10 Python中的正则表达式用法

更新时间：2019-06-14 14:34:42

> 理想的书籍是智慧的钥匙。
>
> ——列夫·托尔斯泰

本节将分为两个部分介绍如何在 python 中使用正则表达式，第一部分将会从`re`这个标准模块入手，详细介绍其用法与重要的函数；第二部分，通过将正则表达式在爬虫中运用常见实例来让你更深刻地理解正则表达式给爬虫程序带来的便利性。

`re`模块

Python自1.5版本起增加了`re`模块，它提供Perl风格的正则表达式模式。
`re`模块使Python语言拥有全部的正则表达式功能。`compile`函数根据一个模式字符串和可选的标志参数生成一个正则表达式对象。该对象拥有一系列方法用于正则表达式匹配和替换。`re`模块也提供了与这些方法功能完全一致的函数，这些函数使用一个模式字符串作为它们的第一个参数。

正则表达式修饰符——可选标志

正则表达式可以包含一些可选标志修饰符来控制匹配的模式。修饰符被指定为一个可选的标志。多个标志可以通过按位`OR(|)`来指定，如`re.I | re.M`被设置成`I`和`M`标志：

修饰符	描述
`re.I`	使匹配对大小写不敏感
`re.L`	做本地化识别（locale-aware）匹配
`re.M`	多行匹配，影响`^`和`$`
`re.S`	使`.`匹配包括换行在内的所有字符
`re.U`	根据Unicode字符集解析字符。这个标志影响`\w`、`\W`、`\b`、`\B`
`re.X`	该标志通过更灵活的格式以便正则表达式可以写得更易于理解

`re.match`函数

`re.match`尝试从字符串的起始位置匹配一个模式，如果起始位置匹配不成功的话，则`match()`返回`None`。

函数语法：

```
re.match(pattern, string, flags=0)
```

函数参数说明如下表所示。

参数	描述
`pattern`	匹配的正则表达式
`string`	要匹配的字符串
`flags`	标志位，用于控制正则表达式的匹配方式，如是否区分大小写、多行匹配等

匹配成功则`re.match`方法返回一个匹配的对象，否则返回`None`。

示例：

```
>>> import re
>>> print(re.match('www', 'www.example.com').span())  # 从起始位置匹配(可以执行成功)
>>> print(re.match('com', 'www.example.com'))         # 不在起始位置匹配(将执行失败)
(0, 3)
None
```

我们可以使用`group(num)`或`groups()`匹配对象函数来获取匹配表达式。

匹配对象方法	描述
`group(num=0)`	匹配的整个表达式的字符串，`group()`可以一次输入多个组号，在这种情况下它将返回一个包含那些组所对应值的元组
`groups()`	返回一个包含所有小组字符串的元组，从`1`到所含的小组号

示例：

```
#!/usr/bin/python
import re

line = "Cats are smarter than dogs"

matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)

if matchObj:
   print("matchObj.group() : ", matchObj.group())
   print("matchObj.group(1) : ", matchObj.group(1))
   print("matchObj.group(2) : ", matchObj.group(2))
else:
   print("No match!!")
```

结果输出如下：

```
matchObj.group() :  Cats are smarter than dogs
matchObj.group(1) :  Cats
matchObj.group(2) :  smarter
```

`re.search`方法

`re.search`扫描整个字符串并返回第一个成功的匹配。

函数语法：

```
re.search(pattern, string, flags=0)
```

函数参数说明如下表所示。

参数	描述
`pattern`	匹配的正则表达式
`string`	要匹配的字符串
`flags`	标志位，用于控制正则表达式的匹配方式，如是否区分大小写、多行匹配等

匹配成功则`re.search`方法返回一个匹配的对象，否则返回`None`。

示例：

```
>>> import re
>>> print(re.search('www', 'www.example.com').span())  # 在起始位置匹配
>>> print(re.search('com', 'www.example.com').span())  # 不在起始位置匹配
(0,3)
(12,15)
```

我们可以使用`group(num)`或`groups()`匹配对象函数来获取匹配表达式。

匹配对象方法	描述
`group(num=0)`	匹配的整个表达式的字符串，`group()`可以一次输入多个组号，在这种情况下它将返回一个包含那些组所对应值的元组
`groups()`	返回一个包含所有小组字符串的元组，从`1`到所含的小组号

示例：

```
#!/usr/bin/python
import re

line = "Cats are smarter than dogs"

searchObj = re.search(r'(.*) are (.*?) .*', line, re.M | re.I)

if searchObj:
    print("searchObj.group() : ", searchObj.group())
    print("searchObj.group(1) : ", searchObj.group(1))
    print("searchObj.group(2) : ", searchObj.group(2))
else:
    print("Nothing found!!")
```

以上实例执行结果如下：

```
searchObj.group() :  Cats are smarter than dogs
searchObj.group(1) :  Cats
searchObj.group(2) :  smarter
```

`re.match`与`re.search`的区别

`re.match`只匹配字符串的开始，如果字符串开始不符合正则表达式，则匹配失败，函数返回`None`。
而`re.search`匹配整个字符串，直到找到一个匹配。

示例：

```
#!/usr/bin/python
import re

line = "Cats are smarter than dogs"

matchObj = re.match( r'dogs', line, re.M|re.I)
if matchObj:
   print("match --> matchObj.group() : ", matchObj.group())
else:
   print ("No match!!")

matchObj = re.search( r'dogs', line, re.M|re.I)
if matchObj:
   print ("search --> matchObj.group() : ", matchObj.group())
else:
   print ("No match!!")
```

以上实例运行结果如下：

```
No match!!
search --> matchObj.group() :  dogs
```

检索和替换

Python的`re`模块提供了`re.sub`方法，作用是对于输入的一个字符串，利用正则表达式，去实现字符串替换处理，然后返回被替换后的字符串

语法：

```
re.sub(pattern, repl, string, count=0, flags=0)
```

参数说明

参数	说明
`pattern`	正则中的模式字符串
`repl`	替换的字符串，也可为一个函数
`string`	要被查找替换的原始字符串
`count`	模式匹配后替换的最大次数，默认为0，表示替换所有的匹配

实例

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import re

phone = "2004-959-559 # 这是一个国外电话号码"

# 删除字符串中的Python注释
num = re.sub(r'#.*$', "", phone)
print("电话号码是: ", num)

# 删除非数字（-）的字符串
num = re.sub(r'\D', "", phone)
print("电话号码是 : ", num)
```

以上实例执行结果如下：

```
电话号码是 :  2004-959-559
电话号码是 :  2004959559
```

替换

以下实例将字符串中的匹配的数字乘以`2`：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import re

# 将匹配的数字乘以2
def double(matched):
    value = int(matched.group('value'))
    return str(value * 2)

s = 'A23G4HFD567'
print(re.sub('(?P<value>\d+)', double, s))
```

执行输出结果为：

```
A46G8HFD1134
```

爬虫用于分析响应数据常见方法

抓取 title 标签间的内容

延续本章第一节所创建的网易新闻爬虫的示例， 爬取网页的标题，采用正则表达式"直接匹配返回正文的标题，代码如下：

> 这个 title 是不是有问题？

```
# -*- coding: utf-8 -*-
import re
import scrapy
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule

class NeteaseSpider(CrawlSpider):
    name = 'netease'
    allowed_domains = ['163.com']
    start_urls = ['https://www.163.com/']

    # 提取以http://sports.163.com/ 为开始的所有链接
    rules = (
        Rule(LinkExtractor(allow=r'^https://sports.163.com/'), callback='parse_item', follow=True),
    )

    def parse_item(self, response):
    content = response.body
      title = re.findall(r'<title>(.*?)</title>', content)
  return {'title':title}
```

如果我们想得到更好的性能，那么就可以将正则表达式先进行预编译并放置于`parse_item`方法之外，那`parse_item`函数的代码则为：

```
# 将正则表达式进行预编译
pattern = r'(?<=<title>).*?(?=</title>)'
ex = re.compile(pattern, re.M|re.S)

def parse_item(self, response):
   content = response.body
   obj = re.search(ex,content) # 采用编译好的表达式对象对内容进行搜索
   title = obj.group()
 return {'title':title}
```

为什么`LinkExtractor`可以提取链接

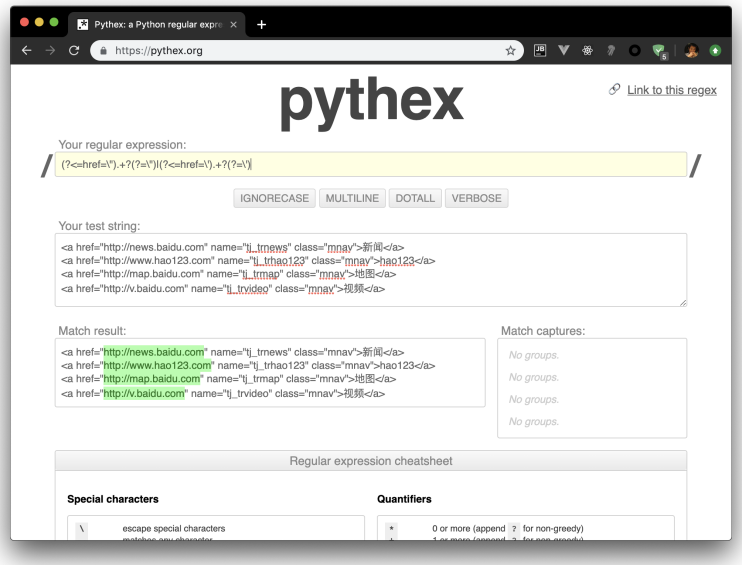
此示例是从网页内容中匹配所有链接URL，从这个示例可以从本质上来解释为什么`LinkExtractor`可以提取链接。

假设网页源码中有以下的内容：

```
<a href="http://news.baidu.com" name="tj_trnews" class="mnav">新闻</a>
<a href="http://www.hao123.com" name="tj_trhao123" class="mnav">hao123</a>
<a href="http://map.baidu.com" name="tj_trmap" class="mnav">地图</a>
<a href="http://v.baidu.com" name="tj_trvideo" class="mnav">视频</a>
```

首先，我们需要设计提取 URL 的正则表达式：

```
(?<=href=\").+?(?=\")|(?<=href=\').+?(?=\')
```



我们可以猜出`LinkExtractor`的代码应该实现如下的代码逻辑：

```
def _exec_links(content,allow):
  pattern = r"(?<=href=\").+?(?=\")|(?<=href=\').+?(?=\')"
  ex = re.compile(pattern, re.M|re.S)
  urls = re.findall(res, content, re.I|re.S|re.M)
  for url in urls:
     # 这就是 allow 的秘密
     if re.match(allow,url):
      yield url
```

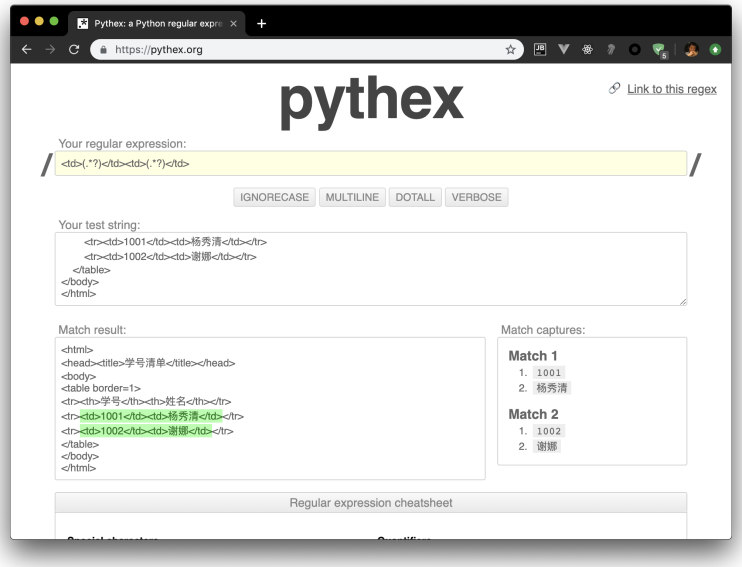
虽然你去查看`LinkExtractor`的源代码一定与上述代码有所差异，但其原理是一致的，就是先以正则表达式从页面内提取出URL，然后再逐一地按你所定义的匹配模式一一筛选符合匹配规则的链接。

提取表格`<td>`的文字内容

这是一个很实用的例子：我们从网页返回一个学生名单且通过正则表达多获取其中的学生信息，该学生名单的网页内容如下所示：

```
<html>
<head><title>学号名单</title></head>
<body>
    <table border=1>
        <tr><th>学号</th><th>姓名</th></tr>
        <tr><td>1001</td><td>杨秀清</td></tr>
        <tr><td>1002</td><td>谢娜</td></tr>
    </table>
</body>
</html>
```

先定义表达式，然后测试其输出结果：



那么爬虫内的`parse_item`代码就可以写成如下的方式

```
#直接获取<td></td>间内容
pattern = r'<td>(.*?)</td><td>(.*?)</td>'
ex = re.compile(pattern, re.M|re.S)

def parse_item(self, response):
   content = response.body
   obj = re.search(ex,content) # 采用编译好的表达式对象对内容进行搜索
    texts = re.findall(res, content, re.S|re.M)
    for m in texts:
       yield {'number': m[0], 'name': m[1]}
```

提取图片地址的简单方法

HTML 内图片使用标签格式为`<img src=图片地址 />`，在某些情况下可能我们需要一次性将网页内正则需要获取图片 URL 链接地址的方法如下：

```
content = '''<img alt="Python" src="http://www.csdn.net/eastmount.jpg" />'''
urls = re.findall('src="(.*?)"', content, re.I|re.S|re.M)
```

小结

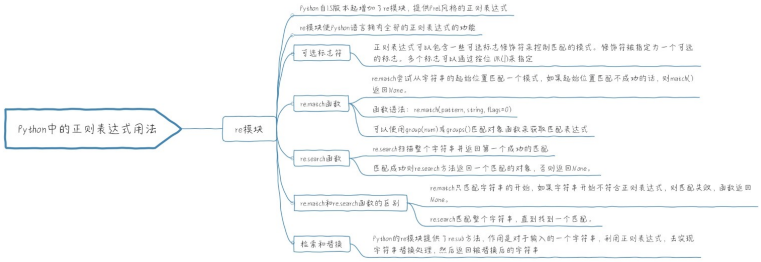
这节课的主要目的是带着你认识一下 Python 中的`re`模块，你要仔细的区分`match`和`search`方法的区别：

- `re.match`尝试从字符串的起始位置匹配一个模式，如果起始位置匹配不成功的话，则`match()`返回`None`。

- `re.search`扫描整个字符串并返回第一个成功的匹配。

同时`re`模块中也提供了`sub`方法来进行字符的替换，语法为`re.sub(pattern, repl, string, count)`。

对于本节课所讲的内容同学们要多加练习，这样才能在爬虫开发中去更好地使用`re`模块，这节课就讲到这里，下节课我们一起来让网易新闻爬虫爬起来。



09 学习正则表达式—构造复杂模式

11 编写网易爬虫NeteaseSpider 让它“爬”起来