爬虫07-闲谈正则表达式（二）

现在讨论第二类匹配规则，我称之为“概括字符集”，主要讨论以下六种：

匹配规则	释义	等价于
\d	表示该位置上的字符是数字，即匹配成功	[0-9]
\D	表示该位置上的字符不是数字，即匹配成功	[^0-9]
\w	表示该位置上的字符是字母或_，即匹配成功	[A-Za-z_]
\W	表示该位置上的字符不是是字母或_，即匹配成功	[^A-Za-z_]
\s	表示该位置上是不可见字符（空格、制表符\t、垂直制表符\v、回车符\r、换行符\n、换页符\f），即匹配成功	[\f\n\t\r\v]
\S	表示该位置上不是不可见字符，即匹配成功	[^\f\n\t\r\v]

接下来，我们要代码一一来解释，看看各自的效果是怎么样的，首先看\d：

```
import re
target = '点赞数：12'
result = re.findall('\d', target)
# 这一行中的\d表示只要该位置上的字符是数字，就匹配成功，返回结果，一次只表示一个字符
print(result)
# 得到的结果是['1', '2']
# 能看到，其实理想的是我们获取的结果是'12',别急，后面我们会讲到数量词，就能获取到。

```

再来看看\D：

```

import re
target = '点赞数：12'
result = re.findall('\D', target)
# 这一行中的\D表示只要该位置上的字符不是数字，就匹配成功，返回结果，一次只表示一个字符
print(result)
# 得到的结果是['点', '赞', '数', '：']
```

再来看看\w：

```python
import re
target = 'i love python_'
result = re.findall('\w', target)
# 这一行中的\w表示只要该位置上的字符是字母或者下划线，就匹配成功，返回结果，一次只表示一个字符
print(result)
# 得到的结果是['i', 'l', 'o', 'v', 'e', 'p', 'y', 't', 'h', 'o', 'n', '_']
```

再来看看\W：

```python
import re
target = 'i love python_'
result = re.findall('\W', target)
# 这一行中的\W表示只要该位置上的字符不是字母或者下划线，就匹配成功，返回结果，一次只表示一个字符
print(result)
# 得到的结果是[' ', ' ']

```

再来看看\s：

```python
import re
target = 'life is short \n i love python'
result = re.findall('\s', target)
# 这一行中的\s表示只要该位置上的字符是不可见字符，就匹配成功，返回结果，一次只表示一个字符
print(result)
# 得到的结果是[' ', ' ', ' ', '\n', ' ', ' ', ' ']

```

再来看看\S：

```python
import re
target = 'life is short \n i love python'
result = re.findall('\S', target)
# 这一行中的\S表示只要该位置上的字符不是不可见字符，就匹配成功，返回结果，一次只表示一个字符
print(result)
# 得到的结果是['l', 'i', 'f', 'e', 'i', 's', 's', 'h', 'o', 'r', 't', 'i', 'l', 'o', 'v', 'e', 'p', 'y', 't', 'h', 'o', 'n']

```

通过上述六个简单的例子，就能发现，单个地使用概括字符集意义并不大，无法达到我们想要的效果，往往需要结合数量词来使用，才会发挥出最大的效用。

接下来，我们来看看第三类匹配规则——数量词，并结合第二类概括字符集来实现某些功能：

匹配规则（举例说明）	释义
{3}	表示{3}前面的一个字符出现3次
{3,8}	表示{3}前面的一个字符出现3-8次
?	表示?前面的一个字符出现0次或1次
+	表示+前面的一个字符出现1次或无限多次
*	表示*前面的一个字符出现0次或无限多次

假设现在我们获取到了一本英文书的全部内容，想要判断这本书有多少字数，该怎么做呢？

还是老规矩，我们通过代码来解释：

```
import re
content = 'To be or not to be,that is a question'
result = re.findall('\w{1,30}', content)
# 这一行中的\w表示一个字母或者_，{1,30}表示\w出现1次到30次之间，只要一个单词的长度在1-30之间就能被匹配出来
print(result)
# 得到的结果是['To', 'be', 'or', 'not', 'to', 'be', 'that', 'is', 'a', 'question']
print(len(result))
# 得到的结果是10，表明这本书有10个字
```

上述代码还是存在某些缺陷，万一某个英文单词的长度超过30了怎么办？所以我们应该换种方式来写：

```
import re
content = 'To be or not to be,that is a question'
result = re.findall('\w+', content)
# 这一行中的\w表示一个字母或者_，+表示\w出现1次或无限次数，只要一个单词的长度大于1就能被匹配出来
print(result)
# 得到的结果是['To', 'be', 'or', 'not', 'to', 'be', 'that', 'is', 'a', 'question']
```

```
print(len(result))
# 得到的结果是10，表明这本书有10个字

```

再来看一个很实用的例子，提取文章的点赞数或者评论数等等：

```
import re
content = '点赞数：12'
result = re.findall('\d{1,10}', content)
# 这一行中的\d表示一个数字字符，{1,10}表示这个\d出现1-10次都匹配成功，只要点赞数在0-9999999999之间都可以匹配出来。
# 同样可以这样写：
# result = re.findall('\d+', content)
print(result)
# 得到的结果是'12'
```

至此，数量词我们已经讨论了{}和+，至于？和*，我们放在后面和贪婪与非贪婪一起讲，因为?和*还有.经常连在一起使用。

今天的最后，我们再来讨论第四类匹配规则，**边界匹配符**：

匹配规则	释义
^	表示只要是以^后面的字符开头的，即匹配成功
$	表示只要是以$前面的字符结尾的，即匹配成功

同样的，我们用代码来解释，更直观一些：

```
import re
content = 'https://www.zhihu.com'
content1 = '/question/62749917/answer/576934857'
result = re.findall('^http.*', content)
```

这一行的^http表示匹配content的首部是http的内容，后面的.表示一个除换行符\n以外的所有字符，*表示.重复0次或无限多次，.*放在一起就是匹配除换行符以外的任意字符无限多次，这两个字符经常放在一起用，之后会单独讨论。

```
result1 = re.findall('^http.*', content1)
print(result)
# 得到的结果是['https://www.zhihu.com']
print(result1)
# 得到的结果是[]，因为content1并不是以http开头
```

通过上述例子，我们能发现^其实就相当于我们之间讲的字符串的一个方法，叫做startswith。

同样的，有startswith，当然会有类似于endswith的匹配规则，我们用代码来看下：

```
import re
content = 'https://www.zhihu.com/shiyue.png'
content1 = 'https://www.zhihu.com'
result = re.findall('.*png$', content)
# 这一行的.*和之前一样，表示png前面可以有除换行符之外的任意字符，png$表示匹配content以png结尾的内容。
result1 = re.findall('.*png$', content1)
print(result)
# 得到的结果是['https://www.zhihu.com/shiyue.png']
print(result1)
 # 得到的结果是[]，因为content1不是以png结尾。
```

一百零七 阅