Python玩转正则表达式，看完这篇你就会了

忆想不到的晖 LV.4
2021年04月25日 23:11 · 阅读 1140

关注

什么是正则表达式?

> 正则表达式是一个特殊的字符序列，它能帮助你方便的检查一个字符串是否与某种模式匹配。例如在编写处理字符串的程序或网页时，经常有查找符合某些复杂规则的字符串的需要。正则表达式就是用于描述这些规则的工具。换句话说，正则表达式就是记录文本规则的代码。

单字符匹配

字符	功能
.	匹配任意1个字符（除了\n）
[ ]	匹配[ ]中列举的字符

字符	功能
\d	匹配数字，即0-9
\D	匹配非数字，即不是数字
\s	匹配空白，即 空格，tab键
\S	匹配非空白
\w	匹配单词字符，即a-z、A-Z、0-9、_
\W	匹配非单词字符

多字符匹配

字符	功能
*	匹配前一个字符出现0次或者无限次，即可有可无
+	匹配前一个字符出现1次或者无限次，即至少有1次
?	匹配前一个字符出现1次或者0次，即要么有1次，要么没有
{m}	匹配前一个字符出现m次
{m,n}	匹配前一个字符出现从m到n次

匹配开头和结尾

字符	功能
^	匹配字符串开头
$	匹配字符串结尾

匹配分组

字符	功能

字符	功能
|	匹配左右任意一个表达式
(ab)	将括号中字符作为一个分组
\num	引用分组num匹配到的字符串
(?P<name>)	分组起别名
(?P=name)	引用别名为name分组匹配到的字符串

常用正则表达式

匹配内容	正则表达式
中文字符	[\u4e00-\u9fa5]
双字节字符	[^\x00-\xff]
空白行	\s
Email地址	\w[-\w.+]*@([A-Za-z0-9][-A-Za-z0-9]+\.)+[A-Za-z]{2,14}
网址URL	^((https|http|ftp|rtsp|mms)?:\/\/)[^\s]+
手机号码（国内）	0?(13|14|15|17|18)[0-9]{9}
电话号码（国内）	[0-9-() （）]{7,18}
负浮点数	-([1-9]\d*.\d*|0.\d*[1-9]\d*)
匹配整数	-?[1-9]\d*
正浮点数	[1-9]\d*.\d*|0.d*[1-9]\d*
腾讯QQ号	[1-9]([0-9]{5,11})
邮政编码	\d{6}
身份证号码	\d{17}[\d|x]|\d{15}
格式日期	\d{4}(\-|\/|.)\d{1,2}\1\d{1,2}
正整数	[1-9]\d*

匹配内容	正则表达式
负整数	-[1-9]\d*
用户名	[A-Za-z0-9_\-\u4e00-\u9fa5]+

IP地址

正则表达式

python 复制代码

```
(25[0-5]|2[0-4]\d|[0-1]\d{2}|[1-9]?\d)\.(25[0-5]|2[0-4]\d|[0-1]\d{2}|[1-9]?\d)\.(25[0-5]|2[0
```

Python re模块

> 在 Python 中需要通过正则表达式对字符串进行匹配的时候，可以使用一个模块，名字为 re 。
>
> re 是 regular expression 的缩写，表示正则表达式。
>
> re 模块使 Python 语言拥有全部的正则表达式功能。

re.match函数

re.match 尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话， match() 就返回 None 。

函数语法

python 复制代码

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

函数参数说明

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，多行匹配等等。

匹配成功 re.match 方法返回一个匹配（ Match ）的对象，否则返回 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，默认0。
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。
span()	返回匹配成功的字符的开始位置和结束位置，结果为元组 (start, end)

测试范例

范例 1:

python 复制代码

```python
# -*- coding:utf-8 -*-
import re

print(re.match('www', 'www.csdn.net'))  # 在起始位置匹配
print(re.match('net', 'www.csdn.net'))  # 不在起始位置匹配
```

运行输出结果为:

python 复制代码

```
<re.Match object; span=(0, 3), match='www'>
None
```

`span=(0, 3)` 则表示匹配成功的开始位置和结束位置。

范例 2：

提取文章的主要数据

已赞75，评论12，收藏231

python 复制代码

```
# -*- coding:utf-8 -*-
import re

line = u"已赞75，评论12，收藏231"

match_obj = re.match( r'已赞(\d*).评论(\d*).收藏(\d*)', line, re.M|re.I)

if match_obj:
   print ("match_obj.group() : ", match_obj.group())
   print ("match_obj.group(1) : ", match_obj.group(1))
   print ("match_obj.group(2) : ", match_obj.group(2))
   print ("match_obj.group(3) : ", match_obj.group(3))
else:
   print ("No match!!")
```

运行输出结果如下：

python 复制代码

```
match_obj.group() :  已赞75，评论12，收藏231
match_obj.group(1) :  75
match_obj.group(2) :  12
match_obj.group(3) :  231
```

- `re.I` 使匹配对大小写不敏感
- `re.M` 多行匹配，影响 ^ 和 $

更多的标志文章后续会提到。

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

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

函数参数说明

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，多行匹配等等。

跟 match 一样匹配成功 re.search 方法返回一个匹配（Match）的对象，否则返回 None 。

测试范例

范例 1:

```python
# -*- coding:utf-8 -*-
import re

print(re.search('www', 'www.csdn.net'))         # 在起始位置匹配
print(re.search('net', 'www.csdn.net'))         # 不在起始位置匹配
```

以上实例运行输出结果为:

```python
<re.Match object; span=(0, 3), match='www'>
<re.Match object; span=(9, 12), match='net'>
```

范例 2:

提取博客的信息

作者Clever_Hui，阅读量4155，收藏231，分类专栏python

python 复制代码

```python
# -*- coding:utf-8 -*-
import re

line = u"作者Clever_Hui，阅读量4155，收藏231，分类专栏python"

search_obj = re.search( r'作者(\w*).阅读量(\d*).收藏(\d*).分类专栏(\w*)', line, re.M|re.I)

if search_obj:
   print ("search_obj.group() : ", search_obj.group())
   print ("search_obj.group(1) : ", search_obj.group(1))
   print ("search_obj.group(2) : ", search_obj.group(2))
   print ("search_obj.group(3) : ", search_obj.group(3))
   print ("search_obj.group(4) : ", search_obj.group(4))
else:
   print ("Nothing found!!!")
```

以上实例执行结果如下：

python 复制代码

```
search_obj.group() :  作者Clever_Hui，阅读量4155，收藏231，分类专栏python
search_obj.group(1) :  Clever_Hui
search_obj.group(2) :  4155
search_obj.group(3) :  231
search_obj.group(4) :  python
```

re.match与re.search的区别

`re.match` 尝试从字符串的起始位置匹配一个模式，只匹配字符串的开始，如果不是起始位置匹配成功的话， `match()` 就返回 `None` 。

`re.search` 扫描整个字符串并返回第一个成功的匹配，如果没有则返回 `None` 。

测试范例

python 复制代码

```python
# -*- coding:utf-8 -*-
import re

match_obj = re.match('net', 'www.csdn.net')
```

```
rearch_obj = re.search('net', 'www.csdn.net')

if match_obj:
    print('match    -->  ', match_oj)
else:
    print('No match!!!')

if rearch_obj:
     print('search  -->  ', rearch_obj)
else:
    print('No Match!!!')
```

运行输出结果如下：

python 复制代码

```
No match!!!
search  -->   <re.Match object; span=(9, 12), match='net'>
```

re.findall函数

函数语法

Python 复制代码

```
findall(pattern, string, flags=0)
```

函数参数说明

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，多行匹配等等。

官方文档

python 复制代码

```
findall(pattern, string, flags=0)
    Return a list of all non-overlapping matches in the string.

    If one or more capturing groups are present in the pattern, return
    a list of groups; this will be a list of tuples if the pattern
    has more than one group.

    Empty matches are included in the result.
```

结果返回 `string` 中所有与 `pattern` 相匹配的全部字串，返回形式为列表

如果 `pattern` 中有一个或多个捕获组，则返回组的列表，

如果 `pattern` 中有多个组，这将是一个元组列表

结果中包含空匹配项。

测试范例

统计出 `python` 、 `c` 、 `java` 相应文章阅读的次数，**python = 9999+, c = 7890, java = 12345**

python 复制代码

```
# -*- coding:utf-8 -*-
import re

line = "python = 9999, c = 7890, java = 12345"
ret1 = re.findall(r"\d+", line)
ret2 = re.findall(r"(\w+)\s.\s(\d+)", line)

print(ret1)
print(ret2)
```

运行结果：

python 复制代码

```
['9999', '7890', '12345']
[('python', '9999'), ('c', '7890'), ('java', '12345')]
```

re.sub函数

Python 的re模块提供了re.sub用于替换字符串中的匹配项。

函数语法

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

官方文档

```python
sub(pattern, repl, string, count=0, flags=0)
    Return the string obtained by replacing the leftmost
    non-overlapping occurrences of the pattern in string by the
    replacement repl.  repl can be either a string or a callable;
    if a string, backslash escapes in it are processed.  If it is
    a callable, it's passed the Match object and must return
    a replacement string to be used.
```

函数参数说明

参数	描述
pattern	匹配的正则表达式
repl	替换的字符串或一个函数
string	要匹配的字符串
count	模式匹配后替换的最大次数
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，多行匹配等等

返回的字符串是在字符串中用 re 最左边不重复的匹配来替换。如果模式没有发现，字符将被没有改变地返回。

可选参数 count 是模式匹配后替换的最大次数；count 必须是非负整数。缺省值是 0 表示替换所有的匹配。

测试案例

需求：将匹配到的阅读次数加1

方法1：

python 复制代码

```python
# -*- coding:utf-8 -*-
import re

ret = re.sub(r"\d+", '998', "python = 997")
print(ret)
```

运行结果：

python 复制代码

```
python = 998
```

方法2：

python 复制代码

```python
# -*- coding:utf-8 -*-
import re

def add(temp):
    strNum = temp.group()
    num = int(strNum) + 1
    return str(num)

ret = re.sub(r"\d+", add, "python = 997")
print(ret)

ret = re.sub(r"\d+", add, "python = 99")
print(ret)
```

运行结果：

python 复制代码

```
python = 998
python = 100
```

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(|) 它们来指定。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	该标志通过给予你更灵活的格式以便你将正则表达式写得更易于理解。

贪婪和非贪婪

Python 里数量词默认是贪婪的（在少数语言里也可能是默认非贪婪），总是尝试匹配尽可能多的字符；

非贪婪则相反，总是尝试匹配尽可能少的字符。

在 * 、 ? 、 + 、 {m,n} 后面加上 ? ，使贪婪变成非贪婪。

python 复制代码

```python
# -*- coding:utf-8 -*-
import re

s="This is a number 234-235-22-423"
r=re.match(".+(\d+-\d+-\d+-\d+)",s)
print(r.group(1))
# '4-235-22-423'

r=re.match(".+?(\d+-\d+-\d+-\d+)",s)
print(r.group(1))
# '234-235-22-423'
```

正则表达式模式中使用到通配字，那它在从左到右的顺序求值时，会尽量 抓取 满足匹配最长字符串，在我们上面的例子里面， .+ 会从字符串的起始处抓取满足模式的最长字符，其中包括我们想得

到的第一个整型字段的中的大部分， `\d+` 只需一位字符就可以匹配，所以它匹配了数字 `4` ，而 `.+` 则匹配了从字符串起始到这个第一位数字4之前的所有字符。

解决方式：非贪婪操作符 `?` ，这个操作符可以用在 `*` 、 `?` 、 `+` 、 `{m,n}` 后面，要求正则匹配的越少越好。

python 复制代码

```
>>> re.match(r"aa(\d+)","aa2343ddd").group(1)
'2343'
>>> re.match(r"aa(\d+?)","aa2343ddd").group(1)
'2'
>>> re.match(r"aa(\d+)ddd","aa2343ddd").group(1)
'2343'
>>> re.match(r"aa(\d+?)ddd","aa2343ddd").group(1)
'2343'
>>>
```

r原生字符串的作用

python 复制代码

```
>>> mm = "c:\\a\\b\\c"
>>> mm
'c:\\a\\b\\c'
>>> print(mm)
c:\a\b\c

>>> re.match("c:\\\\",mm).group()
'c:\\'

>>> ret = re.match("c:\\\\",mm).group()
>>> print(ret)
c:\

>>> ret = re.match("c:\\\\a",mm).group()
>>> print(ret)
c:\a

>>> ret = re.match(r"c:\\a",mm).group()
>>> print(ret)
c:\a

>>> ret = re.match(r"c:\a",mm).group()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
AttributeError: 'NoneType' object has no attribute 'group'
>>>
```

说明

Python中字符串前面加上 r 表示原生字符串，

与大多数编程语言相同， 正则表达式里使用"\"作为转义字符 ，这就可能造成反斜杠困扰。假如你需要匹配文本中的字符 \ ，那么使用编程语言表示的正则表达式里将需要4个反斜杠 \ ：**前两个和后两个分别用于在编程语言里转义成反斜杠，转换成两个反斜杠后再在正则表达式里转义成一个反斜杠**。

Python里的原生字符串很好地解决了这个问题，有了原生字符串，你再也不用担心是不是漏写了反斜杠，写出来的表达式也更直观。

python 复制代码

```
>>> ret = re.match(r"c:\\a",mm).group()
>>> print(ret)
c:\a
```

综合案例

匹配变量名是否有效

python 复制代码

```
# -*- coding:utf-8 -*-
import re

names = ["name1", "_name", "2_name", "__name__"]

for name in names:
    ret = re.match("[a-zA-Z_]+[\w]*",name)
    if ret:
        print("变量名 %s 符合要求" % ret.group())
    else:
        print("变量名 %s 非法" % name)
```

运行结果

python 复制代码

```
变量名 name1 符合要求
变量名 _name 符合要求
```

```
变量名 2_name 非法
变量名 __name__ 符合要求
```

匹配出，8到15位的密码

密码组合可以是大小写英文字母、数字，但开头必须是大写字母

python 复制代码

```python
# -*- coding:utf-8 -*-
import re

pwds = ["W123456W", "wwj123456", "W123456789", "w123w", "W12345678abcdefg"]
pwd_pattern_str = "^[A-Z][a-zA-Z0-9]{7,14}$"

for pwd in pwds:
    ret = re.match(pwd_pattern_str, pwd)
    if ret:
        pwd = ret.group()
        print("密码 %s 符合要求,  长度: %s" % (pwd, len(pwd)))
    else:
        print("密码 %s 非法" % pwd)
```

运行结果如下：

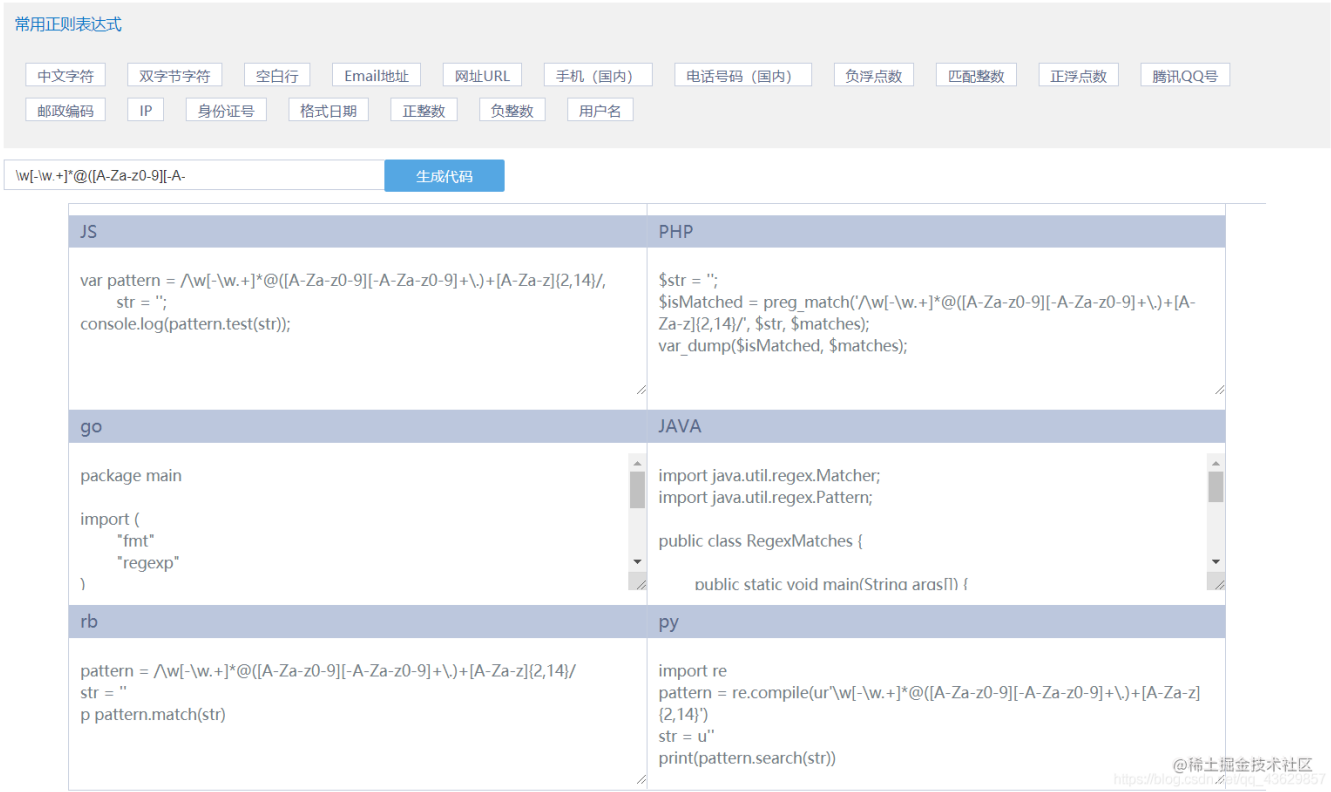
python 复制代码

```
密码 W123456W 符合要求,  长度: 8
密码 wwj123456 非法
密码 W123456789 符合要求,  长度: 10
密码 w123w 非法
密码 W12345678abcdefg 非法
```

正则表达式在线工具

正则表达式在线工具 https://www.w3cschool.cn/tools/index?name=create_reg

这个在线工具提供了常用正则表达式的在线生成功能，可实现诸如字符、网址、邮编、日期、中文等的正则表达式生成功能，并且提供各类常见语言如：**javascript**、**php**、**Go**语言、**java**、**ruby**、**Python**等的正则表达式测试语句供大家参考使用。



公众号

新建文件夹X

> 大自然用数百亿年创造出我们现实世界，而程序员用几百年创造出一个完全不同的虚拟世界。我们用键盘敲出一砖一瓦，用大脑构建一切。人们把1000视为权威，我们反其道行之，捍卫1024的地位。我们不是键盘侠，我们只是平凡世界中不凡的缔造者 。

分类： 后端 标签： Python 后端

文章被收录于专栏：

Python高级进阶，让你更上一层楼！

从概念理解到实践案例以及源码分享，带你熟悉Python高级内容：...

关注专栏

多内容聚合浏览、多引擎快捷搜索、多工具便捷提效、多模式随心畅享，你想要的，这里都有！