正则表达式

概述

正则表达式，Regular Expression，缩写为regex、regexp、RE等。

正则表达式是文本处理极为重要的技术，用它可以对字符串按照某种规则进行检索、替换。

1970年代，Unix之父Ken Thompson将正则表达式引入到Unix中文本编辑器ed和grep命令中，由此正则表达式普及开来。

1980年后，perl语言对Henry Spencer编写的库，扩展了很多新的特性。1997年开始，Philip Hazel开发出了PCRE (Perl Compatible Regular Expressions)，它被PHP和HTTPD等工具采用。

正则表达式应用极其广泛，shell中处理文本的命令、各种高级编程语言都支持正则表达式。

参考 https://www.w3cschool.cn/regex_rmjc/

分类

1. BRE
   基本正则表达式，grep、sed、vi等软件支持。vim有扩展。
2. ERE
   扩展正则表达式，egrep（grep -E）、sed -r等。
3. PCRE
   几乎所有高级语言都是PCRE的方言或者变种。Python从1.6开始使用SRE正则表达式引擎，可以认为是PCRE的子集，见模块re。

基本语法

元字符 metacharacter

代码	说明	举例
.	匹配除换行符外任意一个字符	.
[abc]	字符集合，只能表示一个字符位置。 匹配所包含的任意一个字符	[abc]匹配plain中的'a'
[^abc]	字符集合，只能表示一个字符位置。 匹配除去集合内字符的任意一个字符	[^abc]可以匹配plain中的'p'、'l'、'i'或者'n'
[a-z]	字符范围，也是个集合，表示一个字符位置 匹配所包含的任意一个字符	常用[A-Z][0-9]
[^a-z]	字符范围，也是个集合，表示一个字符位置 匹配除去集合内字符的任意一个字符	
\b	匹配单词的边界	\bb 在文本中找到单词中b开头的b字符
\B	不匹配单词的边界	t\B 包含t的单词但是不以t结尾的t字符，例如write \Bb不以b开头的含有b的单词，例如able
\d	[0-9]匹配1位数字	\d
\D	[^0-9]匹配1位非数字	
\s	匹配1位空白字符，包括换行符、制表符、空格 [\f\r\n\t\v]	
\S	匹配1位非空白字符	
\w	匹配[a-zA-Z0-9_]，包括中文的字	\w
\W	匹配\w之外的字符	

转义

凡是在正则表达式中有特殊意义的符号，如果想使用它的本意，请使用\转义。

反斜杠自身，得使用\\

\r、\n还是转义后代表回车、换行

重复

代码	说明	举例
*	表示前面的正则表达式会重复0次或多次	e\w* 单词中e后面可以有非空白字符
+	表示前面的正则表达式重复至少1次	e\w+ 单词中e后面至少有一个非空白字符
?	表示前面的正则表达式会重复0次或1次	e\w? 单词中e后面至多有一个非空白字符
{n}	重复固定的n次	e\w{1} 单词中e后面只能有一个非空白字符
{n,}	重复至少n次	e\w{1,} 等价 e\w+ e\w{0,} 等价 e\w* e\w{0,1} 等价 e\w?
{n,m}	重复n到m次	e\w{1,10} 单词中e后面至少1个，至多10个非空白字符

练习：
1、匹配手机号码
字符串为"手机号码13851888188。"
2、匹配中国座机
字符串为"号码025-83105736、0543-5467328。"

1、\d{11}
2、\d{3,4}-\d{7,8}

代码	说明	举例
x|y	匹配x或者y	wood took foot food 使用 w|food 或者 (w|f)ood
捕获		
(pattern)	使用小括号指定一个子表达式，也叫分组 捕获后会自动分配组号**从1开始** 可以改变优先级	
\数字	匹配对应的分组	(very) \1 匹配very very，但捕获的组group是very
(?:pattern)	如果仅仅为了改变优先级，就不需要捕获分组	(?:w|f)ood 'industr(?:y|ies)等价'industry|industries'
(?<name>exp) (?'name'exp)	命名分组捕获，但是可以通过name访问分组 Python语法必须是(?P<name>exp)	
零宽断言		wood took foot food
(?=exp)	零宽度正预测先行断言 断言exp一定在匹配的右边出现，也就是说断言后面一定跟个exp	f(?=oo) f后面一定有oo出现
(?<=exp)	零宽度正回顾后发断言 断言exp一定出现在匹配的左边出现，也就是说前面一定有个exp前缀	(?<=f)ood、(?<=t)ook分别匹配ood、ook，ook前一定有t出现
负向零宽断言		
(?!exp)	零宽度负预测先行断言 断言exp一定不会出现在右侧，也就是说断言后面一定不是exp	\d{3}(?!\d)匹配3位数字，断言3位数字后面一定不能是数字 foo(?!d) foo后面一定不是d
(?<!exp)	零宽度负回顾后发断言 断言exp一定不能出现在左侧，也就是说断言前面一定不能是exp	(?<!f)ood ood的左边一定不是f
注释		
(?#comment)	注释	f(?=oo)(?#这个后断言不捕获)

注意：
断言会不会捕获呢？也就是断言占不占分组号呢？
断言不占分组号。断言如同条件，只是要求匹配必须满足断言的条件。

分组和捕获是同一个意思
使用正则表达式时，能用简单表达式，就不要复杂的表达式

贪婪与非贪婪
默认是贪婪模式，也就是说尽量多匹配更长的字符串。

非贪婪很简单，在重复的符号后面加上一个?问号，就尽量的少匹配了。

代码	说明	举例
*?	匹配任意次，但尽可能少重复	
+?	匹配至少1次，，但尽可能少重复	
??	匹配0次或1次，，但尽可能少重复	
{n,}?	匹配至少n次，但尽可能少重复	
{n,m}?	匹配至少n次，至多m次，但尽可能少重复	

```
very very happy 使用v.*y和v.*?y
```

引擎选项

代码	说明	Python
IgnoreCase	匹配时忽略大小写	re.I re.IGNORECASE
Singleline	单行模式 . 可以匹配所有字符，包括\n	re.S re.DOTALL
Multiline	多行模式 ^ 行首、$ 行尾	re.M re.MULTILINE
IgnorePatternWhitespace	忽略表达式中的空白字符，如果要使用空白字符用转义，#可以用来做注释	re.X re.VERBOSE

单行模式：
. 可以匹配所有字符，包括换行符
^ 表示整个字符串的开头，$整个字符串的结尾

多行模式：
. 可以匹配除了换行符之外的字符，多行不影响.点号
^ 表示行首，$行尾，只不过这里的行是每一个行

默认模式：可以看做待匹配的文本是一行，不能看做多行，.点号不能匹配换行符，^和$表示行首和行尾，而行首行尾就是整个字符串的开头和结尾
单行模式：基本和默认模式一样，只是.点号终于可以匹配任意一个字符包括换行符，这时所有文本就是一个长长的只有一行的字符串。^就是这一行字符串的行首，$就是这一行的行尾。
多行模式：重新定义了行的概念，但不影响.点号的行为，^和$还是行首行尾的意思，只不过因为多行模式可以识别换行符了。"开始"指的是\n后紧接着下一个字符；"结束"指的是\n前的字符，注意最后一行结尾可以没有\n

简单讲，单行模式只影响.点号行为，多行模式重新定义行影响了^和$

注意：注意字符串中看不见的换行符,\r\n会影响e$的测试，e$只能匹配e\n

举例
very very happy
harry key

上面2行happy之后，有可能是\r\n结尾。
y$ 单行匹配key的y，多行匹配happy和key的y。
.$指的是此行的结尾，而默认模式和单行模式都是一行，指的是这个大字符串的最后一个字符，就是key的y。