

## 06 | 转义：正则中转义需要注意哪些问题？

2020-06-24 涂伟忠

正则表达式入门课

[进入课程 >](#)



讲述：涂伟忠

时长 10:52 大小 9.96M



你好，我是伟忠。今天我来和你聊聊转义。转义对我们来说都不算陌生，编程的时候，使用到字符串时，双引号里面如果再出现双引号，我们就可以通过转义来解决。就像下面这样：

```
1 str = "How do you spell the word \"regex\"?"
```

复制代码

虽然转义在日常工作中比较常见，但正则中什么时候需要转义，什么时候不用转义，在真正使用的时候可能会遇到这些麻烦。所以我们很有必要来系统了解一下正则中的转义。



### 转义字符

首先我们说一下什么是转义字符（Escape Character）。它在维基百科中是这么解释的：


在计算机科学与远程通信中，当转义字符放在字符序列中，它将对它后续的几个字符进行替代并解释。通常，判定某字符是否为转义字符由上下文确定。转义字符即标志着转义序列开始的那个字符。

这么说可能有点不好理解，我再来给你通俗地解释一下。转义序列通常有两种功能。第一种功能是编码无法用字母表直接表示的特殊数据。第二种功能是由于表示无法直接键盘录入的字符（如回车符）。

我们这节课说的就是第二种情况，转义字符自身和后面的字符看成一个整体，用来表示某种含义。最常见的例子是，C 语言中用反斜线字符 “\” 作为转义字符，来表示那些不可打印的 ASCII 控制符。另外，在 URI 协议中，请求串中的一些符号有特殊含义，也需要转义，转义字符用的是百分号 “%”。之所以把这个字符称为**转义字符**，是因为它后面的字符，不是原来的意思了。


在日常工作中经常会遇到转义字符，比如我们在 shell 中删除文件，如果文件名中有 \* 号，我们就需要转义，此时我们能看出，使用了转义字符后，\* 号就能放进文件名里了。

```
1 rm access_log*      # 删除当前目录下 access_log 开头的文件
2 rm access_log\*     # 删除当前目录下名字叫 access_log* 的文件
```

 复制代码

再比如我们在双引号中又出现了双引号，这时候就需要转义了，转义之后才能正常表示双引号，否则会报语法错误。比如下面的示例，引号中的 Hello World! 也是含有引号的。

```
1 print "tom said \"Hello World!\" to the crowd."
```

 复制代码

下面是一些常见的转义字符以及它们的含义。

转义字符	意义	ASCII码值（十进制）
\n	换行(LF)，将当前位置移到下一行开头	010
\r	回车(CR)，将当前位置移到本行开头	013
\t	水平制表(HT)（跳到下一个TAB位置）	009
\v	垂直制表(VT)	011
\\	代表一个反斜线字符 \	092
\'	代表一个单引号（撇号）字符	039
\"	代表一个双引号字符	034

## 字符串转义和正则转义

说完了转义字符，我们再来看一下正则中的转义。正则中也是使用反斜杠进行转义的。

一般来说，正则中 `\d` 代表的是单个数字，但如果我们想表示成反斜杠和字母 `d`，这时候就需要进行转义，写成 `\\d`，这个就表示反斜杠后面紧跟着一个字母 `d`。

**REGULAR EXPRESSION**

1 match, 3 steps (~0ms)

:

/

\\d

/ gm

**TEST STRING**

SWITCH TO UNIT TESTS ▶

a\*b+c?**\\d**123d\\

刚刚的反斜杠和 `d` 是连续出现的两个字符，如果你想表示成反斜杠或 `d`，可以用管道符号或中括号来实现，比如 `|d` 或 `[d]`。

## REGULAR EXPRESSION

4 matches, 30 steps (~18ms)

:/ \\|d

/gm

### TEST STRING

SWITCH TO UNIT TESTS ▶

a\*b+c?\\d123d\\

需要注意的是，如果你想用代码来测试这个，在程序中表示普通字符串的时候，我们如果要表示反斜杠，通常需要写成两个反斜杠，因为只写一个会被理解成“转义符号”，而不是反斜杠本身。

下面我给出使用 Python3 来测试的情况，你可以看一下。

复制代码

```
1 >>> import re
2 >>> re.findall('\\|d', 'a*b+c?\\d123d\\') # 字符串没转义"反斜杠"
3 File "<input>", line 1
4     re.findall('\\|d', 'a*b+c?\\d123d\\')
5                                     ^
6 SyntaxError: EOL while scanning string literal
7
8 >>> re.findall('\\|d', 'a*b+c?\\d123d\\')
9 []
10
```

看到这里，你内心是不是有很多问号？为什么转义了还不行呢？我们来把正则表达式部分精简一下，看看两个反斜杠在正则中是什么意思。

复制代码

```
1 >>> import re
2 >>> re.findall('\\|d', 'a*b+c?\\d123d\\')
```

```
3 Traceback (most recent call last):
4   省去部分信息
5 re.error: bad escape (end of pattern) at position 0
```

我们发现，正则部分写的两个反斜杠，Python3 处理的时候会报错，认为是转义字符，即认为是单个反斜杠，如果你再进一步测试在正则中写单个反斜杠，你会发现直接报语法错误，你可以自行尝试。

那如何在正则中正确表示“反斜杠”呢？答案是写四个反斜杠。

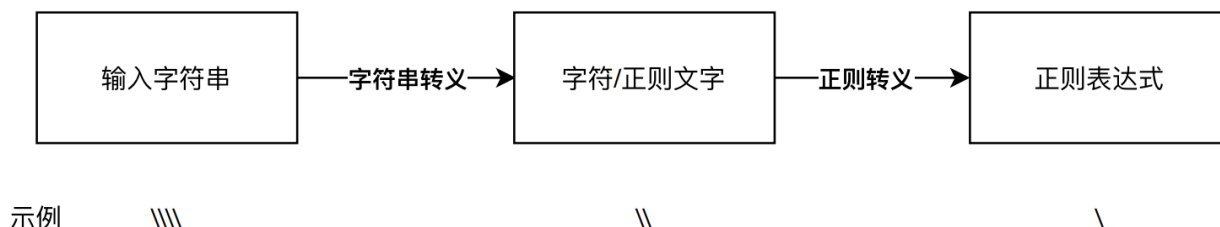
复制代码

```
1 >>> import re
2 >>> re.findall('\\\\\\\\', 'a*b+c?\\\\d123d\\\\')
3 ['\\\\', '\\\\']
```

你可以想一下，为什么不是三个呢？后面的文本部分，也得要用四个反斜杠表示才是正确的么？到这里，你是不是发现，转义其实没那么简单。


我来给你详细解释一下这里面的过程，在程序使用过程中，从输入的字符串到正则表达式，其实有两步转换过程，分别是字符串转义和正则转义。

在正则中正确表示“反斜杠”具体的过程是这样子：我们输入的字符串，四个反斜杠 \\，经过第一步字符串转义，它代表的含义是两个反斜杠 \；这两个反斜杠再经过第二步**正则转义**，它就可以代表单个反斜杠 \了。



你可以用这个过程，推导一下两个和三个反斜杠的转换过程，这样你就会明白上面报错的原因了。

那在真正使用的时候，有没有更简单的方法呢？答案是有的，我们尽量使用原生字符串，在 Python 中，可以在正则前面加上小写字母 r 来表示。

 复制代码

```
1 >>> import re
2 >>> re.findall(r'\\', 'a*b+c?\\d123d\\')
3 ['\\', '\\']
4
```

这样看起来就简单很多，因为少了上面说的第一次转换。

## 正则中元字符的转义


在前面的内容中，我们讲了很多元字符，相信你一定都还记得。如果现在我们要查找比如星号 (\*)、加号 (+)、问号 (?) 本身，而不是元字符的功能，这时候就需要对其进行转义，直接在前面加上反斜杠就可以了。这个转义就比较简单了，下面是一个示例。

 复制代码

```
1 >>> import re
2 >>> re.findall('\\+', '+')
3 ['+']
4
```

## 括号的转义

在正则中方括号 [] 和 花括号 {} 只需转义开括号，但圆括号 () 两个都要转义。我在下面给了你一个比较详细的例子。

 复制代码

```
1 >>> import re
2 >>> re.findall('\\(\\[\\{\\}', '(\\[\\{\\}'))
3 ['(\\[\\{\\}']
4 >>> re.findall('\\(\\[\\[\\]\\]\\{\\}', '(\\[\\{\\}')) # 方括号和花括号都转义也可以
5 ['(\\[\\{\\}']
```


在正则中，圆括号通常用于分组，或者将某个部分看成一个整体，如果只转义开括号或闭括号，正则会认为少了另外一半，所以会报错。



括号的转义示例，你可以参考这里： <https://regex101.com/r/kJfvd6/1>。

## 使用函数消除元字符特殊含义

我们也可以使用编程语言自带的转义函数来实现转义。下面我给出了一个在 Python 里转义的例子，你可以看一下。

 复制代码

```
1 >>> import re
2 >>> re.escape('\d') # 反斜杠和字母d转义
3 '\\\\d'
4 >>> re.findall(re.escape('\d'), '\d')
5 ['\\d']
6
7 >>> re.escape('[+]' ) # 中括号和加号
8 '\\[\\+\\]'
9 >>> re.findall(re.escape('[+]'), '[+]')
10 ['[+]']
11
```

这个转义函数可以将整个文本转义，一般用于转义用户输入的内容，即把这些内容看成普通字符串去匹配，但你还是得好好注意一下，如果使用普通字符串查找能满足要求，就不要使用正则，因为它简单不容易出问题。下面是一些其他编程语言对应的转义函数，供你参考。

编程语言	转义函数
Python	<code>re.escape(text)</code>
Go	<code>regexp.QuoteMeta(text)</code>
Java	<code>Pattern.quote(text)</code>
PHP	<code>preg_quote(text)</code>


## 字符组中的转义

讲完了元字符的转义，我们现在来看看字符组中的转义。书写正则的时候，在字符组中，如果有过多的转义会导致代码可读性差。在字符组里只有三种情况需要转义，下面我来给你讲

讲具体是哪三种情况。


## 字符组中需要转义的有三种情况

1. 脱字符在中括号中，且在第一个位置需要转义：

 复制代码


```
1 >>> import re
2 >>> re.findall(r'^ab', 'ab') # 转义前代表"非"
3 ['^']
4 >>> re.findall(r'\^ab', 'ab') # 转义后代表普通字符
5 ['^', 'a', 'b']
```

2. 中划线在中括号中，且不在首尾位置：

 复制代码

```
1 >>> import re
2 >>> re.findall(r'[a-c]', 'abc-') # 中划线在中间，代表"范围"
3 ['a', 'b', 'c']
4 >>> re.findall(r'[a\-c]', 'abc-') # 中划线在中间，转义后的
5 ['a', 'c', '-']
6 >>> re.findall(r'[-ac]', 'abc-') # 在开头，不需要转义
7 ['a', 'c', '-']
8 >>> re.findall(r'[ac-]', 'abc-') # 在结尾，不需要转义
9 ['a', 'c', '-']
10
```

3. 右括号在中括号中，且不在首位：


 复制代码

```
1 >>> import re
2 >>> re.findall(r'[ ]ab', ' ]ab') # 右括号不转义，在首位
3 [' ', 'a', 'b']
4 >>> re.findall(r'[a]b', ' ]ab') # 右括号不转义，不在首位
5 [] # 匹配不上，因为含义是 a后面跟上b]
6 >>> re.findall(r'[a\]b', ' ]ab') # 转义后代表普通字符
7 [' ', 'a', 'b']
```

## 字符组中其它的元字符



一般来说如果我们要将元字符 (。 $+$ ?() 之类) 表示成它字面上本来的意思, 是需要对其进行转义的, 但如果它们出现在字符组中括号里, 可以不转义。这种情况, 一般都是单个长度的元字符, 比如点号 (。 $.$ )、星号 ( $*$ )、加号 ( $+$ )、问号 (?)、左右圆括号等。它们都不再具有特殊含义, 而是代表字符本身。但如果在中括号中出现  $\backslash d$  或  $\backslash w$  等符号时, 他们还是元字符本身的含义。

 复制代码

```
1 >>> import re
2 >>> re.findall(r'[.*+?()]', '[.*+?()]') # 单个长度的元字符
3 ['.', '*', '+', '?', '(', ')']
4 >>> re.findall(r'[\d]', 'd12\') # \w, \d等在中括号中还是元字符的功能
5 ['1', '2'] # 匹配上了数字, 而不是反斜杠\和字母d
```

下面我来给你简单总结一下字符组中的转义情况, 我们提到了三种必须转义的情况, 其它情况不转义也能正常工作, 但在实际操作过程中, 如果遇到在中括号中使用这三个字符原本的意思, 你可以都进行转义, 剩下其它的元字符都不需要转义。

## 总结

好了, 今天的内容讲完了, 我来带你总结回顾一下。

正则中转义有些情况下会比较复杂, 从录入的字符串文本, 到最终的正则表达式, **经过了字符串转义和正则转义两个步骤**。元字符的转义一般在前面加反斜杠就行, 方括号和花括号的转义一般转义开括号就可以, 但圆括号两个都需要转义, 我们可以借助编程语言中的转义函数来实现转义。另外我们也讲了字符组中三种需要转义的情况, 详细的可以参考下面的脑图。



## 思考题

通过今天的学习，不知道你对转义掌握的怎么样了？再来一个例子加深一下你的理解吧，文本部分是反斜杠，n，换行，反斜杠四个部分组成。正则部分分别是 1 到 4 个反斜杠和字母 n，我用 Python3 写了对应的示例，相应的查找过程是这样子的。

复制代码

```
1 >>> import re
2 >>> re.findall('\n', '\\n\n\\')
3 ['\n'] # 找到了换行符
4 >>> re.findall('\\n', '\\n\n\\')
5 ['\n'] # 找到了换行符
6 >>> re.findall('\\\\n', '\\n\n\\')
7 ['\n'] # 找到了换行符
8 >>> re.findall('\\\\\\n', '\\n\n\\')
9 ['\\n'] # 找到了反斜杠和字母n
```

例子虽然看上去简单，不过你能不能解释出这四个示例中的转义过程呢？

好了，今天的课程就结束了，希望可以帮助到你，也希望你在下方的留言区和我参与讨论，同时欢迎你把这节课分享给你的朋友或者同事，一起交流一下。

上一篇 05 | 断言：如何用断言更好地实现替换重复出现的单词？

## 精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。