

正则表达式

BBCLOUD 林凡

https://github.com/lvancer/course_python

lin029011@163.com

大纲

re模块

正则表达式

练习



re模块

正则表达式就是用一系列的通配符去匹配字符串。

目前最强大的字符串处理工具。Python中使用re模块处理正则表达式。

我们曾经使用过 `w = re.sub('[^a-zA-Z]', '', w)` # 去除非字母

sub其实就是一个替换方法，第一个参数是匹配的方式，即要学习的正则表达式。

第二个是匹配到内容后进行替换的字符串，最后一个是要处理的字符串。

除了sub方法，还有match、search、findall、finditer等主要方法。

定义正则表达式：

```
reg_exp = r'[a-zA-Z]' # 匹配非字母
```

正则表达式可以用r开头后面跟上字符串，就代表这是一个正则表达式。

这样可以避免很多转义的问题。

re模块

匹配一个对象：`match`、`search`。

```
ret = re.match(reg_exp, '12')
if ret is not None:
    print(ret.group()) # 匹配到一个
```

输出 1，即匹配得到的第一个非字母。

如何没有匹配到，返回None。`match`必须从头开始匹配。

```
ret = re.match(reg_exp, 'a12')
print(ret) # 空
ret = re.search(reg_exp, 'a12')
print(ret.group()) # 正常返回1
```

`search`则可以从任意位置开始匹配，只要有匹配到的就返回。

re模块

查找全部匹配：`findall`、`finditer`。

```
ret = re.findall(reg_exp, 'a12')  
print(ret)
```

`findall`以列表返回所有匹配到内容。输出：`['1', '2']`

```
for i in re.finditer(reg_exp, 'a12'):  
    print(i.group())
```

`finditer`则是以迭代器方式【一种需要用for循环方式使用的类型】

使用for循环可以遍历出所有的匹配值。

两者功能等价。

re模块

获得匹配位置：

除了findall，其他匹配后返回的结果都是一个Match对象。

通过group()方法可以得到内容。

通过span()方法就可以获得起始位置和结束位置的元组。

```
for i in re.finditer(reg_exp, 'a12'):  
    print(i.group(), i.span())
```

```
1 (1, 2)  
2 (2, 3)
```

替换：sub。

```
ret = re.sub(reg_exp, '*', 'a12')  
print(ret)
```

a**

将非字母替换成*号。

正则表达式

开始学习正则表达式的书写。

前后匹配：**^**、**\$**。

^ 表示**前匹配**，**\$** 表示**后匹配**。

```
re.search(r'^abc', 'abc1')
```

从头开始匹配，如果匹配到 **^** 后面的内容，则返回匹配的Match对象，否则为空。

```
re.search(r'c1$', 'abc1')
```

尾部如果匹配到 **\$** 前面的内容，则返回匹配的Match对象，否则为空。

功能上类似于 **startswith** 和 **endswith**。

但配合后面的匹配方法，就不止这个功能了。

正则表达式

基础字符匹配

字符	匹配内容
.	除了\n以外的所有字符
\s	空白字符【包括空格、换行符等】
\S	非空白字符
\w	字母与数字
\d	数字
\n	换行符

```
re.search(r'\s.', 'I am a student.')    # I a
re.search(r'\d', 'I am a student.')     # None
re.search(r'stu..nt', 'I am a student.') # student
```


正则表达式

匹配自定义字符组：[]。

```
re.search(r'[abcde]', 'I am a student.')    # a
re.search(r'^[abcde]', 'I am a student.')    # I
re.search(r'[a-z]', 'I am a student.')       # a
```

- 1、匹配中括号内的字母组中任意一个字符。
- 2、加一个 ^ 号，表示非，即匹配不在中括号内的字母组的字符。
- 3、匹配中括号内一个范围的字符。

```
reg_exp = r'^[a-zA-Z]' # 匹配非字母
```

所以这个匹配了非a到z和A到Z的字符，即非字母。

正则表达式

重复匹配：前面都是单个字符的匹配，现在学习匹配N个字符。

乘号【*】表示匹配之前的字符0到无限次。

加号【+】表示匹配之前的字符1到无限次。

大括号【{}】表示匹配多少次。{n}表示n次，{n,m}表示n到m次，{n,}表示至少n次。

```
print(re.findall(r'\w*', 'You are good'))  
print(re.findall(r'\w+', 'You are good'))  
print(re.findall(r'\w{1,3}', 'You are good'))
```

```
['You', '', 'are', '', 'good', '']  
['You', 'are', 'good']  
['You', 'are', 'goo', 'd']
```

正则表达式

问号：？。

单个问号表示0次或1次。

*? 表示非贪婪的0到无限次。

+? 表示非贪婪的1到无限次。

贪婪表示得到最大的匹配，非贪婪表示获得最小的匹配。

```
print(re.findall(r'@ab?', '@abc@abc@'))  
print(re.findall(r'@.+@', '@abc@abc@'))      # 贪婪  
print(re.findall(r'@.+?@', '@abc@abc@'))      # 非贪婪
```

```
['@ab', '@ab']  
['@abc@abc@']  
['@abc@']
```

贪婪情况下，得到第一个@到最后一个@。

非贪婪情况下，当遇到第二个@时就认为匹配到了。

正则表达式

分组是通过匹配获取信息的方式：**括号 ()**。

```
mail = 'lin029011@163.com'  
ret = re.match(r'(.+)@(.+)', mail)  
print(ret.group())      # 没有变化  
print(ret.groups())     # 分组结果
```

```
lin029011@163.com  
('lin029011', '163.com')
```

添加括号后，对于匹配并没有变化，结果的**group**不变。

但是在获取**groups**时，结果就有了**分组**。将我们的邮箱分解为了前后两个部分。

正则表达式

一个url的验证与分组。

```
url = 'https://github.com/lvancer/course_python.git'  
ret = re.match(r'(\w+):\/\/(.+?)\/(.+)\.(\w+)', url)  
print(ret.group())  
print(ret.groups())
```

输出：

```
https://github.com/lvancer/course\_python.git  
('https', 'github.com', 'lvancer/course_python', 'git')
```

实际使用中，我们一般会先用工具验证正则表达式是否正确。

如 <http://tool.chinaz.com/regex/>。当验证正确了，再写代码。

练习

- 1、正则表达式的深入<https://github.com/ziishaned/learn-regex/blob/master/translations/README-cn.md>与练习 <https://regex101.com/>
- 2、自己将1中的正则用python实现。

