

## 第三章 正则表达式

### 本章导读

字符串是程序设计中经常涉及的数据结构，而正则表达式是字符串处理的重要手段，其设计思想是用一种描述语言定义字符串规则，凡符合规则的字符串，可认为是“匹配成功”，否则“匹配失败”。

正则表达式是Python的标准库(Python Standard Library)，其名称为re，用import载入。

### 学习目标：

- 1. 掌握Python语言中正则表达式的基本用法；
- 2. 掌握match()方法和groups()方法；
- 3. 掌握贪婪匹配与懒惰匹配；

### 本章目录

- 第一节 快速了解
  - 1、从示例出发
  - 2、贪婪与懒惰
- 第二节 正则表达式
- 第三节 修饰符
- 第四节 表达式编译
- 第五节 常用方法与属性
  - 1、字符串切分：re.split()
  - 2、全匹配：re.findall()
  - 3、迭代器查找：re.finditer()
  - 4、字符串替换：re.sub()
- 第六节 匹配对象
  - 1、分组方法：re.group()
  - 2、字典分组方法：re.groupdict()
  - 3、匹配位置：re.span()
- 第七节 小结

### 第一节 快速了解

#### 1、从示例出发

例程3-1是Python中正则表达式的简单应用示例，图3-1是其执行效果。正则表达式是Python的标准模块，无需安装即可使用，即import re即可使用，如例程第3行代码所示。第5行以及第9行是Python正则表达式的常见用法。第5行代码中的re.search("\d{5}",strA)用于搜索"\d{5}"在字符串strA中的第1次出现，返回值为match对象；第9行代码中的re.findall("\d{5}",strA)用于搜索"\d{5}"在字符串strA中的全部匹配，返回值为list。代码中的"\d{5}"即为正则表达式，\d表示数字，{5}表示出现5次。

例程3-1

第1行	strA="消费维权投诉电话12315物价举报电话12358地税纳税服务热线12366劳动和社会保障局12333....."
第2行	
第3行	import re
第4行	
第5行	match=re.search(r"\d{5}",strA); #找到第1个匹配
第6行	if match is None:print("匹配失败! ")
第7行	else:print(match.group())#输出：12315
第8行	

```

第9行 match=re.findall("\d{5}",strA);
第10行 if match is None:print("匹配失败! ")
第11行 else:print(match)#输出: ['12315', '12358', '12366', '12333']
第12行
第13行 strB=""北京大学简称北大, 诞生于1898年, 初名京师大学堂, 是中国近代第一所国立大学,
第14行 是第一个以大学之名创办的学校, 其成立标志着中国近代高等教育的开端。北大是中国近代
第15行 以来唯一以国家最高学府身份创立的学校, 最初也是国家最高教育行政机关, 行使教育部职能,
第16行 统管全国教育。北大催生了中国最早的现代学制, 开创了中国最早的文科、理科、社科、农科、
第17行 医科等大学学科, 是近代以来中国高等教育的奠基者。""
第18行
第19行 stopWord=r", |。|、|的|了|在|是|有|和|就|不|人|都|上|也|很|到|说|要|去|会|着|看|好|这|那|于|\\n|\\r"
第20行 #中文信息处理中的停词, 此处仅为正则表达式示例
第21行
第22行 match=re.split(stopWord,strB) #用正则表达式切分str, 结果为list
第23行 print(match)
第24行
第25行 strC=""蒹葭苍苍, 白露为霜。所谓伊人, 在水一方。溯洄从之, 道阻且长。溯游从之, 宛在水中央。
第26行 蒹葭萋萋, 白露未晞。所谓伊人, 在水之湄。溯洄从之, 道阻且跻。溯游从之, 宛在水中坻。
第27行 蒹葭采采, 白露未已。所谓伊人, 在水之涘。溯洄从之, 道阻且右。溯游从之, 宛在水中沚。""
第28行
第29行 print(re.sub(r", |。|\\n",strC)) #re.sub()替换后仍为str
第30行
第31行 #eof

```

图3-1 例程3-1执行效果

对于re.search()或re.findall(), 如果匹配失败, 返回值为None; 如果匹配成功, 则分别是match对象或list对象。因此, 可以用返回值是否为None判断匹配成功与否, 如例程第6-7行、第10-11行所示。

例程3-1第22行re.split()用于字符串切分, 与str的split()功能相似但更加强大。在本例中, 用第19行所列出的标点符号、汉字以及转义字符(\\n和\\r)进行切分, 其结果为list。第29行的re.sub()用于替换, 和字符串的替换相比, re.sub()支持正则表达式替换, 即只要满足正则表达式条件就可以替换, 此处是较为简单运用。

对re.match()而言, 如果匹配成功, 将返回match对象, 其方法group()是匹配成功的内容, 如例程3-1第7行、例程3-2第9行所示。从例程3-2第9行还能看出, re.match().group(0)与re.match().group()功能相同, re.match().group(1)显示第1组匹配内容, re.match().group(2)显示第2组匹配内容。注意: 第1组和第2组来源于例程3-2第7行的"(\\d{3})-(\\d{8})", 其正则表达式的含义

是3个数字和8个数字分别构成两组。括号用于分组。如果正则表达式中有分组，则re.findall()匹配成功返回值虽然仍然是list，但每个list成员为tuple，如例程3-2第15行所示。

## 例程3-2

```
第1行 strTel=""公安部扫黄打非举报电话010-58186722
第2行 公安部经济犯罪举报中心010-66266833
第3行 公安部公安民警违法违纪举报电话010-58186696""
第4行
第5行 import re
第6行
第7行 match=re.search("(\\d{3})-(\\d{8})",strTel);
第8行 if match:
第9行     print(match.group(),match.group(0),match.group(1),match.group(2),sep="$")
第10行 #输出： 010-58186722$010-58186722$010$58186722
第11行
第12行 match=re.findall("(\\d{3})-(\\d{8})",strTel);
第13行 if match:
第14行     print(match)
第15行 #输出： [('010', '58186722'), ('010', '66266833'), ('010', '58186696')]
第16行
第17行 #eof
```

上述例程仅是正则表达式的简单应用，更多正则表示的规则，在第2节展开。

## 2、贪婪与懒惰

观察例程3-3第9-11行与第13-15行会发现第13行比第9行仅仅多了一个问号，但输出效果大不一样。在这里问号表示正则表达式采用**懒惰匹配**模式，即最短匹配。在各种语言的正则表达式中，一般默认**贪婪匹配**，即最长匹配。有些语言或命令，仅仅支持贪婪模式，不支持懒惰模式。在"<a.\*</a>"中句点代表单个字符(不包括换行符等)，\*代表0个或多个其前的字符，如\\d\*则0个或多个数字，.\*则代表0个或多个任意字符。"<a.\*</a>"总体含义就是以<a开始，以</a>结束，总计有多个字符。

## 例程3-3

```
第1行 strText=""<a href='http://www.pku.edu.cn'>北京大学</a>成立于1898年，
第2行 前身是京师大学堂。<a href='http://www.tsinghua.edu.cn'>清华大学</a>成立
第3行 于1911年，前身为清华学堂。""
第4行
第5行 import re
第6行 strText=strText.replace("\\n","")
第7行 print(strText)#输出删除空格后的字符串
第8行
第9行 allFinds=re.findall(r"<a.*</a>",strText)
第10行 print(allFinds)
第11行 #输出： ["<a href='http://www.pku.edu.cn'>北京大学</a>成立.....<a href='http://www.tsinghua.edu.cn'>清华大学</a>"]
第12行
第13行 allFinds=re.findall(r"<a.*?</a>",strText)
第14行 print(allFinds)
第15行 #输出： ["<a href='http://www.pku.edu.cn'>北京大学</a>", "<a href='http://www.tsinghua.edu.cn'>清华大学</a>"]
第16行
```

在例程中，第9行采用贪婪匹配，即以“<a”开始，以找到最远最长一个“</a>”为止，中间有任意多个字符。对于第13行，仍然是以“<a”开始，中间有多个字符，找到最近最短的“</a>”为止。因此，对于9行模式，匹配项仅有一个，而第13行模式，则有两个。贪婪匹配还是懒惰匹配，以?区分。

第二节 正则表达式

表3-1：正则表达式对象字符范围限制

字符集限制		
表达式	描述	示例
[abc]	限制为方括号内指定的字符，abc可以换成其他英文字符和数字。	例：用[123456789][0123456789]限制年龄输入，第1位必须是数字且不能取0。
[^abc]	限制为不是方括号内指定的字符。	例：[^0][0123456789]表示第一位不能为0，其后可以0-9之间的数字。注意：第一位还可以是其他字符如英文等，只是不能为0。
[0-9]	限制为0-9之间的数字。	例：用[1-9][0-9]限制年龄输入，第1位必须是数字且不能取0。
[a-z]	限制为a-z之间的英文字符，起点字符和终点字符可以调整，但其间为连续。	例：[a-c][a-z]*表示匹配a或b或c结尾或者abc之后有任意个a-z的字符。
[A-Z]	限制为A-Z之间的英文字符，起点字符和终点字符可以调整，但其间为连续。	例：用[A-H]表示A-H之间的所有大写英文字母
[A-z]	限制为大写 A 到小写 z 的字符，起点字符和终点字符可以调整，但其间为连续。	例：可以用[A-z0-9]表示所有英文字母和数字
元字符		
元字符	描述	示例
.	代表任意单个字符，除了换行符或行结束符。	例：[a-c].表示匹配含有a或b或c以及其后任意一个字符。
^	代表开始	例：^[a-c].表示开始为a或b或c，其后还有一个字符
\$	代表结束	例：^[a-c].\$表示以a或b或c开始，以任意字符结束，总长度为两个字符。
\w	代表单词字符，如英文字母等。	例：^[a-c]\w*\$表示以a/b/c开始的其后有任意个英文字母。
\W	代表非单词字符，如数字、\$、#等。	例：^.*\W.*\$表示以任意字符开始，中间有非英文字母，其后也还可以任意个字符
\d	代表数字，如0-9，与[0-9]含义相同。	例：^\d\d\$表示两位数字
\D	代表非数字字符，如各种字符、符号等	
\s	代表空白字符，如空格、换行符等等。	例：^.*\s.*\$代表含有空白字符的内容
\S	代表非空白字符，如字符、数字、符号等。	例：^\S*\$不能含有空白字符。
\b	代表单词边界，不匹配任何字符。 \b只是一个位置，一侧是构成单词的字符，另一侧为非单词字符、字符串的开始或结束位置。 \b是零宽度。	例：^\S.*\b代表任意非空白字符开始到单词边界。
\B	代表非单词边界。	
\n	代表换行符。	
数量限制		
元字符	描述	示例
n+	表示n所代表的字符至少有一个。	例：^.?*o+.*?\$表示以任意字符开始任意字符结束但至少

		含有一个o。
n*	表示n所代表的字符有零个或多个。	
n?	表示n所代表的字符有零个或一个。	
n{X}	表示n所代表的字符有X个。	例： <code>^.*?o{2}.*?\$</code> 表示以任意字符开始任意字符结束但至少含有两个o。
n{X,Y}	表示n所代表的字符有X或Y个。	例： <code>^.*?o{2,3}.*?\$</code> 表示以任意字符开始任意字符结束但至少含有两个或者三个o。
n{X,}	表示n所代表的字符至少有X个。	例： <code>^.*?o{2,}.*?\$</code> 表示以任意字符开始任意字符结束但至少含有两个o。
n\$	表示n所代表的字符其后为结尾。	
^n	表示n所代表的字符在开始。	例： <code>^a.*?o{2,3}.*?\$</code> 表示以a字符开始任意字符结束但至少含有两个或者三个o。

第三节 修饰符

在前述的代码中，`re.findall()`只有两个参数，如`re.findall("\d{5}",strA)`，第三个参数为正则表达式修饰符，如省略则按默认值执行。`re.findall("^a.*?$",strWord,re.I)`如果省略`re.I`则表示所有以小写字母开始的字符串，而如果加上则表示忽略大小写，即大写字母开始亦可。绝大多数Python正则表达式函数都支持如下表所示的修饰符。

表3-2：Python正则表达式修饰符

修饰符	描述	示例
re.I	亦作 <code>re.IGNORECASE</code> ，使匹配对大小写不敏感	
re.L	亦作 <code>re.LOCALE</code> ，做本地化识别(locale-aware)匹配	
re.M	亦作 <code>re.MULTILINE</code> ，多行匹配，影响 <code>^</code> 和 <code>\$</code>	对于多行文本，如没有该修饰符，则视之为一个字符串整体。
re.S	亦作 <code>re.DOTALL</code> ，使.匹配包括换行符在内的所有字符	英文句点默认不代表换行符，如果有该修饰符，则代表所有字符包括换行符。
re.A	亦作 <code>re.ASCII</code> ，使 <code>\w</code> 、 <code>\W</code> 、 <code>\b</code> 、 <code>\B</code> 、 <code>\d</code> 、 <code>\D</code> 、 <code>\s</code> 和 <code>\S</code> 执行仅与ASCII匹配而不是完全的Unicode匹配。默认按Unicode字符集解析字符。	
re.X	亦作 <code>re.VERBOSE</code> ，正则表达式中可以增加注释。	如例程3-4所示

正则表达式多个修饰符可以联合使用，每个修饰符之间用`|`连接，如`re.findall("^a.*?$",strWord,re.I|re.M)`表示支持多行且忽略大小写。其中`strWord`表示字符串名称

例程3-4

第1行

第2行

第3行

第4行

第5行

```
正则表达式a和b相同。
a = re.compile(r"""\d + # the integral part
                \. # the decimal point
                \d * # some fractional digits""", re.X)
b = re.compile(r"\d+\.\d*")
```

例程3-5

第1行

第2行

第3行

第4行

第5行

第6行

```
import re #引入re正则表达式库

strWord = ""acolyte
aconite
acorn
acoustic
```

```

第7行  bobby
第8行  bode
第9行  bomb
第10行 bookworm
第11行 boom
第12行 content
第13行 contest
第14行 cookie
第15行 coolest
第16行 """
第17行
第18行 print(re.sub("\n","",strWord)) #将换行符替换
第19行 #输出: acolyteaconiteacornacousticbobbybode.....
第20行
第21行 print(re.findall("oo",strWord))#输出: ['oo', 'oo', 'oo', 'oo']
第22行 print(re.findall(".*?oo.*",strWord))#输出: ['bookworm', 'boom', 'cookie', 'coolest']
第23行 print(re.findall("^.*?oo.*$",strWord))#输出: []相当于没有找到
第24行 print(re.findall("^.*?oo.*$",strWord,re.M))#输出: ['bookworm', 'boom', 'cookie', 'coolest']
第25行 print(re.findall("^([ab]).*?oo.*$",strWord,re.M))#输出: ['bookworm', 'boom']
第26行
第27行 #eof

```

## 第四节 表达式编译

观察例程3-6第7-11行以及第13-16行，会发现代码相似结果相同。表面看来，`re.compile()`似乎价值不大，但`re.compile()`执行效率更高，应用更加简单，尤其是当同一个正则表达式多次被应用时，效果更加明显。`re.compile()`执行后生成正则表达式对象，有一些列属性和方法。

例程3-6

```

第1行  strTel=""公安部扫黄打非举报电话010-58186722
第2行  公安部经济犯罪举报中心010-66266833
第3行  公安部公安民警违法违纪举报电话010-58186696""
第4行
第5行  import re
第6行
第7行  objRe=re.compile(r"(\d{3})-(\d{8})");
第8行  match=objRe.search(strTel)
第9行  if match:
第10行      print(match.group(),match.group(0),match.group(1),match.group(2),sep="$")
第11行  #输出: 010-58186722$010-58186722$010$58186722
第12行
第13行  match=re.search(r"(\d{3})-(\d{8})",strTel)
第14行  if match:
第15行      print(match.group(),match.group(0),match.group(1),match.group(2),sep="$")
第16行  #输出: 010-58186722$010-58186722$010$58186722
第17行
第18行  #eof

```

re.compile()函数同样支持修饰符，以及修饰符联合使用，如例程3-7第7行所示。注意第9行代码中的findall()其功能与re.findall()相似，都是查找全部符合条件的匹配，但少了正则表达式和flags选项，其正则表达式由编译前的正则表达式确定。

#### 例程3-7

```
第1行 import re #引入re正则表达式库
第2行
第3行 strWord=""Object-oriented programming (OOP) is a programming paradigm
第4行 based on the concept of "objects", which may contain data, in the form of fields,
第5行 often known as attributes; and code, in the form of procedures, often known as methods.
第6行 """
第7行 oRe=re.compile(r"\b(\w*)\b",re.M|re.I)
第8行
第9行 wordList=oRe.findall(strWord) #找到所有单词
第10行 print(wordList)
第11行 print(len(wordList))
第12行
第13行 print(oRe.pattern) #输出被编译的正则表达式
第14行 print(oRe.flags) #输出正则表达式使用的修饰符
第15行 print(oRe.groups) #输出分组信息
第16行 print(oRe.groupindex)
第17行
第18行 #eof
```

## 第五节 常用方法与属性

### 1、字符串切分：re.split()

字符串对象也提供了split()方法，但远没有正则表达式方式灵活，如例程3-8所示。

#### 例程3-8

```
第1行 strText="蒹葭苍苍，白露为霜。所谓伊人，在水一方。"
第2行
第3行 import re
第4行
第5行 afterSplit=re.split(r", |。",strText)
第6行 if afterSplit:
第7行     print(afterSplit)
第8行 #输出：['蒹葭苍苍', '白露为霜', '所谓伊人', '在水一方', '']
第9行
第10行 afterSplit=strText.replace("。",", ").split(", ")
第11行 print(afterSplit)
第12行 #输出：['蒹葭苍苍', '白露为霜', '所谓伊人', '在水一方', '']
第13行
第14行 #eof
```

### 2、全匹配：re.findall()

#### 例程3-9

```
第1行 strText=""<a href='http://www.pku.edu.cn'>北京大学</a>成立于1898年，前身是京师大学堂。
```



```

第2行 <a href='http://www.tsinghua.edu.cn'>清华大学</a>成立于1911年，前身为清华学堂。"""
第3行
第4行 import re
第5行
第6行 allFinds=re.findall(r"\<a href=[\']{0,1}(http://.*?)[\']{0,1}\>(.*?)\</a\>",strText)
第7行
第8行 print(allFinds)
第9行 #输出: [('http://www.pku.edu.cn', '北京大学'), ('http://www.tsinghua.edu.cn', '清华大学')]
第10行
第11行 #eof

```

### 3、迭代器查找：re.finditer()

re.finditer()与re.findall()相似，不过re.finditer()返回值为迭代器，可通过迭代器方式访问，如for-in循环等。

#### 例程3-10

```

第1行 strText="""<a href='http://www.pku.edu.cn'>北京大学</a>成立于1898年，前身是京师大学堂。
第2行 <a href='http://www.tsinghua.edu.cn'>清华大学</a>成立于1911年，前身为清华学堂。"""
第3行
第4行 import re
第5行 allFinds=re.finditer(r"\<a href=[\']{0,1}(http://.*?)[\']{0,1}\>(.*?)\</a\>",strText)
第6行
第7行 for i in allFinds:
第8行     print(i.group(0),i.group(1),i.group(2),sep="$")
第9行
第10行 #eof

```

### 4、字符串替换：re.sub()

#### 例程3-11

```

第1行 strText="""Python具有丰富和强大的库。它常被昵称为胶水语言，
第2行 能够把用其他语言制作的模块（尤其是C/C++）很轻松地联结在一起。
第3行 常见的一种应用情形是，使用Python快速生成程序的原型（有时甚至是程序的最终界面），
第4行 然后对其中有特别要求的部分，用更合适的语言改写，比如3D游戏中的图形渲染模块，
第5行 性能要求特别高，就可以用C/C++重写，而后封装为Python可以调用的扩展类库。
第6行 需要注意的是在您使用扩展类库时可能需要考虑平台问题，某些可能不提供跨平台的实现。"""
第7行
第8行 import re
第9行 afterSub=re.sub(r"和|的|就|可以|时|可能|不|为|有|是|把|对|在|\\n"," ",strText)
第10行
第11行 print(afterSub)
第12行
第13行 #eof

```

re.sub()还可以对其匹配项进行处理，如例程3-11所示。

#### 例程3-12

```

第1行 strText="""蒹葭苍苍，白露为霜。所谓伊人，在水一方，溯洄从之，道阻且长。溯游从之，宛在水中央。
第2行 蒹葭萋萋，白露未晞。所谓伊人，在水之湄。溯洄从之，道阻且跻。溯游从之，宛在水中坻。
第3行 蒹葭采采，白露未已。所谓伊人，在水之涘。溯洄从之，道阻且右。溯游从之，宛在水中沚。"""

```



```

第4行
第5行 import re
第6行 afterSub=re.sub(r"蒹葭|伊人|溯洄|宛在",lambda s:'<b>'+s.group(0)+'<b>',strText)
第7行
第8行 print(afterSub)
第9行
第10行 #eof

```

注：Python还提供了re.subn()，其功能与re.sub()相似，不过其返回值为tuple，第一个值是替换后的字符串，第二个值是被替换的数量。

## 第六节 匹配对象

匹配对象在Python文档中被称为Match Object。匹配对象总是有一个布尔值True。如果匹配失败，re.match()和re.search()将返回None，因此可以用if语句进行判断。匹配对象有多个属性和方法。

- 1、分组方法：re.group()
- 2、字典分组方法：re.groupdict()
- 3、匹配位置：re.span()

### 例程3-13

```

第1行 import re
第2行 strText="Noodle,feet,Zoo,Pool,peep,school,jeep,proof,broom,needle"
第3行
第4行 searchMatch=re.search(r"oo|ee",strText)
第5行 print(searchMatch)
第6行 print(searchMatch.span())
第7行 #输出：(1,3)
第8行 print(strText[1:3])
第9行 print(strText[searchMatch.start():searchMatch.end()])
第10行
第11行 print(searchMatch.pos)
第12行 print(searchMatch.endpos)
第13行 print(searchMatch.lastindex)
第14行 print(searchMatch.string)#输出：Noodle,feet,Zoo,Pool,peep,school,jeep,proof,broom,needle
第15行 print(searchMatch.re)#输出：re.compile('oo|ee')
第16行
第17行 #eofimport re
第18行 strText="Noodle,feet,Zoo,Pool,peep,school,jeep,proof,broom,needle"
第19行
第20行 searchMatch=re.search(r"oo|ee",strText)
第21行 print(searchMatch)
第22行 print(searchMatch.span())
第23行 #输出：(1,3)
第24行 print(strText[1:3])
第25行 print(strText[searchMatch.start():searchMatch.end()])
第26行

```

第27行	print(searchMatch.pos)
第28行	print(searchMatch.endpos)
第29行	print(searchMatch.lastindex)
第30行	print(searchMatch.string)#输出： Noodle,feet,Zoo,Pool,peep,school,jeep,proof,broom,needle
第31行	print(searchMatch.re)#输出： re.compile('oo ee')
第32行	
第33行	#eof

第七节 小结