



正则表达式介绍



朱映秋

CONTENTS

▶ **PART 1**

匹配规则

▶ **PART 2**

匹配方法

▶ **PART 3**

匹配函数



- 正则表达式是一种用特定字符和组合定义的逻辑公式，用于对字符串进行过滤和匹配。
- 正则表达式的大致匹配过程是：
 - 依次拿出表达式和文本中的字符比较。
 - 如果每一个字符都能匹配，则匹配成功；一旦有匹配不成功的字符则匹配失败。
 - 如果表达式中有量词或边界，则匹配过程会稍有不同。



- 开源中国提供的正则表达式测试工具<http://tool.oschina.net/regex/>，输入待匹配的文本，然后选择常用的正则表达式，就可以得出相应的匹配结果了。

 OSCHINA Gitee  资讯 动弹 专区 问答 活动 软件库 Tool 博客

在线正则表达式测试

在此输入待匹配文本

正则表达式

在此输入正则表达式

☒ 全局搜索 ☐ 忽略大小写

测试匹配

匹配结果:

替换文本

在此输入替换文本

替换

Feedback

常用正则表达式

[匹配中文字符](#)

[匹配双字节字符\(包括汉字在内\)](#)

[匹配空白行](#)

[匹配Email地址](#)

[匹配网址URL](#)

[匹配国内电话号码](#)

[匹配腾讯QQ号](#)

[匹配中国邮政编码](#)

[匹配18位身份证号](#)

[匹配\(年-月-日\)格式日期](#)

[匹配正整数](#)

4-26



➤ 下面为常用的匹配规则。

模式	描述
\w	匹配字母、数字及下划线
\W	匹配不是字母、数字及下划线的字符
\s	匹配任意空白字符，等价于[\t\n\r\f]
\S	匹配任意非空字符
\d	匹配任意数字，等价于[0-9]
\D	匹配任意非数字的字符
\A	匹配字符串开头
\Z	匹配字符串结尾，如果存在换行，只匹配到换行前的 结 束字符串
\z	匹配字符串结尾，如果存在换行，同时还会匹配换行 符
\G	匹配最后匹配完成的位置
\n	匹配一个换行符

模式	描述
\t	匹配一个制表符
^	匹配一行字符串的开头
\$	匹配一行字符串的结尾
.	匹配任意字符，除了换行符，当re.DOTALL标记被指定时 则可以匹配包括换行符的任意字符
[...]	用来表示一组字符，单独列出，比如[amk]匹配a、m或k
[^...]	不在[]中的字符，比如[^abc]匹配除了a、b、c之外的字 符
*	匹配0个或多个表达式
+	匹配1个或多个表达式
?	匹配0个或1个前面的正则表达式定义的片段，非贪婪方 式
{n}	精确匹配n个前面的表达式
{n, m}	匹配n到m次由前面正则表达式定义的片段，贪婪方式
()	匹配括号内的表达式，也表示一个组

CONTENTS

▶ **PART 1**

匹配规则

▶ **PART 2**

匹配方法

▶ **PART 3**

匹配函数



- 使用re的步骤为：先将正则表达式的字符串形式编译为Pattern实例；然后使用Pattern实例的search()、match()、findall()等方法处理文本并获得匹配结果（一个Match实例）；最后使用Match实例获得信息，进行其他的操作。
- re模块中常用的方法及其说明如下。

方法	说明
compile	将正则表达式的字符串转化为Pattern匹配对象
match	将输入的字符串从头开始对输入的正则表达式进行匹配，一直向后直至遇到无法匹配的字符或到达字符串末尾，将立即返回None，否则获取匹配结果
search	将输入的字符串整个扫描，对输入的正则表达式进行匹配，获取匹配结果，否则输出None
split	按照能够匹配的字符串作为分隔符，将字符串分割后返回一个列表
findall	搜索整个字符串，返回一个列表包含全部能匹配的子串
finditer	与findall方法作用类似，以迭代器的形式返回结果
sub	使用指定内容替换字符串中匹配的每一个子串内容



- 匹配方法match(), 向它传入要匹配的字符串以及正则表达式, 就可以检测这个正则表达式是否匹配字符串。
- match() 方法会尝试从字符串的起始位置匹配正则表达式, 如果匹配, 就返回匹配成功的结果; 如果不匹配, 就返回None

```
1 import re
2
3 sentence = "Hello 111 1234 Another_word"
4
5 result = re.match('`Hello\s\d\d\d\s\d{4}\s\w{12}`', sentence)
6 print(result)
```

```
<re.Match object; span=(0, 27), match='Hello 111 1234 Another_word'>
```

```
1 print(result.group())
```

```
Hello 111 1234 Another_word
```

```
1 print(result.span())
```

```
(0, 27)
```

- 开头的^是匹配字符串的开头, 也就是以Hello开头;
- \s匹配空白字符, 用来匹配目标字符串的空格;
- \d匹配数字, 3个\d匹配111; 然后再写1个\s匹配空格;
- {4}以代表匹配前面的规则4次, 前面是\d即匹配4个数字1234;
- \w{10}匹配12个字母及下划线。

- group()方法是输出匹配到的内容; span()是输出匹配的范围, 就是匹配到的结果字符串再原字符串中的位置。



- 如何从匹配的字符串中提取一部分内容：
- 用()括号将想提取的字符串括起来，()实际上标记了一个子表达式的开始和结束位置，被标记的每个子表达式会依次对应每一个分组，调用group()方法传入分组的索引即可获取提取的结果。
 - group()会输出完整的匹配结果，group(1)则会输出第一个被()包围的匹配结果，以此类推。

```
: 1 import re
  2
  3 sentence = "Hello 111 1234 Another_word"
  4
  5 result = re.match('`Hello\s(\d\d\d)\s(\d{4})\s(\w{12})`', sentence)
  6 print(result)
```

<re.Match object; span=(0, 27), match='Hello 111 1234 Another_word'>

```
: 1 print(result.group())
```

Hello 111 1234 Another_word

```
1 print(result.group(0))
```

Hello 111 1234 Another_word

```
1 print(result.group(1))
```

111

```
1 print(result.group(2))
```

1234

```
1 print(result.group(3))
```

Another_word



- 通用匹配，符号为.*（点星）。
- 其中.（点）代表匹配任意字符（除换行符），*（星）代表匹配前面的字符无限次。所以组合在一起.*就是匹配任意字符。
- 这里我们把中间部分省略，全部用.*来代替，\$表示匹配字符串的结尾。

```
1 import re
2
3 sentence = "Hello 111 1234 Another_word"
4
5 result = re.match('^Hello.*word$', sentence)
6 print(result)
```

```
<re.Match object; span=(0, 27), match='Hello 111 1234 Another_word'>
```

```
1 print(result.group())
```

```
Hello 111 1234 Another_word
```

```
1 print(result.span())
```

```
(0, 27)
```



➤ 贪婪匹配

- `*` 会匹配尽可能多的字符。正则表达式中`*` 后面是`\d+`，也就是至少一个数字，并没有指定具体多少个数字，因此，`*` 就尽可能匹配多的字符，这里就把1234匹配了，给`\d+`留下一个可满足条件的数字4，最后得到的内容就只有数字7了。

➤ 非贪婪匹配

- 就是尽可能匹配少的字符。当`*` ?匹配到Hello后面的空白字符时，再往后的字符就是数字了，而`\d+`恰好可以匹配，那么这里`*` ?就不再进行匹配，交给`\d+`去匹配后面的数字。所以这样`*` ?匹配了尽可能少的字符，`\d+`的结果就是1234了。

- 贪婪匹配是尽可能匹配多的字符，非贪婪匹配就是尽可能匹配少的字符。

```
1 import re
2
3 sentence = "Hello 1234 Another_word"
4
5 result = re.match('`He.*(\d+).*word$', sentence)
6 print(result)
```

<re.Match object; span=(0, 23), match='Hello 1234 Another_word'>

```
1 print(result.group())
```

Hello 1234 Another_word

```
1 print(result.group(1))
```

4

```
1 import re
2
3 sentence = "Hello 1234 Another_word"
4
5 result = re.match('`He.*?(\d+).*word$', sentence)
6 print(result)
```

<re.Match object; span=(0, 23), match='Hello 1234 Another_word'>

```
1 print(result.group())
```

Hello 1234 Another_word

```
1 print(result.group(1))
```

1234



- 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。
- 可以发现，增加换行符之后，运行直接报错，也就是说正则表达式没有匹配到这个字符串，返回结果为None。

```
1 import re
2
3 sentence = '''Hello 1234
4 Another_word'''
5
6 result = re.match('`He.*?(\d+).*word$', sentence)
7 print(result)
```

None

```
1 print(result.group())
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-278-666d063f295f> in <module>
----> 1 print(result.group())

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



- 匹配的是除换行符之外的任意字符，当遇到换行符时，`*` `?`就不能匹配了，所以导致匹配失败。这里只需加一个修饰符`re.S`，即可修正这个错误。

```
1 import re
2
3 sentence = '''Hello 1234
4 Another_word'''
5
6 result = re.match('He.*?(\d+).*word$', sentence, re.S)
7 print(result)
```

<re.Match object; span=(0, 24), match='Hello 1234 \nAnother_word'>

```
1 print(result.group())
```

Hello 1234
Another_word

```
1 print(result.group(1))
```

1234

```
1 print(result.span())
```

(0, 24)

- 这个`re.S`在网页匹配中经常用到。因为HTML节点经常会包含换行，加上它，就可以匹配节点与节点之间的换行符。



➤ 常用修饰符

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和\B

➤ 在网页匹配中, 较为常用的有re.S和re.I。



- 当遇到用于正则匹配模式的特殊字符时，需要在前面加反斜线进行转义。

```
1 import re
2
3 sentence = "www.baidu.com(*百度网址*)"
4
5 result = re.match('www\\.baidu\\.com\\(\\*百度网址\\*\\)', sentence, re.S)
6 print(result)
```

<re.Match object; span=(0, 21), match='www.baidu.com(*百度网址*)'>

```
1 print(result.group())
```

www.baidu.com(*百度网址*)

- 例如：.就可以用\\.来匹配。*可以用*来匹配。

CONTENTS

▶ **PART 1**

匹配规则

▶ **PART 2**

匹配方法

▶ **PART 3**

匹配函数



- re模块中使用compile()方法将正则表达式的字符串转化为Pattern匹配对象，其语法格式如下：
`re.compile(pattern, flags=0)`
- compile方法常用的参数及其说明如下。

参数	说明
string	接收string。表示需要转换的正则表达式的字符串。无默认值
flag	接收string。表示匹配模式，取值为运算符“ ”时表示同时生效，如re.I re.M。默认为None



- flag参数的可选值如下。

可选值	说明
re.I	忽略大小写
re.M	多行模式，改变“^”和“\$”的行为
re.S	“.”任意匹配模式，改变“.”的行为
re.L	使预定字符类\w\W\b\B\s\S取决与当前区域设定
re.U	使预定字符类\w\W\b\B\s\S\d\D取决于unicode定义的字符属性
re.X	详细模式，该模式下正则表达式可为多行，忽略空白字符并可加入注释



- `search()`, 它在匹配时会扫描整个字符串, 然后返回第一个成功匹配的结果。
- 在匹配时, `search()`方法会依次扫描字符串, 直到找到第一个符合规则的字符串, 然后返回匹配内容, 如果搜索完了还没有找到, 就返回`None`。

```
1 import re
2
3 sentence = "First sentence is: Hello 1234 Another_word. Second sentence is here."
4
5 result = re.match('He.*?(\\d+).*sentence', sentence)
6 print(result)
```

None

```
1 import re
2
3 sentence = "First sentence is: Hello 1234 Another_word. Second sentence is here."
4
5 result = re.search('He.*?(\\d+).*sentence', sentence)
6 print(result)
```

<re.Match object; span=(19, 59), match='Hello 1234 Another_word. Second sentence'>

```
1 print(result.group())
```

Hello 1234 Another_word. Second sentence

- `match()` 方法从头开始匹配, 匹配不到内容; `search()` 方法可以匹配到内容。



- 这里匹配的<li是开头，然后中间省略，直到singer=" "，双引号里面的内容提取出来，所以代码中用小括号括起来；接着提取第二个内容，它的左边界是>，右边界是< /a >。
- 然后它会搜索整个HTML文本，找到符合正则表达式的第一个内容返回。这里返回“周杰伦 双截棍”。

```
1 html = """
2 <div id="songs-list">
3     <ul id="list" class="list-group">
4         <li class="song">荷塘月色</li>
5         <li class="song">
6             <a href="/2.mp3" singer="周杰伦">双截棍</a>
7         </li>
8         <li class="song">
9             <a href="/3.mp3" singer="齐秦">往事随风</a>
10        </li>
11        <li class="song">
12            <a href="/4.mp3" singer="林俊杰">江南</a>
13        </li>
14    </ul>
15 </div>
16 """
```

```
1 import re
2
3 result = re.search('<li.*?singer="(.*?)">(.*?)</a>', html, re.S)
4 print(result)
```

<re.Match object; span=(69, 171), match='<li class="song">荷塘月色\n' <li class="so

```
1 print(result.group(1))
2 print(result.group(2))
```

周杰伦
双截棍

```
1 import re
2
3 result = re.search('<li.*?singer="([~"]*)">([~<]*)</a>', html, re.S)
4 print(result)
```

<re.Match object; span=(69, 171), match='<li class="song">荷塘月色\n' <li class="so

```
1 print(result.group(1))
2 print(result.group(2))
```

周杰伦
双截棍



- 这里在singer=前面加上 “/4.mp3” \s, 表示会匹配到 “/4.mp3” singer=这个内容, 结果为林俊杰, 江南, 匹配是正确的。

```
1 html = """
2 <div id="songs-list">
3     <ul id="list" class="list-group">
4         <li class="song">荷塘月色</li>
5         <li class="song">
6             <a href="/2.mp3" singer="周杰伦">双截棍</a>
7         </li>
8         <li class="song">
9             <a href="/3.mp3" singer="齐秦">往事随风</a>
10        </li>
11        <li class="song">
12            <a href="/4.mp3" singer="林俊杰">江南</a>
13        </li>
14    </ul>
15 </div>
16 """
```

```
1 result = re.search('<li.*?"/4.mp3"\ssinger="(.*?)">([^\<]*)</a>', html, re.S)
2 print(result)
```

```
<re.Match object; span=(69, 350), match='<li class="song">荷塘月色</li>\n          <li class="so
```

```
1 print(result.group(1))
2 print(result.group(2))
```

林俊杰
江南



- findall()方法会搜索整个字符串，然后返回匹配正则表达式的所有内容。
- 如果只是获取第一个内容，可以用search()方法。当需要提取多个内容时，可以用findall()方法。

```
1 results = re.findall('<li.*?singer="(.*?)">([^\<]*)</a>', html, re.S)
2 print(results)
3 print(type(results))
```

```
[('周杰伦', '双截棍'), ('齐秦', '往事随风'), ('林俊杰', '江南')]
<class 'list'>
```

```
1 for item in results:
2     print(item)
```

```
('周杰伦', '双截棍')
('齐秦', '往事随风')
('林俊杰', '江南')
```



- sub()函数可以替换字符。
- sub()函数的参数为：第一个参数是匹配的字段，第二个参数是要替换成的字符串，第三个参数是原字符串。
- sub()函数第一个参数传入\d+来匹配所有的数字，第二个参数是空，即删除了数字。

```
1 sentence = "12中34文5字6789符"  
2 result = re.sub('\d+', '', sentence)  
3 print(result)
```

中文字符



- 这里把含有a的节点经过sub()的方法处理后就没有了。

```
1 new_html = re.sub('<a.*?>|</a>', '', html)
2 print(new_html)
```

```
<div id="songs-list">
  <ul id="list" class="list-group">
    <li class="song">荷塘月色</li>
    <li class="song">
      双截棍
    </li>
    <li class="song">
      往事随风
    </li>
    <li class="song">
      江南
    </li>
  </ul>
</div>
```




- 在前面处理的基础上，通过findall()方法直接提取信息，其中，strip()函数是去除了空格。

```
1 results = re.findall('<li.*?>(.*?)</li>', new_html, re.S)
2 for item in results:
3     print(item.strip())
```

荷塘月色
双截棍
往事随风
江南

Thank you!

