正则表达式的构造摘要

的非捕获组

构造	匹配
字符	
x	字符 *x*
\\	反斜线字符
\0*n*	带有八进制值 0 的字符 *n*（0 <= *n* <= 7）
\0*nn*	带有八进制值 0 的字符 *nn*（0 <= *n* <= 7）
\0*mnn*	带有八进制值 0 的字符 *mnn*（0 <= *m* <= 3、0 <= *n* <= 7）
\x*hh*	带有十六进制值 0x 的字符 *hh*
\u*hhhh*	带有十六进制值 0x 的字符 *hhhh*
\t	制表符（'\u0009'）
\n	新行（换行）符（'\u000A'）
\r	回车符（'\u000D'）
\f	换页符（'\u000C'）
\a	报警（bell）符（'\u0007'）
\e	转义符（'\u001B'）
\c*x*	对应于 *x* 的控制符
字符类	
[abc]	a、b 或 c（简单类）
[^abc]	任何字符，除了 a、b 或 c（否定）
[a-zA-Z]	a 到 z 或 A 到 Z，两头的字母包括在内（范围）
[a-d[m-p]]	a 到 d 或 m 到 p：[a-dm-p]（并集）
[a-z&&[def]]	d、e 或 f（交集）
[a-z&&[^bc]]	a 到 z，除了 b 和 c：[ad-z]（减去）
[a-z&&[^m-p]]	a 到 z，而非 m 到 p：[a-lq-z]（减去）
预定义字符类	
.	任何字符（与行结束符可能匹配也可能不匹配）
\d	数字：[0-9]
\D	非数字： [^0-9]

\s	空白字符：[\t\n\x0B\f\r]
\S	非空白字符：[^\s]
\w	单词字符：[a-zA-Z_0-9]
\W	非单词字符：[^\w]

POSIX 字符类（仅 US-ASCII）

\p{Lower}	小写字母字符：[a-z]
\p{Upper}	大写字母字符：[A-Z]
\p{ASCII}	所有 ASCII：[\x00-\x7F]
\p{Alpha}	字母字符：[\p{Lower}\p{Upper}]
\p{Digit}	十进制数字：[0-9]
\p{Alnum}	字母数字字符：[\p{Alpha}\p{Digit}]
\p{Punct}	标点符号：!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
\p{Graph}	可见字符：[\p{Alnum}\p{Punct}]
\p{Print}	可打印字符：[\p{Graph}\x20]
\p{Blank}	空格或制表符：[\t]
\p{Cntrl}	控制字符：[\x00-\x1F\x7F]
\p{XDigit}	十六进制数字：[0-9a-fA-F]
\p{Space}	空白字符：[\t\n\x0B\f\r]

java.lang.Character 类（简单的 java 字符类型）

\p{javaLowerCase}	等效于 java.lang.Character.isLowerCase()
\p{javaUpperCase}	等效于 java.lang.Character.isUpperCase()
\p{javaWhitespace}	等效于 java.lang.Character.isWhitespace()
\p{javaMirrored}	等效于 java.lang.Character.isMirrored()

Unicode 块和类别的类

\p{InGreek}	Greek 块（简单块）中的字符
\p{Lu}	大写字母（简单类别）
\p{Sc}	货币符号
\P{InGreek}	所有字符，Greek 块中的除外（否定）
[\p{L}&&[^\p{Lu}]]	所有字母，大写字母除外（减去）

边界匹配器

^	行的开头
$	行的结尾
\b	单词边界
\B	非单词边界
\A	输入的开头
\G	上一个匹配的结尾
\Z	输入的结尾，仅用于最后的结束符（如果有的话）
\z	输入的结尾

Greedy 数量词(以下的 XY 均代表一个正则表达式)

X?	*X*，一次或一次也没有
*X**	*X*，零次或多次
X+	*X*，一次或多次
X{*n*}	*X*，恰好 *n* 次
X{*n*,}	*X*，至少 *n* 次
X{*n*,*m*}	*X*，至少 *n* 次，但是不超过 *m* 次

Reluctant 数量词

X??	*X*，一次或一次也没有
*X**?	*X*，零次或多次
X+?	*X*，一次或多次
X{*n*}?	*X*，恰好 *n* 次
X{*n*,}?	*X*，至少 *n* 次
X{*n*,*m*}?	*X*，至少 *n* 次，但是不超过 *m* 次

Possessive 数量词

X?+	*X*，一次或一次也没有
*X**+	*X*，零次或多次
X++	*X*，一次或多次
X{*n*}+	*X*，恰好 *n* 次
X{*n*,}+	*X*，至少 *n* 次
X{*n*,*m*}+	*X*，至少 *n* 次，但是不超过 *m* 次

Logical 运算符

XY	*X* 后跟 *Y*
X|*Y*	*X* 或 *Y*
(*X*)	X，作为捕获组

Back 引用

n	任何匹配的 n^{th} 捕获组

引用

\	Nothing，但是引用以下字符
\Q	Nothing，但是引用所有字符，直到 \E
\E	Nothing，但是结束从 \Q 开始的引用

特殊构造（非捕获）

(?:*X*)	*X*，作为非捕获组
(?idmsux-idmsux)	Nothing，但是将匹配标志 i d m s u x on - off
(?idmsux-idmsux:*X*)	*X*，作为带有给定标志 i d m s u x on - off
(?=*X*)	*X*，通过零宽度的正 lookahead
(?!*X*)	*X*，通过零宽度的负 lookahead
(?<=*X*)	*X*，通过零宽度的正 lookbehind
(?<!*X*)	*X*，通过零宽度的负 lookbehind
(?>*X*)	*X*，作为独立的非捕获组

反斜线、转义和引用

反斜线字符（'\'）用于引用转义构造，如上表所定义的，同时还用于引用其他将被解释为非转义构造的字符。因此，表达式 \\ 与单个反斜线匹配，而 \{ 与左括号匹配。

在不表示转义构造的任何字母字符前使用反斜线都是错误的；它们是为将来扩展正则表达式语言保留的。可以在非字母字符前使用反斜线，不管该字符是否非转义构造的一部分。

根据 Java Language Specification 的要求，Java 源代码的字符串中的反斜线被解释为 Unicode 转义或其他字符转义。因此必须在字符串字面值中使用两个反斜线，表示正则表达式受到保护，不被 Java 字节码编译器解释。例如，当解释为正则表达式时，字符串字面值 "\b" 与单个退格字符匹配，而 "\\b" 与单词边界匹配。字符串字面值 "\(hello\)" 是非法的，将导致编译时错误；要与字符串（hello）匹配，必须使用字符串字面值 "\\(hello\\)"。

字符类

字符类可以出现在其他字符类中，并且可以包含并集运算符（隐式）和交集运算符（&&）。并集运算符表示至少包含其某个操作数类中所有字符的类。交集运算符表示包含同时位于其两个操作数类中所有字符的类。

字符类运算符的优先级如下所示，按从最高到最低的顺序排列：

1	字面值转义	\x
2	分组	[…]
3	范围	a-z
4	并集	[a-e][i-u]
5	交集	[a-z&&[aeiou]]

注意，元字符的不同集合实际上位于字符类的内部，而非字符类的外部。例如，正则表达式 . 在字符类内部就失去了其特殊意义，而表达式 - 变成了形成元字符的范围。

行结束符

行结束符 是一个或两个字符的序列，标记输入字符序列的行结尾。以下代码被识别为行结束符：

- 新行（换行）符（'\n'）、
- 后面紧跟新行符的回车符（"\r\n"）、
- 单独的回车符（'\r'）、
- 下一行字符（'\u0085'）、
- 行分隔符（'\u2028'）或
- 段落分隔符（'\u2029）。

如果激活 UNIX_LINES 模式，则新行符是唯一识别的行结束符。

如果未指定 DOTALL 标志，则正则表达式 . 可以与任何字符（行结束符除外）匹配。

默认情况下，正则表达式 ^ 和 $ 忽略行结束符，仅分别与整个输入序列的开头和结尾匹配。如果激活 MULTILINE 模式，则 ^ 在输入的开头和行结束符之后（输入的结尾）才发生匹配。处于 MULTILINE 模式中时，$ 仅在行结束符之前或输入序列的结尾处匹配。

组和捕获

捕获组可以通过从左到右计算其开括号来编号。例如，在表达式 ((A)(B(C))) 中，存在四个这样的组：

1	((A)(B(C))))
2	\A
3	(B(C))

4 (C)
组零始终代表整个表达式。

之所以这样命名捕获组是因为在匹配中，保存了与这些组匹配的输入序列的每个子序列。捕获的子序列稍后可以通过 Back 引用在表达式中使用，也可以在匹配操作完成后从匹配器获取。

与组关联的捕获输入始终是与组最近匹配的子序列。如果由于量化的缘故再次计算了组，则在第二次计算失败时将保留其以前捕获的值（如果有的话）例如，将字符串 "aba" 与表达式 (a(b)?)+ 相匹配，会将第二组设置为 "b"。在每个匹配的开头，所有捕获的输入都会被丢弃。

以 (?) 开头的组是纯的*非捕获* 组，它不捕获文本，也不针对组合计进行计数。

Unicode 支持

此类符合 *Unicode Technical Standard #18:Unicode Regular Expression Guidelines* 第 1 级和 RL2.1 Canonical Equivalents。

Java 源代码中的 Unicode 转义序列（如 \u2014）是按照 Java Language Specification 的 第 3.3 节中的描述处理的。这样的转义序列还可以由正则表达式解析器直接实现，以便在从文件或键盘击键读取的表达式中使用 Unicode 转义。因此，可以将不相等的字符串 "\u2014" 和 "\\u2014" 编译为相同的模式，从而与带有十六进制值 0x2014 的字符匹配。

与 Perl 中一样，Unicode 块和类别是使用 \p 和 \P 构造编写的。如果输入具有属性 *prop*，则与 \p{*prop*} 匹配，而输入具有该属性时与 \P{*prop*} 不匹配。块使用前缀 In 指定，与在 InMongolian 中一样。可以使用可选前缀 Is 指定类别：\p{L} 和 \p{IsL} 都表示 Unicode 字母的类别。块和类别在字符类的内部和外部都可以使用。

受支持的类别是由 Character 类指定版本中的 *The Unicode Standard* 的类别。类别名称是在 Standard 中定义的，即标准又丰富。Pattern 所支持的块名称是 UnicodeBlock.forName 所接受和定义的有效块名称。

行为类似 java.lang.Character boolean 是 *methodname* 方法（废弃的类别除外）的类别，可以通过相同的 \p{*prop*} 语法来提供，其中指定的属性具有名称 java*methodname*。

与 Perl 5 相比较

Pattern 引擎用有序替换项执行传统上基于 NFA 的匹配，与 Perl 5 中进行的相同。

此类不支持 Perl 构造：

- 条件构造 (?{*X*}) 和 (?(*condition*)*X*|*Y*)、
- 嵌入式代码构造 (?{*code*}) 和 (??{*code*})、
- 嵌入式注释语法 (?#comment) 和

- 预处理操作 \l \u、\L 和 \U。

此类支持但 Perl 不支持的构造：

- Possessive 数量词，它可以尽可能多地进行匹配，即使这样做导致所有匹配都成功时也如此。
- 字符类并集和交集，如上文所述。

与 Perl 的显著不同点是：

- 在 Perl 中，\1 到 \9 始终被解释为 Back 引用；如果至少存在多个子表达式，则大于 9 的反斜线转义数按 Back 引用对待，否则在可能的情况下，它将被解释为八进制转义。在此类中，八进制转义必须始终以零开头。在此类中，\1 到 \9 始终被解释为 Back 引用，较大的数被接受为 Back 引用，如果在正则表达式中至少存在多个子表达式的话；否则，解析器将删除数字，直到该数小于等于组的现有数或者其为一个数字。
- Perl 使用 g 标志请求恢复最后匹配丢失的匹配。此功能是由 Matcher 类显式提供的：重复执行 find 方法调用可以恢复丢失的最后匹配，除非匹配器被重置。
- 在 Perl 中，位于表达式顶级的嵌入式标记对整个表达式都有影响。在此类中，嵌入式标志始终在它们出现的时候才起作用，不管它们位于顶级还是组中；在后一种情况下，与在 Perl 中类似，标志在组的结尾处还原。
- Perl 允许错误匹配构造，如在表达式 *a 中，以及不匹配的括号，如在在表达式 abc] 中，并将其作为字面值对待。此类还接受不匹配的括号，但对 +、? 和 * 不匹配元字符有严格限制；如果遇到它们，则抛出 PatternSyntaxException。

有关正则表达式构造行为更准确的描述，请参见 *Mastering Regular Expressions, 2nd Edition*，该书由 Jeffrey E. F. Friedl、O'Reilly 和 Associates 合著，于 2002 年出版。