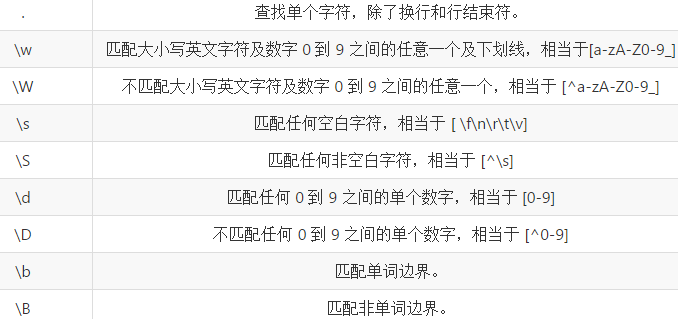
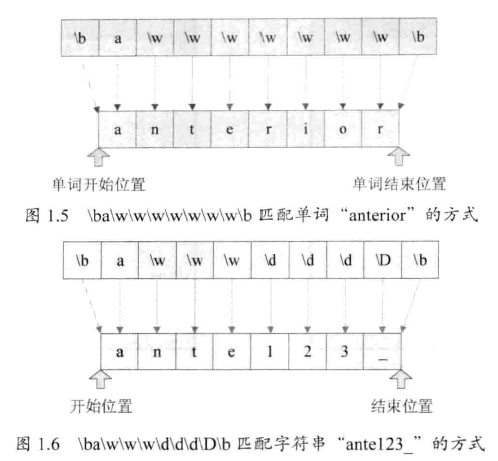
**一、元字符分为两类**：1、匹配位置的元字符。2、匹配单个字符的元字符。

1、匹配位置的元字符有：^（以某str开头）$（以某str结尾） \b（以某str开头或者结尾。

\ba：以a开头但是a\b：以a结尾）

2、匹配单个字符的元字符有：. \w \W \b \s \S \d \D





**二、字符集：字符类中的[]**：匹配其中的任何一个字符或匹配除了[]中字符的任意其他字符

[123456789]:匹配其中的任何一个数字

[1-9]、[a-zA-z]:匹配所有的英文字母

[^a-z]:匹配非小写字母的字符

**三、字符转义使用\**

**四、反义表达式使用^**

**五、限定符：**上面的元字符和字符类[]都是匹配单个字符，但是更多的时候可能需要多次匹配，这时候需要使用限定符规定重复次数。



巧记方法：？表示不确实，所以相当于{0,1}，因为不知道是不重复还是重复一次。+一看就是需要量，所以至少要来1次，所以相当于{1，}。\*长得像0，就是至少{0，}。

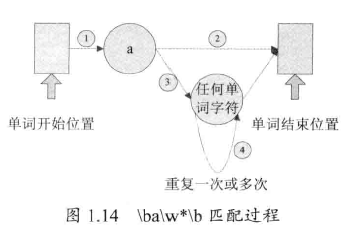
Tip：

1、限定符都是跟在字符后面，也就是相当于单个字符的运算符。需要跟在单个字符的后面。比如“.\*”，至少重复0次以上非换行符的字符。

2、说单个字符的时候，其实都是单个字符的概念。

3、上面的如果理解为跟在字符或者字符集后面的操作符的话，那么上面的操作符和其他的一样都是从左往右开始运算的。于是，上面的\*？（至少0次）和+？（至少1次）都可以理解了，但是加上？都代表懒惰，也就是尽可能少的重复。

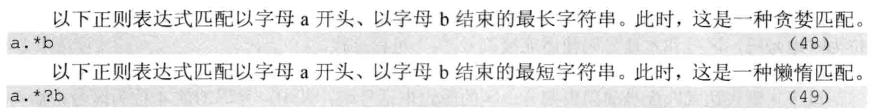
**4、注意a\*：没有这种用法，因为a不是一个元字符或者字符集，所以上面的匹配没有任何意义。**



**六、贪婪匹配和懒惰匹配**：

不加？，比如\*和+都是贪婪匹配，但是\*？和+？都是懒惰匹配。

所谓的贪婪匹配就是匹配长的满足要求的字符串，但是懒惰匹配的意思就是能短则短。



再次注意没有a\*这种用法，因为a不是一个元字符。

比如：

对于源字符串：aabab

如果正则表达式为：a.\*b那么将会匹配出aabab

如果正则表达式为：a.\*?b，将会匹配出aab和ab不会匹配出最长的字符串

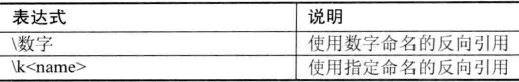
**七、分组**

**分组又叫做子表达式，使用（）进行标识。可以自己给分组进行命名，命名的方式为：**

**（?<name>表达式）。如果不自己命名的话，正则表达式会自动给这个分组有一个数字命名。**

**从左至右的第一个分组的序号是1，依次往右递增。**

**反向引用分组：**



**八、零宽度断言和负向零宽度断言：?=,?!,?<=,?<!**



**?!**就是位置后面不是expression：比如(jave)(?!(hello)),意味着java这个字符串后面不能是hello，否则不进行匹配。

**?=**刚好是这个位置后面必须是这个才能进行匹配，而且会返回这个位置前面的部分。

比如：[a-z]\*（？=ing）可以匹配cooking和singing中的“cook”与“sing”

**?<!** 这个位置的前面一定不能是，否则不进行匹配。例子:(?<!(java))(hello)匹配hello字符串但是前面不能有java

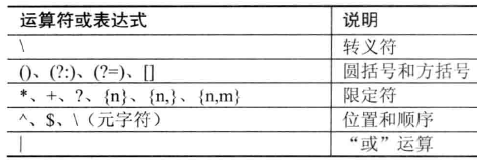
**?<=** 这个位置的前面是，并且匹配这个位置前面的东西，比如：(?<=abc).\*可以匹配abcdefgabc中的defgabc而不是abcdefg

<http://blog.csdn.net/lcore/article/details/8744392>

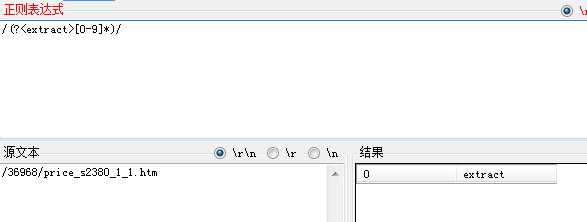
**九、()，[],{}**

**1、子表达式 2、字符集 3、重复次数**

**优先级总结：**



**下面的这种情况不能匹配到东西：**



当正则部分把所有字符串补全的时候可以提取出相应的东西。

