

第8章 正则表达式

提纲

- 8.1 正则表达式概述
- 8.2 特殊符号与字符
- 8.3 正则表达式模块re



8.1 正则表达式概述

- ❑ 在信息化时代的今天，计算机需要做大量的**文本与数据处理**等工作。
- ❑ 我们可能不清楚这些文本或数据的具体内容，因此以某种可被计算机自动识别和处理的**模式**表达出来是非常有用的。
- ❑ **正则表达式**提供了这样的模式，是一个特殊的字符序列，帮助你方便的检查一个字符串是否与某种模式匹配。
- ❑ 在Python中内置了一个**re模块**以支持正则表达式。
- ❑ 正则表达式有两种基本的操作，分别是**匹配和替换**。
 - 匹配就是在一个文本字符串中**搜索匹配**一特殊表达式
 - 替换就是在一个字符串中查找并**替换**匹配一特殊表达式的字符串

8.2 特殊符号和字符

□ 基本字符

字符	描述
<code>text</code>	匹配 <code>text</code> 字符串
<code>.</code>	匹配除换行符之外的任意一个单个字符
<code>^</code>	匹配一个字符串的开头
<code>\$</code>	匹配一个字符串的末尾

8.2 特殊符号和字符

匹配限定符

- 用来约束匹配的次数

限定符	描述
*	重复匹配前表达式零次或多次
+	重复匹配前表达式一次或多次
?	重复匹配前表达式零次或一次
+或*后跟?	表示非贪婪匹配，即尽可能少的匹配，如*?重复任意次，但尽可能少重复；
. * ?	表示匹配任意数量的重复，但是在能使整个匹配成功的前提下使用最少的重复。
{m}	精确重复匹配前表达式m次
{m,}	至少重复匹配前表达式m次
{m,n}	至少重复匹配前表达式m次，至多重复匹配前表达式n次

8.2 特殊符号和字符

□ 组和运算符

组	描述
[...]	匹配集合内的字符，如[a-z],[1-9]或[,./;']
[^...]	匹配除集合外的所有字符，相当于取反操作
A B	匹配表达式A或B，相当于OR操作
(...)	表达式分组，每对括号为一组，如([a-b]+)([A-Z]+)([1-9]+)
(?P<name>...)	分组的命名模式，取此分组的内容时可以使用索引，也可以使用name

8.2 特殊符号和字符

□ 特殊字符序列

字符	描述
\A	只匹配字符串的开始
\b	匹配一个单词边界。例如\b匹配字符边界，food\b匹配"food"、"zoofood"，而和"foodies"不匹配。
\B	匹配一个非单词边界。又譬如，\bthe\b仅匹配单词the
\d	匹配任意十进制数字字符，等价于r'[0-9]'
\D	匹配任意非十进制数字字符，等价于r'^0-9]'
\s	匹配任意空格字符（空格符、tab制表符、换行符、回车、换页符、垂直线符号）
\S	匹配任意非空格字符
\w	匹配任意字母数字及下划线
\W	匹配任意非字母数字及下划线
\Z	仅匹配字符串的尾部
\\	匹配反斜线字符

8.2 特殊符号和字符

□ 正则表达式例子

正则表达式模式	匹配的字符串
<code>b[aeiu]t</code>	<code>bat,bet,bit,but</code>
<code>[cr][23]</code>	一个包含2个字符的字符串：第一个是“r”或“c”,第二个是“2”或“3”，例如： <code>c2、r3、r2</code> 等

正则表达式模式	匹配的字符串
<code>z.[0-9]</code>	字符“z”，后面跟任意一个字符，然后是一个十进制数字
<code>[r-u][env-y][us]</code>	“r”到“u”中的任意一个字符，后面跟的是“e”到“y”中的任意一个字符，在后面是字符“u”或“s”
<code>[^aeiou]</code>	一个非元音字符
<code>[^\t\n]</code>	除TAB制表符和换行符以外的任意一个字符

8.2 特殊符号和字符

□ 正则表达式例子

正则表达式模式	匹配的字符串
<code>[dn]ot?</code>	字符“d”或“o”,后面是一个“o”,最后是最多一个字符“t”,即do、no、dot、not
<code>0?[1-9]</code>	1~9中的任意一位数字,前面可能还有一个“0”
<code>[0-9]{15,16}</code>	15位或16位数字表示
<code>[KQRBNP][a-h][1-8]-[a-h][1-8]</code>	表示国际象棋合法的棋盘移动。即“K”、“Q”、“R”、“B”、“N”或“P”等字母后面加上两个用连字符连在一起的“a1”到“h8”之间的棋盘坐标
正则表达式模式	匹配的字符串
<code>\w+-\d+</code>	一个由字母或数字组成的字符串和至少一个数字,两部分中间由连字符连接
<code>[A-Za-z]\w*</code>	第一个字符是字母,其余字符是字母或数字
<code>\d{3}-\d{3}-\d{4}</code>	电话号码,前面带区号前缀,例如800-555-1212
<code>\w+@\w+\.com</code>	简单的XXX@YYY.com格式的电子邮件地址

8.3 正则表达式模块re

□ re模块

- 通过**re模块**，可在python中利用正则表达式对字符串进行**搜索、抽取和替换**操作。
- re 模块使 Python 语言拥有全部的正则表达式功能

□ 核心函数和方法

- **模块的函数或正则表达式对象的方法**compile

函数	描述
compile(pattern,flags=0)	对正则表达式 pattern 进行编译， flags 是可选表示符，并返回一个 regex 对象

8.3 正则表达式模块re

□ 两种调用方式

- 利用正则表达式对象的方法
- 直接调用模块re的函数
- 以findall为例

```
2 #regexobj.py
3 import re
4 text="Joodie is a handsome boy. He is cool, clever.."
5 rex=re.compile('\w*oo\w*')#查找所有包含'oo'的单词
6 #1.正则表达式对象的方法findall
7 a=rex.findall(text)
8 print("正则表达式对象的方法:",a)
9 #2.模块re的函数findall
10 b=re.findall('\w*oo\w*',text)
11 print("模块re的函数:",b)
12 |
```

正则表达式对象的方法: ['Joodie', 'cool']

模块re的函数: ['Joodie', 'cool']

8.3 正则表达式模块re

□ 核心函数和方法(续)

- re模块的函数或regex(正则表达式)对象的方法(函数与方法同名)
- 返回一个匹配对象或列表

函数/方法	说明
match (pattern, string, flags=0)	尝试用正则表达式模式pattern匹配字符串string,flags是可选标识符, 如果匹配成功, 则返回一个匹配对象; 否则返回None
search (pattern, string, flags=0)	在字符串string中搜索正则模式pattern的第一次出现, flags是可选标识符, 如果匹配成功, 则返回一个匹配对象; 否则返回None
findall (pattern, string[, flags])	在字符串string中搜索正则表达式模式pattern的所有(非重复)出现; 返回一个匹配对象的列表
finditer (pattern, string[, flags])	和findall()相同, 但返回的不是列表而是迭代器; 对于每个匹配, 该迭代器返回一个匹配对象
split (pattern, string, max=0)	根据pattern中的分隔符把字符串string分割为一个列表, 返回成功匹配的列表, 最多分割max次(默认是分割所有匹配的地方)
sub (pattern, repl, string, max=0)	把字符串string中所有匹配正则表达式pattern的地方替换成字符串repl, 如果max的值没有给出, 则对所有匹配的地方进行替换

8.3 正则表达式模块re

□ 核心函数和方法(续)

● 匹配对象的方法

方法	描述
group (num=0)	返回全部匹配对象
groups (default=None)	返回一个包含全部匹配的子组的元组

● 常用属性或处理标志

标志	描述
re.I 或 re.IGNORECASE	忽略表达式的大小写来匹配文本。

8.3-1 compile()编译正则表达式

- ❑ **re.compile()** 可以把正则表达式编译成一个 **regex对象**。
- ❑ 对于那些经常使用的正则表达式，这样的预编译可以提高一定的效率。
 - regex对象的一个例子：

```
2 #regexobj.py
3 import re
4 text = "JGood is a handsome boy, he is cool, clever, and so on..."
5 regex = re.compile(r'\w*oo\w*')
6 #查找所有包含'oo'的单词
7 print(regex.findall(text))
8
9
```

```
['JGood', 'cool']
```


8.3-2 match()

- ❑ 在处理正则表达式时，match()函数只匹配字符串的起始部分。
- ❑ match()函数尝试匹配字符串的起始部分。
- ❑ 如果匹配成功，返回一个Match对象；否则返回None。

```
2 #rematch_search.py
3 import re
4 pattern=re.compile('h.llo')
5 result1=re.match(pattern,'hello C') #1
6 result2=re.match(pattern,'C hello') #2
7 result3=re.match(pattern,'helo') #3
8 result4=re.match(pattern,'hello world') #4
9 print(result1)
10 if result1: print('1匹配:',result1.group())
11 else: print('1匹配:失败')
12 if result2: print('2匹配:',result2.group())
13 else: print('2匹配:失败')
14 if result3: print('3匹配:',result3.group())
15 else: print('3匹配:失败')
16 if result4: print('4匹配:',result4.group())
17 else: print('4匹配:失败')
```

```
<_sre.SRE_Match object; span=(0, 5), match='hello'>
1匹配: hello
2匹配:失败
3匹配:失败
4匹配: hello
```

8.3-3 search()

❑ search和match工作方式一样，区别在于：

- `re.match(pattern, string, flags=0)`从开头开始匹配string
- `re.search(pattern, string, flags=0)`从anywhere 开始匹配string

❑ search和match的相同之处：

- 均返回`_sre.SRE_Match`对象，如果不能匹配，返回None。

```
19 #rematch_search.py
20 import re
21 res1=re.match('[a-z]oo','seafood')
22 print('match匹配:',res1)
23 if res1: print(res1.group())
24 res2=re.search('[a-z]oo','seafood')
25 print('search匹配:',res2)
26 if res2: print(res2.group())
27
```

match匹配: None

search匹配: <_sre.SRE_Match object; span=(3, 6), match='foo'>
foo

8.3-4 匹配对象的group方法

□ **group**: 获取子模式(组)的匹配项

```
3 import re
4 pat=re.compile(r'www\.(.*)\.(.*)') #用()表示组1, 组2,...
5 m=pat.match('www.dxy.com')
6 m0=m.group()           #默认为0, 表示匹配整个字符串
7 m1=m.group(1)          #返回给定组1匹配的子字符串
8 m2=m.group(2)          #返回给定组2匹配的子字符串
9 print(m0)
10 print(m1)
11 print(m2)
12 |
```

```
www.dxy.com
dxy
com
```

8.3-5 findall()和finditer()

❑ **findall()**搜索string，以列表形式返回全部能匹配的子串。

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

❑ **finditer()**搜索string，返回顺序访问每个匹配结果的迭代器。

● **finditer(string[,flags])** 或 **re.finditer(pattern, string[, flags])**

```
1 #refind.py
2 import re
3 p=re.compile('\d+')
4 f1=re.findall(p, 'one1two2three3four4')
5 print('findall:',f1)
6 f2=re.finditer(p, 'one1two2three3four4')
7 print('finditer:',f2)
8 for m in f2: print(m,':',m.group())
```

```
findall: ['1', '2', '3', '4']
```

```
finditer: <callable_iterator object at 0x0000021098F1BE10>
```

```
<_sre.SRE_Match object; span=(3, 4), match='1'> : 1
```

```
<_sre.SRE_Match object; span=(7, 8), match='2'> : 2
```

```
<_sre.SRE_Match object; span=(13, 14), match='3'> : 3
```

```
<_sre.SRE_Match object; span=(18, 19), match='4'> : 4
```

8.3-6 sub()

□ **sub**: substitute的缩写，表示替换

- **p.sub(repl, string[, count, flags])**(**p**为正则表达式对象) 或 **re.sub(pattern, repl, string[, count, flags])**
- 通过正则表达式，可以实现比普通字符串的**replace**更加强大的**替换功能**
- **re.sub()**的各参数详解
 - ✓ **pattern**: 表示正则中的模式字符串；需要知道的是：若出现反斜杠加数字（\d），则用对应的匹配组（**matched group**）去替换。
 - ✓ **repl**: 就是**replacement**，用来替换的字符串； **repl**可以是**字符串**，也可以是**函数**。
 - ✓ **string**: 即表示要被处理，要被替换的那个**string**字符串。
 - ✓ **count**: 指定要处理的部分；默认全部处理。

8.3-6 sub()

❑ sub方法的例子1: repl是字符串，不含'\d'

```
15 #resub.py
16 import re
17 r1=re.sub('a','A','abcasd') #找到a用A替换
18 print(r1)
19 pat = re.compile('a')
20 r2=pat.sub('A','abcasd')
21 print(r2)
22 pat=re.compile('(blue|white|red)')
23 r3=pat.sub('colour','blue socks and red shoes')
24 print(r3)
25
```

AbcAsd

AbcAsd

colour socks and colour shoes

8.3-6 sub()

❑ sub方法的例子2: repl是字符串, 含'\d'

```
26 pat=re.compile(r'www\.(.*)\..{3}') #正则表达式
27 r4=pat.match('www.dxy.com').group(1)
28 print(r4)
29 r5=re.sub(r'www\.(.*)\..{3}',r'\1','hello,www.dxy.com')
30 print(r5)
31 r6=pat.sub(r'\1','hello,www.dxy.com')
32 print(r6)
33 #r'1'是第一组的意思
34 #匹配找到符合规则的"www.dxy.com",用组1字符串去替换整个匹配。
```

```
dxy
hello,dxy
hello,dxy
```

8.3-6 sub()

❑ sub方法的例子3: repl是字符串，含多个'\d'

```
36 pat=re.compile(r'(\w+) (\w+)')      #正则表达式
37 s='hello world ! hello hz !'
38 r7=pat.findall('hello world ! hello hz !')
39 print(r7)
40 r8=pat.sub(r'\2 \1',s)
41 #通过正则得到组1(hello)，组2(world)
42 #再通过sub去替换。即组1替换组2，组2替换组1，调换位置。
43 print(r8)
44
```

```
[('hello', 'world'), ('hello', 'hz')]
world hello ! hz hello !
```


8.3-6 sub()

□ sub方法的例子4: repl是函数

● 字符串中的数字+111

```
46 def _add111(matched):
47     intStr = matched.group("number") #group(name), 由?P<name>定义
48     intValue = int(intStr)
49     addedValue = intValue + 111
50     addedValueStr = str(addedValue)
51     return addedValueStr
52 inputStr = "hello 123 world 456"
53 replacedStr = re.sub("(?P<number>\d+)", _add111, inputStr)
54 #模式字符串中?P<name>是给正则匹配中的组起一个名字,
55 #无论它出现在第几个括号中, 都是这个名字, 与圆括号的顺序无关,
56 #它出现在的那对括号内, 就表示这个括号内匹配的内容就是group(name)
57 print("replacedStr=", replacedStr) #结果: hello 234 world 567
58
```

```
replacedStr= hello 234 world 567
```

8.3-6 split()

□ `split()` 用于字符串分割:

- `split(string[, maxsplit])` 或 `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，若不指定，则将全部分割。

```
3 #resplit.py
4 import re
5 p=re.compile(r'\d+')
6 print(p.split('one1two2three3four4'))
7
```

```
['one', 'two', 'three', 'four', '']
```

- 如果分隔符没有使用由特殊符号表示的正则表达式来匹配多个模式，那 `re.split()` 和 `string.split()` 的执行过程是一样的:

```
8 s="str1:str2:str3"
9 print(re.split(':',s))
10 print(s.split(':'))
```

```
['str1', 'str2', 'str3']
['str1', 'str2', 'str3']
```


8.3-6 split()

❑ re.split()例子：比string.split()功能更为强大

```
13 #resplit.py
14 import re
15 print(re.split(',', 'a,s,d,**asd'))#返回列表
16 pat = re.compile(',')
17 print(pat.split('a,s,d,**asd'))
18 print(re.split('[, *]+', 'a , s ,d ,,**asd'))
19 #正则匹配: [, *]+,1个或多个逗号、空格、*等符号均可作为分隔符。
20 print(re.split('[, *]+', 'a , s ,d ,,**asd', maxsplit=2))
21 # maxsplit 最多分割次数
22
```

```
['a', 's', 'd', '**asd']
['a', 's', 'd', '**asd']
['a', 's', 'd', 'asd']
['a', 's', 'd ', '**asd']
```