Python的正则表达式

Python使用re模块提供了正则表达式处理的能力。

常量

常量	说明
re.M re.MULTILINE	多行模式
re.S re.DOTALL	单行模式
re.I re.IGNORECASE	忽略大小写
re.X re.VERBOSE	忽略表达式中的空白字符

使用 `| 位或` 运算开启多种选项

方法

编译

`re.compile(pattern, flags=0)`
设定flags，编译模式，返回正则表达式对象regex。
pattern就是正则表达式字符串，flags是选项。正则表达式需要被编译，为了提高效率，这些编译后的结果被保存，下次使用同样的pattern的时候，就不需要再次编译。

re的其它方法为了提高效率都调用了编译方法，就是为了提速。

单次匹配

`re.match(pattern, string, flags=0)`
`regex.match(string[, pos[, endpos]])`
match匹配从字符串的开头匹配，regex对象match方法可以重设定开始位置和结束位置。返回match对象

`re.search(pattern, string, flags=0)`
`regex.search(string[, pos[, endpos]])`
从头搜索直到第一个匹配，regex对象search方法可以重设定开始位置和结束位置，返回match对象

`re.fullmatch(pattern, string, flags=0)`
`regex.fullmatch(string[, pos[, endpos]])`
整个字符串和正则表达式匹配

```
import re

s = '''bottle\nbag\nbig\napple'''
```

```
for i,c in enumerate(s, 1):
    print((i-1, c), end='\n' if i%10==0 else ' ')
print()

(0, 'b')(1, 'o')(2, 't')(3, 't')(4, 'l')(5, 'e')(6, '\n')(7, 'b')(8, 'a')(9, 'g')
(10, '\n')(11, 'b')(12, 'i')(13, 'g')(14, '\x07')(15, 'p')(16, 'p')(17, 'l')(18, 'e')

# match方法
print('--match--')
result = re.match('b', s) # 找到一个就不找了
print(1, result) # bottle
result = re.match('a', s) # 没找到，返回None
print(2, result)
result = re.match('^a', s, re.M) # 依然从头开始找，多行模式没有用
print(3, result)
result = re.match('^a', s, re.S) # 依然从头开始找
print(4, result)
# 先编译，然后使用正则表达式对象
regex = re.compile('a')
result = regex.match(s) # 依然从头开始找
print(5, result)
result = regex.match(s, 15) # 把索引15作为开始找
print(6, result) # apple
print()

# search方法
print('--search--')
result = re.search('a', s) # 扫描找到匹配的第一个位置
print(7, result) # apple
regex = re.compile('b')
result = regex.search(s, 1)
print(8, result) # bag
regex = re.compile('^b', re.M)
result = regex.search(s) # 不管是不是多行，找到就返回
print(8.5, result) # bottle
result = regex.search(s, 8)
print(9, result) # big

# fullmatch方法
result = re.fullmatch('bag', s)
print(10, result)
regex = re.compile('bag')
result = regex.fullmatch(s)
print(11, result)
result = regex.fullmatch(s, 7)
print(12, result)
result = regex.fullmatch(s, 7, 10)
print(13, result) # 要完全匹配，多了少了都不行，[7, 10)
```

全文搜索

`re.findall(pattern, string, flags=0)`
`regex.findall(string[, pos[, endpos]])`
对整个字符串，从左至右匹配，返回所有匹配项的列表

`re.finditer(pattern, string, flags=0)`
`regex.finditer(string[, pos[, endpos]])`
对整个字符串，从左至右匹配，返回所有匹配项，返回迭代器。
注意每次迭代返回的是match对象。

```
import re

s = '''bottle\nbag\nbig\nable'''
for i,c in enumerate(s, 1):
    print((i-1, c), end='\n' if i%10==0 else ' ')
print()

(0, 'b') (1, 'o') (2, 't') (3, 't') (4, 'l') (5, 'e') (6, '\n') (7, 'b') (8, 'a') (9, 'g')
(10, '\n') (11, 'b') (12, 'i') (13, 'g') (14, '\n') (15, 'a') (16, 'b') (17, 'l') (18, 'e')

# findall方法
result = re.findall('b', s)
print(1, result)
regex = re.compile('^b')
result = regex.findall(s)
print(2, result)
regex = re.compile('^b', re.M)
result = regex.findall(s, 7)
print(3, result) # bag big
regex = re.compile('^b', re.S)
result = regex.findall(s)
print(4, result) # bottle
regex = re.compile('^b', re.M)
result = regex.findall(s, 7, 10)
print(5, result) # bag

# finditer方法
result = regex.finditer(s)
print(type(result))
print(next(result))
print(next(result))
```

匹配替换

`re.sub(pattern, replacement, string, count=0, flags=0)`
`regex.sub(replacement, string, count=0)`
使用pattern对字符串string进行匹配，对匹配项使用repl替换。
replacement可以是string、bytes、function。

`re.subn(pattern, replacement, string, count=0, flags=0)`
`regex.subn(replacement, string, count=0)`
同sub返回一个元组（new_string， number_of_subs_made）

```
import re

s = '''bottle\nbag\nbig\napple'''
for i,c in enumerate(s, 1):
    print((i-1, c), end='\n' if i%8==0 else ' ')
print()

(0, 'b') (1, 'o') (2, 't') (3, 't') (4, 'l') (5, 'e') (6, '\n') (7, 'b') (8, 'a') (9, 'g')
(10, '\n')(11, 'b')(12, 'i')(13, 'g')(14, '\n')(15, 'a')(16, 'p')(17, 'p')(18, 'l')(19, 'e')

# 替换方法
regex = re.compile('b\wg')
result = regex.sub('magedu', s)
print(1, result) # 被替换后的字符串
result = regex.sub('magedu', s, 1) # 替换1次
print(2, result) # 被替换后的字符串

regex = re.compile('\s+')
result = regex.subn('\t', s)
print(3, result) # 被替换后的字符串及替换次数的元组
```

分割字符串

字符串的分割函数split，太难用，不能指定多个字符进行分割。

`re.split(pattern, string, maxsplit=0, flags=0)`
re.split分割字符串

```
import re

s = """
os.path.abspath(path)
normpath(join(os.getcwd(), path)).
"""

# 把每行单词提取出来
print(s.split())  # 做不到['os.path.abspath(path)', 'normpath(join(os.getcwd(),', 'path)).']
print(re.split('[\.()\s,]+', s))
```

分组

使用小括号的pattern捕获的数据被放到了组group中。

match、search函数可以返回match对象；findall返回字符串列表；finditer返回一个个match对象

如果pattern中使用了分组，如果有匹配的结果，会在match对象中

1. 使用group(N)方式返回对应分组，1到N是对应的分组，0返回整个匹配的字符串
2. 如果使用了命名分组，可以使用group('name')的方式取分组
3. 也可以使用groups()返回所有组
4. 使用groupdict() 返回所有命名的分组

```
```

```
import re

s = '''bottle\nbag\nbig\napple'''
for i,c in enumerate(s, 1):
    print((i-1, c), end='\n' if i%10==0 else ' ')
print()

# 分组
regex = re.compile('(b\w+)')
result = regex.match(s) # 从头匹配一次
print(type(result))
print(1, 'match', result.groups())

result = regex.search(s, 1) # 从指定位置向后匹配一次
print(2, 'search', result.groups()) #

# 命名分组
regex = re.compile('(b\w+)\n(?P<name2>b\w+)\n(?P<name3>b\w+)')
result = regex.match(s)
print(3, 'match', result)
print(4, result.group(3), result.group(2), result.group(1))
print(5, result.group(0).encode()) # 0 返回整个匹配字符串，即match
print(6, result.group('name2'), result.group('name3'))
print(6, result.groups())
print(7, result.groupdict())

result = regex.findall(s) # 返回什么，有几项？
for x in result:
    print(type(x), x)

regex = re.compile('(?P<head>b\w+)')
result = regex.finditer(s)
for x in result:
    print(type(x), x, x.group(), x.group('head'))
```

练习

匹配邮箱地址

```
test@hot-mail.com
v-ip@magedu.com
web.manager@magedu.com.cn
super.user@google.com
a@w-a-com
```

匹配html标记内的内容

```
<a href='http://www.magedu.com/index.html' target='_blank'>马哥教育</a>
```

匹配URL

```
http://www.magedu.com/index.html
https://login.magedu.com
file:///ect/sysconfig/network
```

匹配二代中国身份证ID

```
321105700101003
321105197001010030
11210020170101054X
17位数字+1位校验码组成
前6位地址码，8位出生年月，3位数字，1位校验位（0-9或X）
```

判断密码强弱

```
要求密码必须由 10-15位 指定字符组成：
十进制数字
大写字母
小写字母
下划线
要求四种类型的字符都要出现才算合法的强密码

例如：Aatb32_67mnq，其中包含大写字母、小写字母、数字和下划线，是合格的强密码
```

单词统计word count

```
对sample文件进行单词统计，要求使用正则表达式
```