匹配上的子串
r = re.match(m,"abbbcdeabbbkfg") # 红色部分少一个b则不能匹配
print(r.group(3))                 # abbb
print(r.group())                  # abbbcdeabbb
```

分组(...)

➢在分组的右边可以通过分组的编号引用该分组所匹配的子串

```
pt = 'a(.)\\1*b'    #或 pt = r'a(.)\1*b'
print(re.search(pt,'kacccccb').group())  #>>acccccb
print(re.search(pt,'kaxxxxb').group())   #>>axxxxb
print(re.search(pt,'kaxb').group())      #>>axb
x = re.search(pt,'kaxyb')
if x != None:
    print(x.group())                     #不会执行
```

分组(...)

➢分组作为一个整体，后面可以跟量词

```
import re
m = "(((ab*)+c)d)e"
r = re.match(m,"ababcdefg")
print(r.groups())          #>>('ababcd', 'ababc', 'ab')

r = re.match(m,"abacdefg")
print(r.groups())          #>>('abacd', 'abac', 'a')
```

不要求分组的多次出现必须匹配相同字符串

re.findall和分组

- 在正则表达式中没有分组时，re.findall返回所有匹配子串构成的列表。有且只有一个分组时，re.findall返回的是一个子串的列表，每个元素是一个匹配子串中分组对应的内容。

```
import re
m = '[a-z]+(\d+)[a-z]+'
x = re.findall(m,"13 bc12de ab11 cd320ef")
print(x)        #>>['12', '320']
```

findall和分组

➢在正则表达式中有超过一个分组时，re.findall返回的是一个元组的列表，每个元组对应于一个匹配的子串，元组里的元素，依次是1号分组、2号分组、3号分组.....匹配的内容

```
import re
m = '(\w+) (\w+)'
r = re.match(m,"hello world")
print(r.groups())                #>>('hello', 'world')
print(r.group(1))                #>>hello
print(r.group(2))                #>>world
r = re.findall(m,"hello world, this is very good")
#找出由所有能匹配的子串的 groups() 构成的元组，互相不重叠
print(r) #>>[('hello', 'world'), ('this', 'is'), ('very', 'good')]
```

re.sub和分组

➢re.sub(模式串，替换串，母串)　用于替换匹配的子串

```
import re
s = 'abc.xyz'
print(re.sub(r'(.*)\.(.*)', r'\2.\1', s))        #>>xyz.abc

# 用  '(.*)\.(.*)' 匹配 s,并用 '\2.\1' 替换s中被匹配的子串
# \2代表第2个分组匹配的内容，\1是第一个分组匹配的内容
```

| 的用法

圣彼得堡阿芙乐尔号巡洋舰

"|" 的用法

- 表示“或”，如果没有放在"()"中，则起作用范围是直到整个正则表达式开头或结尾或另一个 "|"

```
"\w{4}ce|c\d{3}|p\w"
```

可以匹配：

"c773"

"ab12ce"

"pk"

"|" 的用法

- 从左到右短路匹配（匹配上一个后就不计算是否还能匹配后面的）

```
import re
pt = "\d+\.\d+|\d+"
print(re.findall(pt,"12.34 this is 125"))
#>>['12.34', '125']
pt = "aa|aab"
print(re.findall(pt,"aabcdeaa12aab"))
#>>['aa', 'aa', 'aa']
```

"|" 的用法

➢'|' 也可以用于分组中，起作用范围仅限于分组内

```
import re
m ="(((ab*)+c|12)d)e"
print(re.findall(m,'ababcdefgKK12deK'))
#>>[('ababcd', 'ababc', 'ab'), ('12d', '12', '')]
for x in re.finditer(m,'ababcdefgKK12deK'):
    print(x.groups())
#>>('ababcd', 'ababc', 'ab')
#>>('12d', '12', None)
m = '\[(\d+)\]|<(\d+)>'
for x in re.finditer(m,'233[32]88ab<433>'):
    print(x.group(),x.groups())
#>>[32] ('32', None)
#>><433> (None, '433')
```

贪婪模式
和
懒惰模式

美国圣地亚哥中途岛号航母

量词的贪婪模式

- 量词 +,*,?,{m,n} 默认匹配尽可能长的子串

```
import re
print(re.match("ab*", "abbbbk").group())      #>>abbbb
print(re.findall("<h3>(.*)</h3>",
                 "<h3>abd</h3><h3>bcd</h3>"))
#>>['abd</h3><h3>bcd']
print(re.findall('\(.+\)',
                   "A dog has(have a).这(哈哈)true()me"))
#>>['(have a).这(哈哈)true()']
```

量词的非贪婪(懒惰)模式

- 在量词 +,*,?,{m,n} 后面加 '?' 则匹配尽可能短的字符串 。

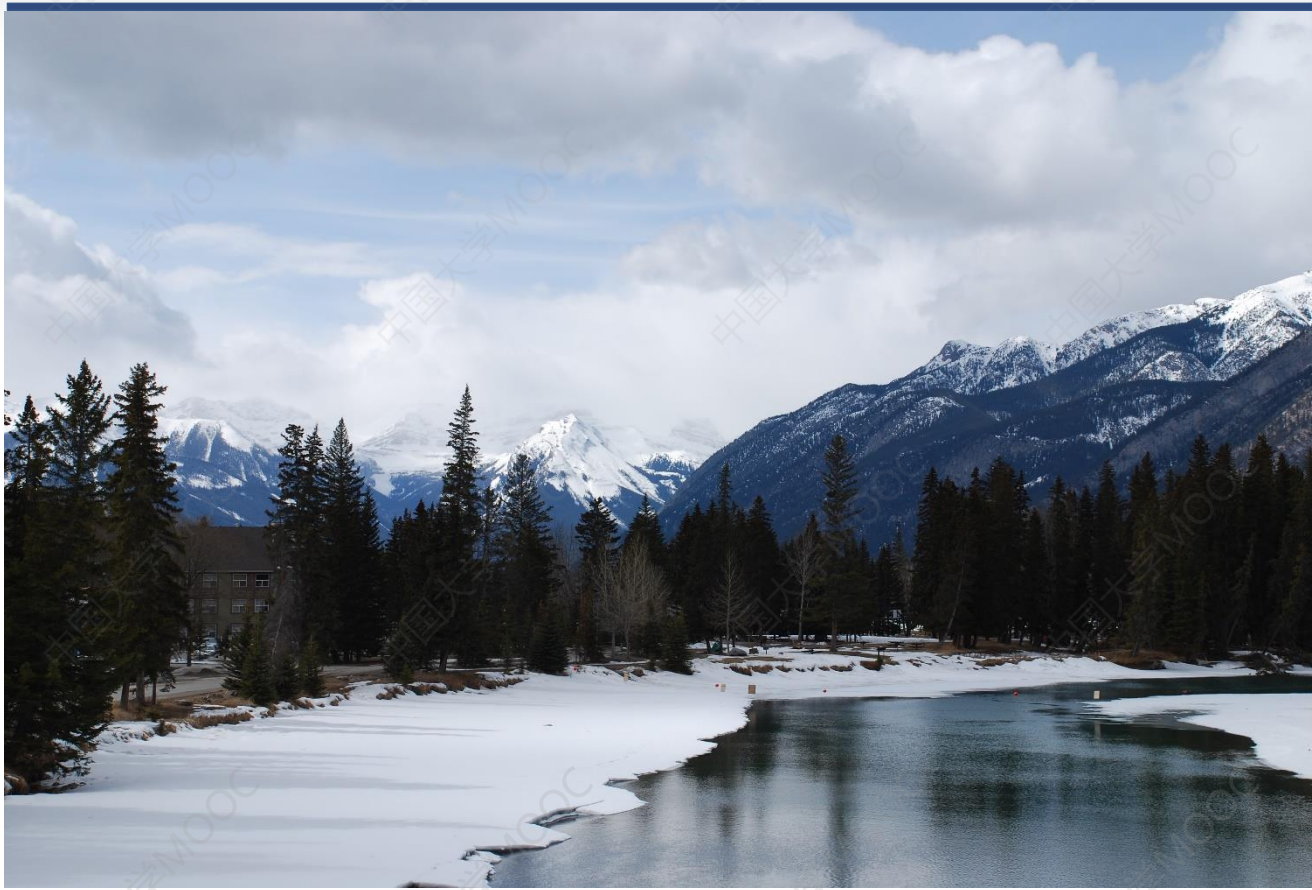
```
import re
m = "a.*?b"
for k in re.finditer(m,"aabab"):
    print(k.group(),end=" ") #>>aab ab
m = "<h3>.*?</h3>"
a = re.match(m,"<h3>abd</h3><h3>bcd</h3>")
print(a.group())    #><h3>abd</h3>
m = "<h3>.*?[M|K]</h3>"
a = re.match(m,"<h3>abd</h3><h3>bcK</h3>")
print(a.group())    #><h3>abd</h3><h3>bcK</h3>
```

量词的非贪婪(懒惰)模式

- 在量词 +,*,?,{m,n} 后面加 '?' 则匹配尽可能短的字符串 。

```
import re
print(re.findall('\d+?',"this is 334 what me 774gw"))
#>>['3', '3', '4', '7', '7', '4']
print(re.findall('[a-zA-Z]+?',"A dog has 4 legs.这是true"))
#>>['A', 'd', 'o', 'g', 'h', 'a', 's', 'l', 'e', 'g', 's', 't',
   'r', 'u', 'e']
print(re.findall('\(.*?\)',"A dog has(have).这(哈哈)true()me"))
#>>['(have)', '(哈哈)', '()']
```

匹配对象



加拿大班芙国家公园

“匹配对象”（匹配成功时的返回结果）的属性

- **string**: 匹配时使用的母串。
- **lastindex**: 最后一个被匹配的分组的编号(不是最大编号）。如果没有被匹配的分组，将为None。

“匹配对象”的函数

- **group([n1,n2, …]):**

  - 获得一个或多个分组匹配的字符串；指定多个参数时将以元组形式返回。n1,n2...可以使用编号也可以使用名字；

  - 编号0代表整个匹配的子串(与模式里面的"()"无关)；
  - group() 等价于 group(0)；
  - 没有匹配字符串的组返回None；
  - 匹配了多次的组返回最后一次匹配的子串。

“匹配对象”的函数

- **groups([default]):**
  以元组形式返回全部分组匹配的字符串。相当于调用group(1,2,…last)。default表示没有匹配字符串的组以这个值替代，默认为None。
- **groupdict([default]):**
  返回以有名字的组的名字为键、以该组匹配的子串为值的字典，没有名字的组不包含在内。default含义同上。
- **start([group]):**
  返回指定的组匹配的子串在string中的起始位置。group默认值为0。
- **end([group]):**
  返回指定的组匹配的子串在string中的结束位置（子串最后一个字符的位置 +1）。group默认值为0。

“匹配对象”的函数

- **span([group]):**
  返回(start(group), end(group))。
  group可以是组编号，也可以是组名字，缺省为0

“匹配对象”的函数

```
m = re.match(r'(\w+) (\w+)(.)', 'hello world!ss')
print( m.string)   #>>hello world!ss
print(m.lastindex) #>>3
print(m.group(0,1,2,3))#>>('hello world!', 'hello', 'world', '!')
print(m.groups())  #>>('hello', 'world', '!')
print(m.start(2)) #>>6
print(m.end(2))    #>>11
print(m.span(2))   #>>(6, 11)
```

应用实例

瑞典斯德哥尔摩

诸葛亮口中的曹操

- 找出三国演义中，在诸葛亮提到曹操的所有场景，他是怎么说的

模式：

孔明曰：“倘曹操引兵来到，当如之何？”
孔明曰：“公瑾主意欲降操，甚为合理。”
孔明答曰：“曹操乃汉贼也，又何必问？”
孔明笑曰：“今操引百万之众，. . . . . ”

：“”都是中文的

诸葛亮口中的曹操

```
import re
f = open("c:/tmp/三国演义utf8.txt","r",encoding="utf-8")
txt = f.read()
f.close()
pt = "(孔明.{0,2}曰：“[^”]*(曹操|曹贼|操贼|曹阿瞒|操).*?”)"
a = re.findall(pt,txt)
print(len(a))          #>>58
for x in a: #x形如：  ('孔明答曰：“曹操乃汉贼也，又何必问？”', '操')
   print(x[0])
```

抽取ip地址、邮箱、网址

```
import re
ipadr = r"\b((25[0-5]|2[0-4]\d|((1\d{2})|([1-
   9]?\d)))\.){3}(25[0-5]|2[0-4]\d|((1\d{2})|([1-9]?\d)))\b"
mailbox = r"\b[a-zA-Z0-9_-]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-
   ]+)+\b"
url = r'http://[a-zA-Z0-9][-a-zA-Z0-9]{0,62}(\.[a-zA-Z0-9][-
   a-zA-Z0-9]{0,62})+(/[-a-zA-Z0-9]+)*\b'
s = "My ip is 223.44.3.4, this is
   http://www.pku.edu.cn/python/new, http://www.sohu.com my
   mailbox is guo_wei@pku.edu.cn. ok?"
m = re.search(ipadr,s)
if m != None:
    print(m.group())   #>>223.44.3.4
```

抽取ip地址、邮箱、网址

```
m = re.search(mailbox,s)
if m != None:
    print(m.group())          #>>guo_wei@pku.edu.cn
for x in re.finditer(url,s):
    print(x.group())
#>>
http://www.pku.edu.cn/python/new
http://www.sohu.com
```

简化正则表达式的编写

一般来说，要写一个精确的正则表达式，比如写一个正则表达式来匹配ip地址，要做到匹配它的字符串一定是ip地址，且ip地址一定能匹配它，是比较困难的。

正则表达式可以写得宽容一些，即ip地址一定能匹配它，但是能匹配它的不一定是ip地址。对匹配的字符串，另外再做一些额外的判断排除掉非ip地址，这样做比写精确的正则表达式容易。

例如：`'\d+\.(\d{1,3}\.){2}\d+'`

然后再split以后手动判断每一段是不是都小于255