模式匹配基础

匹配字符串字面值：

原始字符串：”yanbo”
正则表达式：”yanbo”

如上就是最简单最直接的字面匹配字符串，属于最简单的正则表达式。

匹配数字：

正则表达式： “\d” 或者 “[0-9]” 或者 “[0123456789]”
原始字符串：”3”

如上三个正则表达式的匹配效果是一样的，都是匹配指定字串中的一个 0-9 的数字，只是匹配一个数字。三种写法各有优势，”\d”可以表示任意数字，”[m-n]”可以表示 m-n 的一位数字，”[abcd]”可以匹配指定的 abcd 中的一个数字。特别的认为”[0123456789]”匹配的是”0123456789”字符串是错误的想法，特别留意，匹配的是一位！！！还可以如下：

正则表达式：”[015-7]”
可匹配的数字：0,1,5,6,7

匹配非数字字符：

正则表达式：”\D” 或者 “[^0-9]” 或者 “[^\d]”
匹配字串：非数字字符（切记：也是匹配一个字符）。

如上三个正则表达式的匹配效果是一样的，都是匹配一个非数字字符。”[]”中的”^”就是取反，除去”^”后面的内容。

匹配单词和非单词字符：

首先强调一点，匹配的是单词和非单词的字符，不是单词！！！

“\w” 这个简写式将匹配所有单词字符（字母、数字、下划线）。
“\D” 匹配非数字字符，包含空格、标点（引号、连字符、反斜杠、方括弧）等字符。
在英语环境下 “\w” 与 “[_a-z0-9A-Z]” 匹配相同的单词字符。
“\W” 匹配非单词字符（空格、标点及其他非字母、数字字符等）。
在英语环境下 “\W” 与 “[^_a-z0-9A-Z]” 匹配相同的单词字符。

如下提供更多的字符简写式，不过要注意！！！！*不是所有的正则表达式处理器可以识别以下简写式*。

字符简写式	描述
\a	报警符
[\b]	退格字符
\c x	控制字符
\d	数字符
\D	非数字符
\w	单词字符
\W	非单词字符
\0	空字符
\x xx	字符的十六进制值
\u xxx	字符的 Unicode 值

匹配空白符与非空白符：

正则表达式：”\s” 或者 “[\t\n\r]”
匹配结果： 空白符（空格、制表符、换行符、回车符）

如上 “[\t\n\r]” 的 “\t\n\r” 前至少有一个空格，否则无效。

正则表达式：”\S” 或者 “[^\s]” 或者 “[^ \t\n\r]”
匹配结果： 非空白符（除空格、制表符、换行符、回车符以外的字符）。

除过 “\s” 匹配的字符之外还有如下一些不太常见的空白符：

字符简写式	描述
\s	空白符
\S	非空白符
\f	换页符
\h	水平空白符
\H	非水平空白符

字符简写式	描述
\n	换行符
\r	回车符
\t	水平制表符
\v	垂直制表符
\V	非垂直制表符

匹配任意符：

“.” 匹配除过行结束符以外的所有字符，个别情况除外。

正则表达式：”\b\w\w\w\b”
匹配字符串：三个字符的单词

如上表达式中”\b”简写式匹配单词边界，不消耗任何字符，一般两个边界都写。如下例子是”.”的特例。

正则表达式： `"a.c\."`
匹配字符串：axc.

如上匹配的是 axc. 其中 x 可以是任意字符，而最后 . 就是转义字符而不是 . 。

实战装逼展示：

学习了一些入门级的正则表达式以后我们就来装装逼，装逼也要装的有逼格，所以如上第一章的工具就太简单了，我们用 Linux 的 sed 流编辑器来装逼。（PS：不了解 Linux 的 sed 流编辑器的请自己 google 或者度娘）。

这里演示一个将字符串”This is YanBo’s Blog!”以 Html 二级标题输出的例子。

sed 编辑器命令：

```
echo "This is YanBo's Blog!" | sed 's/^/<h2>/;s/$/<\/h2>/p;q'
```

如上命令在 linux 终端执行之后如下：

1. echo 打印”This is YanBo’s Blog!”到屏幕，然后通过”|”管道将输出作为 sed 的输入。
2. sed 默认操作是直接复制每行输入并输出。
3. `s/^/<h2>/`在行的开头(^)添加 HTML 的二级标题`<h2>`标签。
4. 分号(;)用于分割命令。

5. `s/$/<\/h2>/`用于在结尾($)添加 HTML 二级标题`</h2>`标签。
6. 命令 p 打印受影响的一行。
7. 命令 q 结束 sed 程序。

总结：

这一部分我们学习正则表达式的基础匹配，算是入门技能。接下来继续带你装逼带你飞。

三、正则表达式—边界

热身准备：

不多 BB，边界这部分算是正则表达式的核心之一。断言（零宽度断言）这一词足矣。

断言（零宽度断言）标记边界，并不耗费字符，不匹配字符，匹配字符串中的位置。

字符串或者行起始与结束：

“^” 匹配行或者字符串的起始位置，或者整个文档的起始位置。
“$” 匹配行或者字符串结尾位置。

例子：
正则表达式： “^word$”
匹配字符串： word（仅有该单词的字符串，w 开头，d 结束）。

单词编辑与非单词边界：

“\bxxx\b” 匹配单词边界。

“\b” 是一个零宽度断言，表面上它会匹配空格或者行起始，实际上它匹配的是一个零宽度不存在的东西。

“\B” 是一个匹配非单词边界，匹配除单词之外的位置。

例子：
正则表达式： “\Ba\B”
匹配字符串： “fhrrhahhr”(类似这样的 a 两边不是单词边界的字符，这里匹配字符 a)。

其他锚位符：

“\A” 与”^”相似，该锚位符匹配主题词的开始。这个写法不是在所有的正则表达式程序中都可以使用的，但是可以在 Perl 和 PCRE 中使用，要匹配主题词的结尾用”\Z”, 某些上下文中还可以用”\z”。

例子：
正则表达式：”\Aaaaa\Z”
匹配字串：”aaaa” (仅仅 aaaa 开头结尾的字符串，也即主题词开始结束)

使用元字符的字面值：

可以用”\Q”和”\E”之间的字符集匹配字符串字面值。”.^$*+?|(){}[]-“这 15 个元字符在正则表达式中有特殊含义，用来编写匹配模式。其中的连字符”-“在正则表达式的方括号中用来表示范围，其他情况下无特殊含义。你在正则表达式中直接输入这些字符不会显示，如果想显示这些字符本身你就需要把他放在”\Q”和”\E”之间，当然，也可以在其前面加”\”即可。

例子：
正则表达式： `"\Q$\E"` 或者 `"\$"`
匹配字符： $字符本身

实战装逼一把：

继续像上一节一样装逼，继续添加标签，继续使用 linux 的 sed 命令 BB。sed 中的命令(i)允许你在文件或者字符串中的某个位子之前插入文本，与(i)相反的命令是(a)，他是在某个位置之后添加文本。关于 sed（或者 grep 或者 vi 与 vim）的实战正则表达式例子这里不给出，自行 google 尝试，这里重在讲解正则表达式。

总结：

学习了边界与断言（零宽度断言）。没啥总结的，开始正则表达式的精华，继续 BB。

四、选择、分组和后向引用

选择操作：

选择操作可以在多个可选模式中匹配一个。譬如你想在”The **Android** developer need fix bug on the Bug System.”中找出”the”（the, The, THE）出现过多少次，这时候就使用选择模式。

正则表达式：”(the|The|THe|THE|tHE|thE|ThE|tHe)” 或者 “(?i)the”
原始字符串：”The android developer need fix bug on the Bug System.”
匹配结果： The，the

如上正则表达式匹配所有大小写的 the。

以下是其他各种选项和修饰符(注意：如下选项不实用所有正则表达式的平台)：

选项	描述	支持平台
(?d)	unix 中的行	java
(?i)	不区分大小写	PCRE、Perl、Java
(?J)	允许重复的名字	PCRE
(?m)	多行	PCRE、Perl、Java
(?s)	单行(dotall)	PCRE、Perl、Java
(?u)	Unicode	java
(?U)	默认最短匹配	PCRE
(?x)	忽略空格和注释	PCRE、Perl、Java
(?-…)	复原或者关闭选项	PCRE

子模式：

子模式是正则表达式分组中的一个或者多个分组，就是模式中的模式。多数情况，子模式中的条件能得到匹配的前提是前面的模式得到匹配，但是也有例外（譬如”(the|THE|The)”匹配 THE 不依赖于 the，因为 the 会先去匹配，这个例子有三个子模式，分别是 the、THE、The），子摸式写法很多种，这里只关注括弧中的子模式。

例子（子模式匹配依赖于前面的模式）：

正则表达式：(t|T)h(e|E)
匹配：the、The、thE、ThE

上面例子中第二个子模式”(e|E)”依赖于第一个子模式”(t|T)”。

特别的，括弧对于子模式不是必须的！！！！！如下：

正则表达式：”\b[tT]h[eE]”
匹配：the、The、thE、ThE

以上”[tT]”字符组可以看作第一个子模式，同理第二个。

捕获分组和后向引用：

当一个模式的全部或者部分内容由一对括号分组时，他就对内容进行捕获并临时存储于内存中，可以通过后向引用重用捕获的内容，形式如下：

“\1” 、”\2” 或者 “1"、"2”，捕获的 n 个分组。

在 sed 命令中只接受”\1”这种分组。

例子（使用 linux 的 sed 命令模拟后向引用）：

```
echo "YanBo is an Android Developer!" | sed -En 's/(YanBo is) (an Android Developer)/\2 \1/p'
```

输出：an Android Developer YanBo is!
解释：
-E 是 sed 调运 ERE（扩展正则表达式），因此，括号可以当作字面值来使用。
-n 覆盖打印每一行的默认设置。
捕获分组 1,2 进行替换。

命名分组：

命名分组就是有名字的分组。由此可以通过名字引用分组而不是数字。

命名分组语法：

语法	描述
`(?<name>...)`	命名分组
`(?name...)`	另一种命名分组方式
`(?P<name>...)`	Python 中的命名分组
`\k<name>`	在 Perl 中引用分组名
`\k'name'`	在 Perl 中引用分组名
`\g{name}`	在 Perl 中引用分组名
`\k{name}`	在.NET 中引用分组名
`(?P=name)`	在 Python 中引用分组名

非捕获分组：**

非捕获分组不会将其内容存储在内存中。在你并不想引用分组时可以使用它。由于不存储分组，所以非捕获分组性能较高。

例子：

捕获分组的写法：”(the|THE|The)”
不需要任何后向引用可以写为：”(?:the|The|THE)”
不区分大小写：”(?i)(?:the)” 或者 “(?:(?i)the)” 或者（推荐）”(?i:the)”

原子分组：

还有一种非捕获分组时原子分组。如果你用正则表达式引擎进行回朔操作，这种分组可以关闭回朔操作，但是他只争对原子分组部分，而不是整个表达式。语法如下：

“(?>the)”

正则表达式慢的一个原因就是回朔操作。

总结：

没啥总结的，继续装逼继续飞，下面的逼格更高更嗨！！！

六、正则表达式—量词

贪心、懒惰、占有：

量词本来是贪心的。贪心量词首先会匹配整个字串，然后一个一个回退（回朔），直到找到匹配的为止。所以他最消耗资源。

懒惰的量词使用另一种策略，他从目标的起始位置开始寻找匹配，每次检查一个字符，最后尝试匹配整个字符串。想要量词变为懒惰的，必须在普通量词后添加一个问号(？)。

占有量词会覆盖整个目标然后尝试寻找匹配内容，但是只尝试一次，不会回朔。占有量词是在普通量词之后添加一个加号(+)。

正则表达式*、+、?进行匹配：

如下基本量词默认都是贪心的。

语法	描述
?	零个或者一个

语法	描述
+	一个或者多个
*	零个或者多个

例如：
正则表达式：”9+”
匹配：一个或者多个 9

匹配特定次数：

如下花括弧量词是匹配最精确的量词，默认也是贪心的。

语法	描述
{n}	精确匹配 n 次
{n,}	匹配 n 或者更多次
{m,n}	匹配 m-n 次
{0,1}	与？相同，零次或一次
{1,0}	与+相同，一次或更多
{0,}	与*相同，零次或者更多

懒惰量词：

这个懒惰量词直接实战来说：

正则表达式：”8?”
匹配：一个或者 0 个 8

正则表达式：”8?？” (懒惰)
匹配：一个 8 都没匹配，因为懒惰，尽可能少。

正则表达式：”8*？” (懒惰)
匹配：一个 8 都没匹配，因为懒惰，尽可能少。

正则表达式：”8+？” (懒惰)
匹配：匹配了一个 8。

正则表达式：”8{3,8}？” (懒惰)
匹配：匹配了三个 8。

懒惰量词表：

语法	描述
??	懒惰匹配 0-1 次
+?	懒惰匹配 1-多次
*?	懒惰匹配 0-多次
{n}?	懒惰匹配多次
{n,}?	懒惰匹配 n-多次
{m,n}?	懒惰匹配 m-n 次

占有量词：

占有量词表：

语法	描述
?+	占有匹配 0-1 次
++	占有匹配 1-多次
*+	占有匹配 0-多次
{n}+	占有匹配多次
{n,}+	占有匹配 n-多次
{m,n}+	占有匹配 m-n 次

例子：

正则表达式：”1.*+”
匹配：所有的 1 全被高亮。

正则表达式：”.*+1”
匹配：没有匹配，因为没有回朔。

正则表达式：”.*1”
匹配：匹配末尾为 1 的字串，贪心模式。

总结：

这里介绍的量词算是正则表达式效率方面的精华所在，没啥解释的，继续装逼继续飞。

七、正则表达式—环视

环视是非捕获分组，也称作零宽断言。

正前瞻：

例子：

正则表达式：”(?i)aaa (?=bbb)”
原始串：”aaa ccc bbb aaa bbb ccc aaa”
匹配：只匹配第二处”aaa”。

以上就是匹配 aaa，同时要求 aaa 单词之后紧随的是 bbb。使用了正前瞻达到目的。

反前瞻：

反前瞻是正前瞻的取反操作。

例子：

正则表达式：”(?i)aaa (?!bbb)”
原始串：”aaa ccc bbb aaa bbb ccc”
匹配：只匹配第一处”aaa”。

以上就是匹配 aaa，同时要求 aaa 单词之后紧随的不能是 bbb。使用了反前瞻达到目的。

正后顾：

正后顾与正前瞻方向相反。

例子：

正则表达式：”(?<=aaa) bbb”
原始串：”aaa ccc bbb aaa bbb ccc aaa ”
匹配：只匹配第二处”bbb”。

反后顾：

反后顾与反前瞻方向相反。

例子：

正则表达式：”(?

总结：

这块更加不需要 BB 总结，就是例子理解，照猫画虎就行。

大结局

整个正则表达式基础到这就差不多够用了。总结学习方法就是大胆实践，多乱想然后编辑器验证就行了。