正则表达式

楪祈

正则表达式的定义

首先对正则表达式要处理的文本进行定义

1、文本是字符串的集合，它是由**有限**的字符组成的集合

2、文本本身可以是**无限**的

正则表达式就是来**描述任意文本**的一种特殊表达式，它拥有两个基本要素：

1、表达式ε表示一个文本，仅包含一个长度为0的字符串，也可以理解为 {NULL} 。通常将NULL记作ε；

2. 对字符集中任意字符a，表达式a表示仅有一个字符a的文本，即 {a}

三种基本元算规则：

1、**两个正则表达式的并，记作XIY**，表示的文本是正则表达式X所表示的文本与正则表达式Y所表示的文本的并集。
比如a|b所表示的文本就是{a,b}，类似于加法

2、**两个正则表达式的连接，记作XY**，表示的文本是将X文本中的每个字符串后面连接上Y文本中的每一个字符串之后再把所有这种连接的结果组成一种新的文本。
比如X=a|b，Y=c|d，那么XY所表示的文本就是 {ac,bc,ad,bd}
连接运算取X文本的每个字符串接上Y文本的每一个字符串，最后得到了4中连接结果。这类似于乘法

3、**一个正则表达式的克林闭包，记作X***，表示分别将0个、1个、2个……n个X与自己连接，然后再把所有这些求并。
也就是说X*=ε|X|XX|XXX|……
比如a*这个正则表达式，就表示的是无穷文本 {ε，a，aa，aaa，……} 。这相当于任意次重复一个文本

以上三种运算写在一起时，克林闭包的优先级高于连接运算，而连接运算的优先级高于并运算。

若仅这样，写出的正则会比较复杂，比如表示A-Z，要写成A|B|C|D|E..|Z
故为了简化正则表达式又引入了一些扩展运算（基于三种基本运算）

1. []方括号表示括号内的字符做并运算，同时支持范围描述符“-”。比如[abcd]就等于a|b|c|d，等价于[a-d]；
2. 由于方括号中支持范围描述“-”，如果要使用“-”字符，则需要将它放在方括号的开头，如[-abc]等于-|a|b|c
3. 方括号中以^字符开头，表示在字符集中配出方括号中的所有字符之后，所剩字符的并运算。
   比如[^ab]则表示除了ab以外的所有字符求并。
4. X?表示X|ε。这就代表X与空字符串之间可选
5. X+表示XX*。这等于限制了X至少要重复1次。

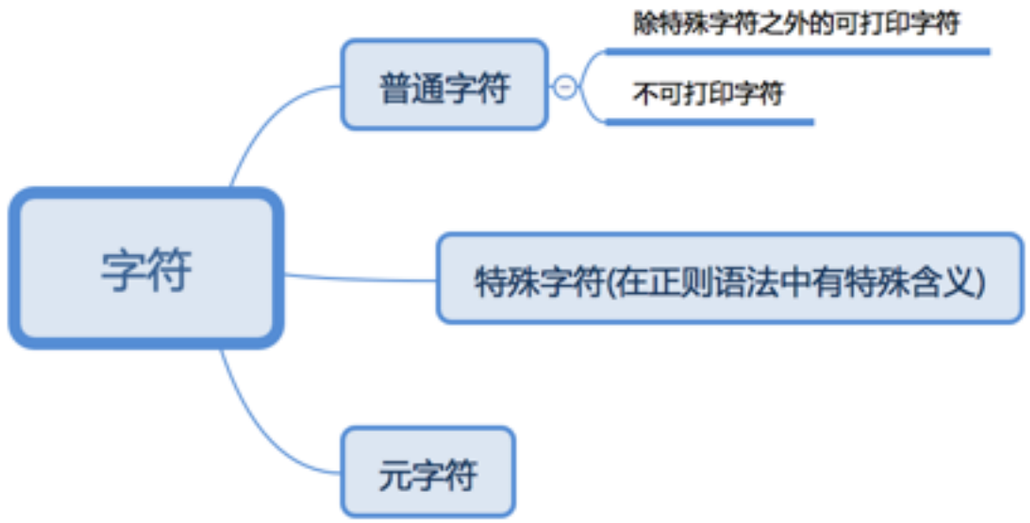
举例：
正则表达式：“name=” 是一个连接运算。由n,a,m,e,= 五个字符连接而成
正则表达式：[a-zA-Z]+ 表示至少出现一个a-z或A-Z的字符

还有的扩展运算，后面放在讲法分类中说~

正则表达式的语法

主要掌握两点：**字符和运算的表示**

字符: 信息单元，一个汉字、一个英文字母、一个数字或一个符号都是字符



元字符: 其实元字符并不是一个字符，而是定义字符的字符
比如:\d表示0-9的数字

字符 - 不可打印的字符

字符	说明
\r	回车符
\n	换行符
\f	换页符
\v	垂直制表符
\t	制表符

建议：用\s来表示空白符，代替\r\n\f\v\t

另：穿插分享下\r与\n的故事，不同操作系统要注意的事项。

字符 - 特殊字符

分类	字符	说明
位置	^	匹配字符串的开始位置([^]不算)
	$	匹配字符串的结束位置
括号类	[]	方括号表达式开始与结束
	{}	限定表达式开始与结束,如{5}
	()	子表达式开始与结束
限定符	*	匹配前面的字符/子表达0或多次
	?	匹配前面的字符/子表达0或1次 或非贪婪匹配
	+	匹配前面的子表达1或多次
	\	与下一字符一起组成特殊含义的字符或其它功能，如\n
	.	匹配除\n以外的任意单字符
	|	|两边子表达式并集关系(或)

注：1、若想表达特殊字符本身的字符，必须转义，比如\^ 、\(、 \\

2、若特殊字符在[]中，会失去“特殊功能”，代表的是字符本身(除了\)。

3、| 也称为正则表达式的分支条件

字符 - 元字符

分类	字符	说明
空白符	\s	表示任意一个空白符，空格、\f\v\n\r\t
	\S	表示任意一个非空白字符
数字	\d	表示任意一个数字，等同于[0-9]
	\D	表示任意一个非数字，等同于[^0-9]
单词	\w	表示任意一个字母、数字、下划线，等同于[a-zA-Z0-9_]
	\W	表示任意一个除字母、数字、下划线以外的字符，等同于[^a-zA-Z0-9_]
边界位置	\b	表示一个字符的边界位置(与空格的边界、行头、行尾)
	\B	表示一个非边界字符

注：1、其中的\S\D\W\B 也称为 “反义字符”

2、^ $ \b 匹配的都是位置

问题：两个正则 [\s\S]+ 与 .+ 有什么区别

运算 - 重复

语法	说明
*	表示前面的字符或子表达式重复任意次
?	表示前面的字符或子表达式重复0或1次
+	表示前面的字符或子表达式重复1次或多次
{n}	表示前面的字符或子表达式重复n次
{n,}	表示前面的字符或子表达式重复至少n次
{n,m}	表示前面的字符或子表达式重复n-m次

注: 上面的语法，也会称为“限定符”

举例: Windows\d+ 匹配Windows后面跟1个或多个数字

运算- 贪婪与非贪婪

当正则表达式中包含限定符时，会匹配**尽可能**多的字符,即”贪婪模式”

举例:a.*b 会匹配最长的以a开头，以b结尾的字符串，比如用来匹配acbabbbb，会匹配整个串

与“贪婪模式”相反的“非贪婪”模式是“**尽可能少**”地匹配字符，只要在前页中的“限定符”后加?

语法	说明
*?	重复任意次，但尽可能少重复
??	重复0-1次，但尽可能少重复
+?	重复1或多次，但尽可能少重复
{n,}?	重复n次以上，但尽可能少重复
{n,m}?	重复n-m次，但尽可能少重复

注:实际使用过程中，*?与+?使用会比较多

上例若改成 a.*?b 会匹配遇到acbabbbb, 会出现多个匹配结果: acb ab

运算-分支条件

一个表达式中有几个规则，只要满足任意一个，就会被匹配。这就是分支条件，通过 | 将多个规则分开。

举例：简单验证电话号码，3位区号-8位电话号码 或4位区号-8位电话号码

^0\d{2}-\d{8}|0d{3}-\d{8}$

^0\d{2,3}-\d{8}$

运算 - 分组

前面学习让我们知道，直接在字符后面加上“限定符”表示单个字符的重复，若想匹配多个字符的重复怎么做呢?

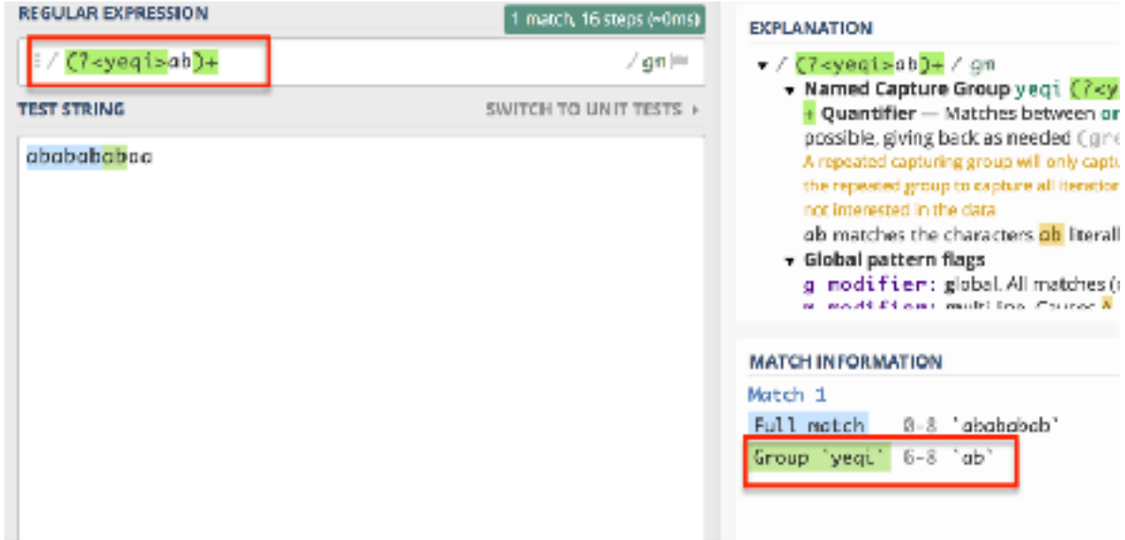
用()来表示子表达式。比如(ab)+ 表示1到多个ab重复，abababab

那为什么又称为分组呢?

正式表达式会将获取到的()子表达式中的内容，缓存下来，并提供方法提取出这个分组值
(注意看相关正则引擎是否实现此功能)

每个分组都会自动拥有一个组号，它的规则是:
从左到右以分组的左括号作为标志，把第一次出现的分组的组号定为1，第二个即2，以此类推下去

分组还可以取名,通过?<name> 或 ?’name’



当然若你只是想将()内的内容当成子表达式用于重复，并不想提取，可以用**(?:exp)**表示不捕获,
例如：(?:ab)+ 只会匹配到多个重复的ab，不会有任何捕获组。

运算 - 反向引用

“反向引用” 用于重复搜索前面某个分组匹配的文本

举例，如何找出连续重复3次或以上的字母，比如aaa,bbbb

用目前学的知识，解决不了此问题,[a-zA-Z]{3,} 是匹配连续的3个或以上的字符，而非同一字符

“反向引用”正在为了解决这一问题而产生

语法：

普通捕获组,\k<number>,简写为\number，其中number为分组编号

比如\1表示反向引用分组1

命名捕获组,\k<name>或\k’name’

反向引用后面也可以加重复限定符，刚才的举例正则是:

([a-zA-Z])\1{2}

注 1、反向引用不一定要紧接着前面的分组,比如\b(\w+)\b\s+\1,表示重复的单词

运算 - 零宽断言

作用是给**指定位置**添加一个**限定条件**，用来规定此位置之前或者之后的字符必须满足限定条件才能使正则中的子表达式匹配成功

举例：一组字符 name=Tom ,score=80　name=Jack ,score=90
　需求：提取成绩90分以上的学生姓名

断言用来声明一个应该为真的事实。正则表达式中只有当断言为真时才会继续进行匹配

为什么是”零宽“呢?

因为匹配“零宽断言”的过程并不**消耗字符**。为什么这么设计?

若消耗字符，后面的表达式想提取断言部分就不可能了，比如上例中
提取成绩90分以上的学生姓名与成绩。

零宽断言是非捕获匹配。

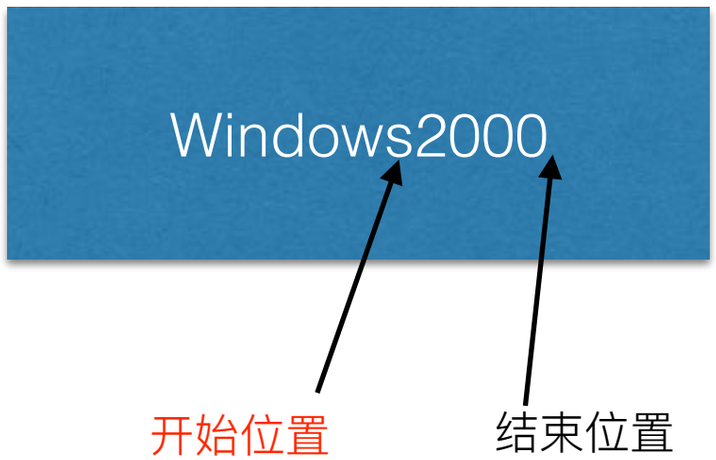
语法	说明
(?=exp)	正向肯定断言，匹配exp前面的位置
(?<=exp)	反向肯定断言，匹配exp后面的位置
(?!exp)	正向否定断言，匹配后面跟的不是exp的位置
(?<!exp)	反向否定断言，匹配前面不是exp的位置

断言-正向肯定断言

（?=exp）正向肯定断言是指：我断言**我出现的位置**的**后面**能匹配表达式exp

“我出现的位置”指的是最终的结果位置（后续其它匹配开始的位置）

比如表达式： Windows(?=95|97|2000|2007)



匹配过程：
匹配Windows表达式后，位置到s与2之间，此时遇到正向肯定断言,尝试开始从此位置匹配exp(95|97|2000|2007)。匹配到，断言为真且结束位置在0之后。

因为是正向断言，返回开始位置：s与2之间的位置

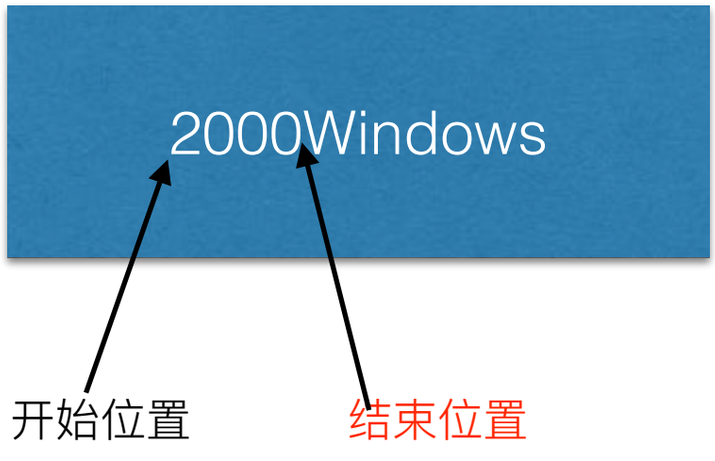
若表达式写成：Windows(?=95|97|2000|2007)\d+ 是可以将2000也匹配出来。
原因是断言出现的位置在s与2之间

问：匹配以ing结尾的单词的正则

断言-反向肯定断言

（?<=exp）反向肯定断言是指：我断言**我出现的位置**的**前面**能匹配表达式exp

比如表达式：(?<=95|97|2000|2007)Windows



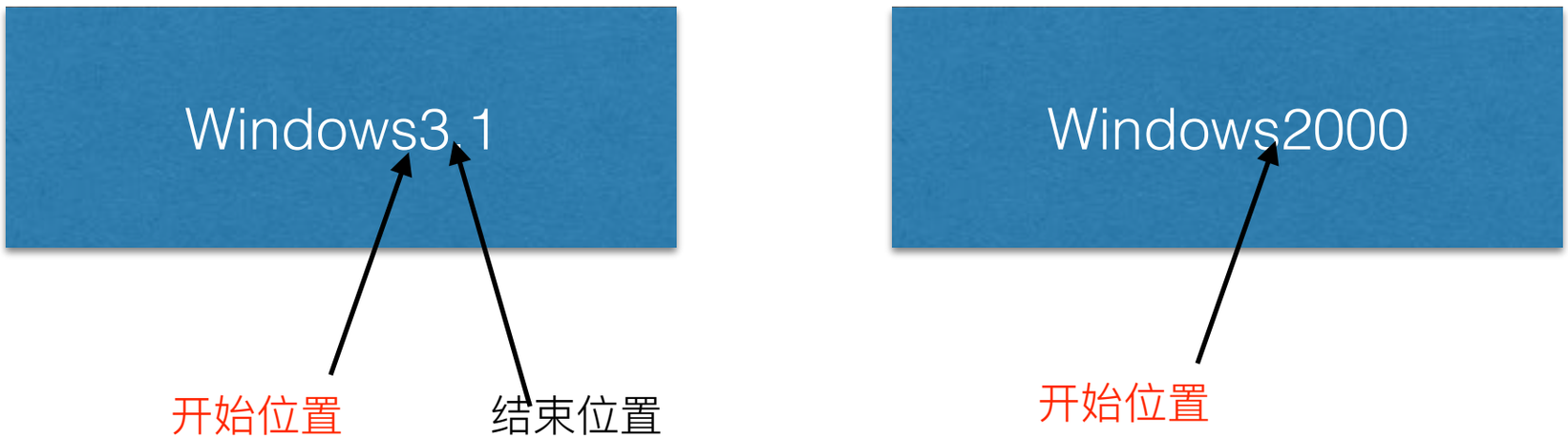
匹配过程：
从开始位置遇到反向肯定断言(?<=95|97|2000|2007),开始匹配，匹配成功，断言为真且结束的位置是0与W之间。
因为是反向肯定断言，故返回的位置是结束位置，也就是0与W之间。

问：匹配以re开头的单词的正则

断言-正向否定断言

（?!exp）正向否定断言是指：我断言**我出现的位置**的**后面 不能**匹配表达式exp

比如表达式： Windows(?!95|97|2000|2007)



匹配Window3.1过程：
匹配Windows后，位置到s与3之间，遇到正向否定断言,尝试开始从此位置匹配exp(95|97|2000|2007)，结果是未匹配到，因为是否定断言，故断言为真。
因为是正向断言，返回开始位置：s与3之间的位置

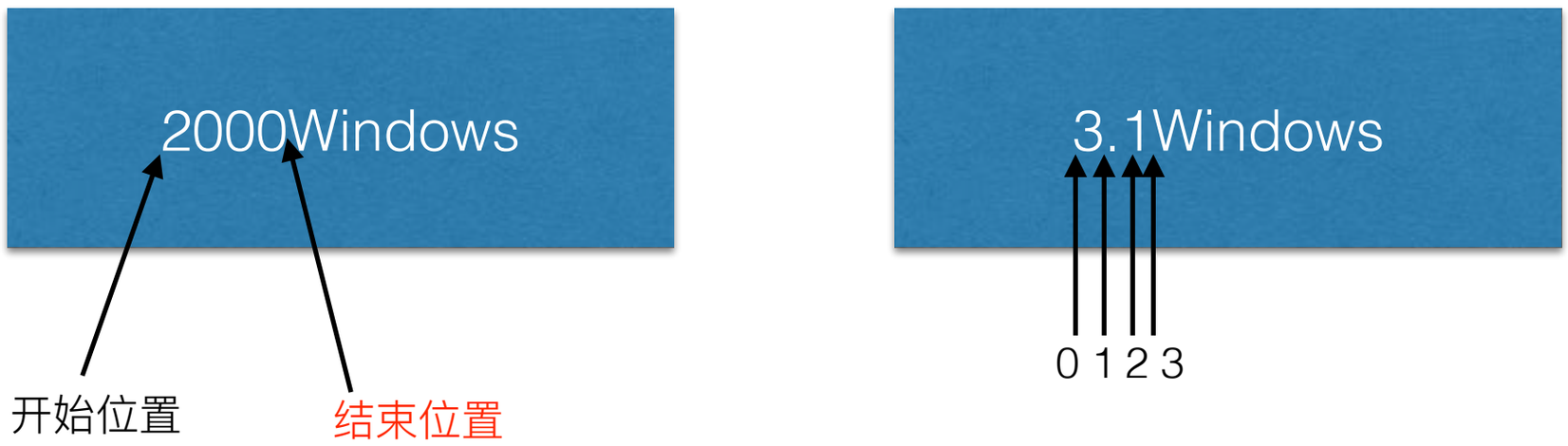
匹配Window2000过程：
匹配Windows后，位置到s与2之间，遇到正向否定断言,尝试开始从此位置匹配exp(95|97|2000|2007)，结果是匹配到，因为是否定断言，故断言为假。匹配失败

问：匹配以不以ing结尾单词的正则

断言-反向否定断言

（?<!exp）反向否定断言是指：我断言**我出现的位置**的**前面 不能**匹配表达式exp

比如表达式：(?<!95|97|2000|2007)Windows



匹配2000Windows过程：
从开始位置遇到反向否定断言(?<!95|97|2000|2007),开始匹配，匹配到0与W位置时匹配成功，但因为是否定断言，故断言为假，匹配失败。

匹配3.1Windows过程：
从开始位置遇到反向否定断言(?<!95|97|2000|2007),开始匹配，因为是否定断言，故0位置断言为真，但匹配后面的Windows失败
以此位置1,2均如此。而位置3匹配Windows成功，故整个匹配结果成功，且断言位置在1与W之间。

问：匹配以不以re开头单词的正则

运算符优先级

正则表达式是从左到右开始计算，并遵循优先级顺序。

下表从最高到最低说明优化级

运算符	说明
\	转义符
[],()	圆括号与方括号
*, +, ?, {n}, {n,}, {n,m}	限定符
^, $, \任何元字符、任何字符	位置与字符
|	分支条件