第八章　正则表达式

本章导读

字符串在程序设计中很常用，而字符串匹配是其常见需求。正则表达式是用一种形式化语法描述字符匹配模式。虽匹配方式千变万化，而正则表达式常多能胜任。

正则表达式常用于涉及大量文本的各种应用程序之中，如MS Word、vi、emacs以及各种IDE，也是UNIX以及类UNIX命令工具不可缺少的组成部分，如sed、awk以及grep等。很多编程语言或原生或通过扩展库支持正则表达式，如：Python、JavaScript、Perl、C++、Java等。

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

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

例程8-1是Python中正则表达式的简单应用示例。正则表达式是Python的标准模块，无需安装即可使用，如例程第1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}",strTel)用于搜索"\d{5}"在字符串strTel中的**第1次出现**，返回值为re.Match类型的对象。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，re.search("\d{5}",strTel)相当于搜索strTel中匹配第1次连续出现的5个数字。

search()是re的函数，如果存在匹配，仅能匹配一次。如果匹配不成功，则返回类型为None，如果匹配成功返回类型为re.Match类型的对象，该对象有多个函数或属性，其中，group()是匹配结果，start()是匹配成功的起始编号，end()是匹配成功的结束编号，string是属性，表示正则表达式输入的字符串即被搜索的字符串。

例程8-1

```
第1行  import re #使用正则表达式的前提
第2行
第3行  #查找字符串中的电话
第4行  strTel="消费者维权12315物价投诉12358地税服务12366劳保咨询12333"
第5行  match=re.search(r"\d{5}",strTel) #返回值为re.Match类型对象
```

```
第6行 if match==None:
第7行     print("没有匹配!")
第8行 else:
第9行     print(match) #直接输出是对象的字符串表达形式
第10行    #输出内容：<re.Match object;span=(5,10),match='12315'>
第11行
第12行    print(match.group()) #输出：12315，整体匹配结果
第13行    print(match.string) #输出：原输入字符串strTel值，即re.search()第二个参数值
第14行    S,E=match.start(),match.end()
第15行    print(S,E) #输出：5 10
第16行    print(strTel[S:E]) #输出：12315
```

re.search()是单次匹配（如果有匹配），re.findall()则是多重匹配，如例程8-2所示。和例程8-1相比，第5行的re.search()被替换为re.findall()，其参数完全相同，但re.search()返回类型是re.Match，而re.findall()返回类型是list。如果没有匹配则返回值空list，否则是全部匹配的字符串，如例程第9行所示。

例程8-2

```
第1行 import re #使用正则表达式的前提
第2行
第3行 #查找字符串中的电话
第4行 strTel="消费者维权12315物价投诉12358地税服务12366劳保咨询12333"
第5行 match=re.findall(r"\d{5}",strTel) #返回值list类型，和re.search()返回类型不同。
第6行 if len(match)==0:
第7行     print("没有匹配!")
第8行 else:
第9行     print(match) #输出：['12315','12358','12366','12333']
第10行 #re.findall()如没有匹配返回空list，否则返回匹配list
```

例程8-3第7行strRE是正则表达式字符串，将用于re.search()和re.findall()。和前述例程相比，正则表达式中多了英文圆括号，其功能是分组，或者说是子表达式。观察例程第11行的输出能更清楚，其结果分为两个部分，第一对圆括号匹配表现为第一部分，第二对圆括号中的\d{8}表现为第二个部分。对于re.findall()如果有分组圆括号，其匹配结果为表现为list的tuple成员，每对圆括号表现tuple的成员。

re.search()将匹配结果放在re.Match类对象中，通过其group()函数可以获得匹配结果，其中group()和group(0)等效，是全部匹配结果，而group(1)和group(2)或者group(N)（N>=1）则是获得匹配分项，其分项数量与分组相关。

例程8-3

```
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 strRE="(\d{3})-(\d{8})"
第8行
第9行 match=re.findall(strRE,strTel);
第10行 print(match)
第11行 #输出：[('010','58186722'),('010','66266833'),('010','58186696')]
第12行
```

```
第13行 match=re.search(strRE,strTel);
第14行 print(match.group(),match.group(0),match.group(1),match.group(2),sep="__")
第15行 #输出：010-58186722__010-58186722__010__58186722
第16行
第17行 #eof
```

正则表达式不仅仅是\d{5}或者\d{3}、\d{8}等，还有很多种用法，例程8-4是一种变化，更多用法将随后讲解。在例程中，第4行代码中的\d{1,}表示\d即数字出现一次或多次，当匹配strTel字符串时，其结果如第8行所示。

例程8-4

```
第1行 import re
第2行 strTel="消费维权12315经济犯罪投诉01066266833物价投诉12358扫黄打非01058186722"
第3行
第4行 strRE=r"(\d{1,})"
第5行
第6行 match=re.findall(strRE,strTel);
第7行 print(match)
第8行 #输出：['12315','01066266833','12358','01058186722']
第9行
第10行 #eof
```

第二节　贪婪与懒惰

在正则表达式中，\d代表数字；英文句点.代表任意字符；{3}表示出现3次；{1,}表示可以多次但至少出现1次；英文星号*则表示出现0次或多次；英文问号?则表示出现0次或1次。

例程8-5用于匹配<h1>开始和</h1>结束及其之间的文本，正则表达式的书写如例程第4行所示，r"<h1>.*</h1>"的含义是以<h1>开始，紧随0个或多个任意字符并以</h1>结束。例程本意是获得<h1>唐代诗歌</h1>和<h1>宋词</h1>，但结果却如第6行所示。通过观察会发现，第6行结果也符合预期，也确实是<h1>开始，且也是<h1>结束。

如何解决这个问题呢？将r"<h1>.*</h1>"修改为r"<h1>.*?</h1>"即可，即在英文星号*后增加英文问号?，其含义是“尽可能少重复”，即由英文句点.代表的任意字符重复0次或多次（由英文星号代表）但尽可能少重复（由英文问号?确定）。**也可以理解为长匹配和短匹配。当没有问号时，采用符合规则的尽量长匹配即贪婪匹配；当有问号时，则采用符合规则的尽量短匹配即懒惰匹配。**

例程8-5

```
第1行 import re
第2行 strTest="<h1>唐代诗歌</h1><h2>唐诗格律</h2><h1>宋词</h1><h2>宋词格律</h2>"
第3行
第4行 strRE=r"<h1>.*</h1>"
第5行 match=re.findall(strRE,strTest)
第6行 print(match) #输出：["<h1 class='XYZ'>唐代诗歌</h1><h2>唐诗格律</h2><h1>宋词</h1>"]
第7行
第8行 #eof
```

在例程8-6中，采用第6行匹配模式，即懒惰模式，将能找到每一个链接。如果删除英文问号?则将不能匹配出两个链接（如图8-1），而是如图8-2所示，明显不符合预期。

例程8-6

```
第1行 import re
第2行 strTest="""<a href='http://www.pku.edu.cn'>北京大学</a>成立于1898年，
第3行 前身是京师大学堂。<a href='http://www.tsinghua.edu.cn'>清华大学</a>成立
```

```
第4行  于1911年，前身为清华学堂。""".replace("\n","")
第5行
第6行  strRE=r"<a.*?</a>"
第7行  match=re.findall(strRE,strTest)
第8行  print(match)
第9行
第10行 #eof
```

```
["<a href='http://www.pku.edu.cn'>北京大学</a>", "<a href='http://www.tsinghua.edu.cn'>清华大学</a>"]
```

图8-1　例程8-6执行结果

```
["<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。<a href='http://www.tsinghua.edu.cn'>清华大学</a>"]
```

图8-2　例程8-6第6行删除?后执行结果

第三节　正则表达式

表8-1：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：^.*?o{2}.*?$表示以任意字符开始任意字符结束但至少含有两个o。
n{X,Y}	表示n所代表的字符有X或Y个。	例：^.*?o{2,3}.*?$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：^.*?o{2,}.*?$表示以任意字符开始任意字符结束但至少含有两个o。
n$	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：^a.*?o{2,3}.*?$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。

第四节　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8-2：Python正则表达式修饰符

修饰符	描述	示例
re.I	亦作re.IGNORECASE，使匹配对大小写不敏感	
re.L	亦作re.LOCALE，做本地化识别(locale-aware)匹配	
re.M	亦作re.MULTILINE，多行匹配，影响 ^ 和 $	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re.DOTALL，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re.ASCII，使\ w、\ W、\ b、\ B、\ d、\ D、\ s和\ S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re.VERBOSE，正则表达式中可以增加注释。	如例程8-7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8-7

```
第1行  正则表达式a和b相同。
第2行  a = re.compile(r"""\d + # the integral part
第3行              \. # the decimal point
```

```
第4行                \d * # some fractional digits""", re.X)
第5行    b = re.compile(r"\d+\.\d*")
```

例程8-8

```
第1行    import re #引入re正则表达式库
第2行
第3行    strWord="""acolyte
第4行    aconite
第5行    acorn
第6行    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
```

第五节　表达式编译

观察例程8-9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.complie()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8-9

```
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8-10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8-10

```
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
```

第六节　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8-11所示。

例程8-11

```
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8-12

```
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8-13

```
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8-14

```
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各种模块（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8-14所示。

例程8-15

```
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。

第七节　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8-16

```
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
```

第八节　小结