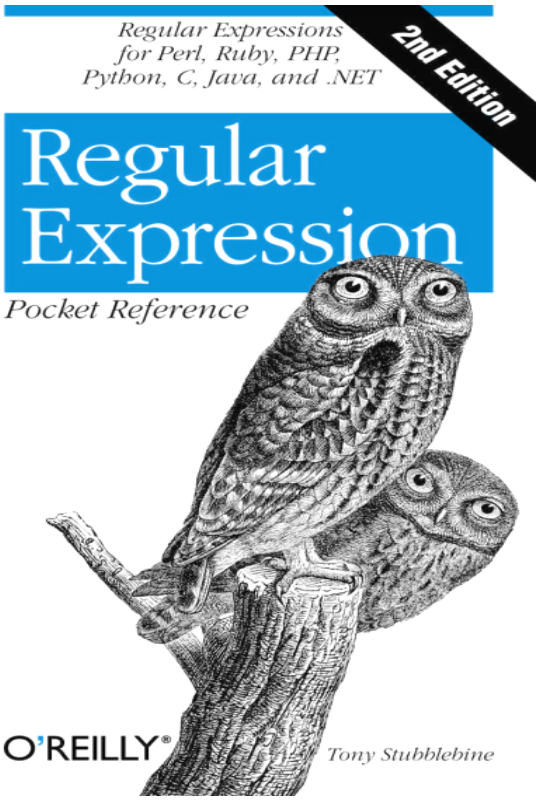
正则表达式袖珍手册



Tony Stubblebine 著

张桂权 译

ajax.mailer@gmail.com

17/11/2007

目　录

正则表达式袖珍手册

正则表达式是一种专门用来解析和操纵文本的语言。正则表达式通常被用来处理复杂的搜索和替换操作，以及验证文本数据的格式。

如今，正则表达式已经被包括在绝大数的编程语言，以及很多脚本语言、编辑器、应用、数据库和命令行工具中。本书的目的是提供一个快速接触绝大多数流行语言正则表达式语法和模式匹配操作的机会，这样你就可以把正则表达式的知识应用到任何环境中。

本书的第二版增加了有关 Ruby、Apache web server 和通用正则表达式部分，同时更新了已有语言的相关章节。

关于本书

本书从正则表达式的一般介绍开始。第一部分描述和定义正则表达式中用到的结构，并建立了模式匹配的通用原则。剩下的部分主要介绍不同语言实现的正则表达式的语法、特性和使用。

本书中实现部分包括Perl、Java™、NET、 C#、 Ruby、Python、PCRE、PHP、Apache web server、 *vi* 编辑器、JavaScript和shell工具。

本书所用的约定

在本书中我们遵从以下列出来的几个约定：

斜体
: 重点、新词汇、程序名和 URL 采用*斜体*

等宽字体
: 可选项、值、代码段以及任何需要逐字输入的文本

等宽斜体
: 需要替换为用户指定的值的文本

等宽粗体
: 命令实例或任何需要用户逐字输入的文本

致谢

Jeffrey E. F. Friedl著的《精通正则表达式》(*Mastering Regular Expressions* (O'Reilly))是正则表达式的权威之作。在编写本书的时候，我基本上依赖于本书和作者的建议。为了方便对正则表达式语法和概念的进一步讨论，本书提供了《精通正则表达式》第三版（MRE）的页码引用。

Nat Torkington和Linda Mui是两位非常优秀的编辑，他们基于第一版中出现的束手问题给了我很多指导。本版的编辑受益于Andy Oram的优秀编辑技术。特别感谢Sarah Burcham把撰写本书的机会给予了我，以及她对“Shell 工具”一章做出的贡献。感谢来自 Philip Hazel, Steve Friedl, Ola Bini, Ian Darwin, Zak Greant, RonHitchens, A.M. Kuchling, Tim Allwine, Schuyler Erle, David Lents, Rabble, Rich Bowan, Eric Eisenhart, and Brad Merrill 的书写和技术评审。

正则表达式和模式匹配简介

一个正则表达式*(regular expression)*就是包含正常字符串和特殊元字符（*metacharacters*）或元序列(*metasequences*)的字符串。正常字符串匹配它们自己。元字符和元序列是字符或表示数量、位置或字符类型的字符序列。“正则表达式元字符、模式和结构”列表，展示了正则表达式中最通用的元字符和元序列。后面的章节中将列出正则表达式的特殊实现所支持的有效的元字符和它的语法。

模式匹配包括搜索由正则表达式匹配的文本一部分。搜索这个文本的核心代码叫正则表达式引擎（*regular expression engine*）。只要记住以下两个规则，你就可以推出绝大多数的正则表达式的结果：

1. 最早匹配成功（最左边的）
   正则表达式应用到输入的时候是从第一个字符，并一次处理到最后一个字符。一旦正则表达式引擎找到一个匹配，就立即返回（请参看 MRE 148-149）。

2. 标准计量器是贪婪的
   计量器指定一个对象（字符、字符串）的最大可重复次数。标准的计量器千方百计想匹配所有可能的次数。如果对于匹配成功有必要那么可以将其值设置为比最大值小的数。遗漏字符和试图少贪婪匹配的处理过程叫回溯（backtracking，请参看 MRE 151-153）。

不同类型的正则表达式有各自不相同的特点。通常由两种类型的引擎：确定有限自动机（Deterministic Finite Automaton, DFA）和非确定有限自动机（Nondeterministic Finite Automaton, NFA）。DFA 速度快，但是缺了 NFA 的许多特性，比如，捕获、环顾四周、非贪婪计量器。NFA 有传统实现和 POSIX 两类。

DFA 引擎

DFAs 用输入字符串的每一个字符依次和正则表达式比较，并记录这个过程中所有的匹配。由于每一字符最多只检测一次，所以 DFA 引擎的速度最快。DFAs 的一个传统的记忆方法是它的交换是贪婪。当在一次匹配中有不止一个可选（foo|foobar）时，选择最长的一个。所以规则 1 可以记作“最长最左匹配成功。”（请参看 MRE 155-156）

传统的 NFA 引擎

传统的 NFA 引擎用正则表达式的每一个元素跟输入字符串进行比较，并记录正则表达式中所选择的两个选项的位置。如果一个选择失败，引擎回溯到最近保存的位置。在标准计量器的作用下，引擎一般采用贪婪的选项来匹配更多的文本。然而，如果这个选项导致匹配的失败，那么引擎会返回到一个保存的位置，并尝试低贪婪度的路径。传统的 NFA 引擎采用顺序交换，这样交换中的每一个选项都依次进行尝试。如果前面已经找到了一个成功的匹配，那么很长的匹配可能会被忽略。所以，规则 1 可以称作去读“贪婪量器满足条件之后的最先最左匹配”。（请参看 MRE 153-154）

POSIX NFA 引擎

OPSIX NFA引擎的原理和传统的NFA相似，主要的一个差别是：POSIX引擎总是选择最长最左匹配。比如，交换cat | category，即使第一个交换（“cat”）已经匹配成功，而且在前面出现过，如果可能的话，仍将匹配“category”整个字。（请参看MRE 153-154）

正则表达式元字符、模式和结构

这里所讲的元字符和元序列表示正则表达式结构和通用语法的可用类型。然而，不同实现之间的语法和可用性差异很大。

字符表示

很多实现提供了难输入字符的快捷表达字符。（请参看MRE115-118）

字符速记

很多正则表达式实现为alert（告警）, backspace（空白）, escape character（换码符）, form feed（换页符）, newline（换行）,carriage return（回车）, horizontal tab（水平制表符）和vertical tab characters（垂直制表符）指定了速记字符。例如，\n是换行符的速记，通常是LF（012，八进制），但是有时可能是CR（015，八进制），依赖于操作系统。比较混乱的是，很多实现中同时用\b表示空格和字边界（“字”字符和非字字符之间的位置）。对于这些实现来说，在字符类（字符串匹配中可能的字符的集合）中表示空格，其余地方表示字边界。

八进制转义：\num

用两个或三个八进制数字表示一个字符。比如\015\0112匹配ASCII中的CR／LF字符序列。

十六进制和Unicode转义：\xnum, \x{num}, \unum, \Unum

用十六进制数表示字符。四位数字和更大的数可以表示Unicode字符范围。比如，\x0D\x0A匹配ASCII的CR／LF字符序列。

控制字符：\cchar

由于ASCII控制字符用小于32的数值编码。为了安全，通常用一个大写的char——有些实现不处理小写表示。例如，\cH匹配Control-H，一个ASCII控制字符。

字符类和类似（class-like）的结构

字符类用来指定自己的集合。一个字符类匹配输入字符串的中字符集定义中的单一字符。（请参看 MRE 118-128）

普通类：[…]和[^...]

字符类，[…]，和求反字符类[^…]，允许你列出你想或不想匹配的字符。一个字符类通

常匹配一个字符。-（破折号）描述字符的范围。例如，[a-z]匹配 ASCII 中任意一个小写字符。把破折号包括在字符列中，要么首先列出它，要么回避它。

任意字符：dot（.），点号

通常匹配换行之外的任意字符。但是匹配的模式可以改变，所以点号也可以匹配换行。在字符集内，点号就是一个点号罢了。

类速记：\w，\d，\s，\W，\D，\S

通常提供字字符，数字和空格字符类的速记。一个字字符通常包括 ASCII 字母数字和下划线（underscore）。不过，依据不同的实现，字母数字表可以包括添加地区或 Unicode 字母数字。小写速记（比如，\s）匹配类中的一个字符。大写速记（比如，\S）匹配不在类中的一个字符。例如，\d 匹配一个数字字符，通常等效于[0-9]。

POSIX 字符类：[:alnum:]

POSIX 定义了一些只能在正则表达式中使用的字符类（参照表 1）。比如[:lower:]。当写作[[:lower:]]时等效于 ASCII 的[a-z]。

POSIX 字符类	
类名	类描述
Alnum	字母和数字
Alpha	字母
Blank	仅表示空格或制表符
Cntrl	控制字符
Digit	十进制数
Graph	打印字符，不包含空格
Lower	小写字母
Print	打印字符，包含空格
Punct	打印字符，不包含字母和数字
Space	空白
Upper	大写字母
Xdigit	十六进制数

表 1 POSIX 字符类

Unicode 属性、字体和区位：\p{prop}，\P{prop}

Unicode 标准定义的字符类拥有特殊的属性，所属的字体和存在区位。属性是字符定义的特性，比如，是一个字母还是数字（参看表 2）。字体就是书写系统，比如 Hebrew、Latin 或 Han。区位就是字符在 Unicode 字符上的范围映射。有些字符要求在 Unicode 属性上加前缀 Is 或 In。比如，\p{Ll}匹配支持 Unicode 编码的语言的小写字母，比如，a 或 α 。

Unicode 联合字符：\X

匹配 Unicode 字符，后续任意数量的 Unicode 联合字符。这是\P{M}\p{M}的速记。例如，\X 匹配è或 e'，以及这两个字符。

标准 Unicode 属性	
属性名	属性描述
\p{L}	字母
\p{Ll}	小写字母
\p{Lm}	修饰字母
\p{Lo}	字母和其他。不区分大小写，不是修饰字母
\p{Lt}	标题字母（Ttitlecase）
\p{Lu}	大写字母

表 2 标准 Unicode 属性

标准 Unicode 属性（续）	
\p{Cc}	ASCII 和 Latin-1控制字符
\p{C}	不在其他类中的控制码和字符
\p{Cf}	非可视格式化字符
\p{Cn}	未指定的码点
\p{Co}	私人使用，比如公司的 Logo
\p{Cs}	替代
\p{M}	连接基本字符的符号，比如重音符号
\p{Mc}	在自己的空间内变化的字符，实例包括元音信号（vowel signs）
\p{Me}	靠近字符的符号，比如园、正方形、菱形
\p{Mn}	改变其他字符的字符，比如重音（accents）和曲音（umlauts）
\p{N}	数字字符
\p{Nd}	不同字体中的十进制数字
\p{Nl}	表示数字的字母，比如随机数
\p{No}	上标（unperscropts）、符号（symbol）或表示数的非数字字符
\p{P}	标点（Punctuation）
\p{Pc}	连接标点，比如下划线
\p{Pe}	破折号和连字符（hyphens）
\p{Pi}	起始标点，比如开引号
\p{Pf}	结束标点，比如闭引号
\p{Po}	其他的标点
\p{Ps}	开标点，比如开括号
\p{S}	符号
\p{Sc}	货币符号
\p{Sk}	组合字符，不如专用字符（individual characters）
\p{Sm}	数学符号
\p{So}	其他符号
\p{Z}	非可视化表示的分隔符
\p{Zl}	行分隔符
\p{Zp}	段分隔符
\p{Zs}	空格符

锚和 0 宽断言

锚和“0 宽断言”匹配输入字符串的位置。（请参看 MRE128-134）

行或字符串的起始：^ ，\A

匹配被搜索字符串的起始位置。当在多行的情况下，^匹配后面的任意换行。有些实现支持\A，它仅匹配文本的起始位置。

行或字符串结尾：$，\Z，\z

$匹配字符串的结尾。在多行的情况下，$匹配前面的任意一个换行。当正则表达式的具体实现支持\Z 时，无论是否是匹配模式，它匹配字符串的结尾或字符串结尾换行之前的位置。有些实现同样支持\z,仅用来匹配字符串的结尾，无论是不是换行。

匹配的起始位置：\G

在重叠匹配中，\G 匹配前一个匹配结束的位置。通常，这个位置被设置为匹配失败的起始位置。

字边界：\b，\B，\<，\>

字边界元字符匹配一个字和紧挨的非字字符之间的位置。\b 通常指定一个字的边界位置，而，\B 指定非字符边界位置。有些实现提供\<和\>作为字起始和结束的分隔元序列。

向前：（?=...），（?!...）
向后：（?<=...），（?<!...）

向前的结构匹配文本中的一个位置，子模式可能在这里匹配（向前），可能不匹配（向后），或结束匹配（向后），或不可能结束匹配（向后）。例如，（?=bar）匹配 foobar 中的 foo，但是不匹配 food。在实现中通常限制子模式向后匹配的预定长度。

注释和模式变换

模式变化改变正则表达式引擎解释正则表达式的方式。（请参看 MRE 110-113，135-136）

多行模式：m

改变^和$的行为来匹配输入字符串中紧挨的换行。

单行模式：s

改变点号（dot）的行为来匹配所有字符，包括输入字符串中的换行。

大小写敏感模式：i

以大小写来唯一区分字母的不同。

自由调整间隔（Free-spacing）模式：x

允许正则表达式中包含空格和注释。空格和注释（以#开始，到行末）被正则表达式引擎忽略。

模式变换器：（?!），（?-!），（?mod：...）

一般情况下,可以通过（?mod）在正则表达式中设置模式变换器,它在后续的子表达式中生效。（?-mod）取消后续子表达式中的模式。（?mod：...）启动或取消逗号和关括号之间的模式。比如，使用（?：perl）匹配使用 perl，使用 Perl，使用 PeRl 等。

注释：（?#...）和＃

在自由调整间隔的模式下，＃表示这是一行注释。当不支持＃时，可以把（?#...）嵌入到正则表达式的任意位置，不论是什么模式。例如，.{0,80}（?#Field limit is 80）允许你就自己的改写原因做批注.{0,80}。

逐个文本跨度（Literal-text span）：\Q... ＼E

在\Q 和\E 之间回避元字符。例如，\Q（．*）\E 和\（\．*）等效。

分组、捕获、条件和控制

本节包括归类子模式、捕获子匹配、条件子匹配的语法和计算子模式匹配出现的次数。（请参看 MRE 137-142）

捕获和归类括号：（...）和\1，\2 等

括号起到两个作用：归类和捕获。括号中被子模式匹配的文本将在后面使用。通过从左到右的方式计算开括号得到捕获括号的个数。如果允许向后引用，那么可以通过\1，\2 等在同一个匹配中引用子匹配。对于捕获文本，可以通过依靠实现指定的方法来实现这种模式。例如，\b(\w+)\b\s+\1 匹配重复的字，不如 the the。

仅限于归类的括号：（?:…）

归类一个子正则表达式是为了变换或计量，而不是捕获一个子匹配。由于效率和高重用性，这一点非常有用。例如，(?:foobar)匹配 foobar，但是不把它保存到一个捕获组中。

命名捕获：（?<*name*>…）

进行捕获和归类，然后用 name 了引用捕获的文本。例如，Subject:(?<subject>.*)捕获 Subject 之后的文本，并进行捕获归类，最后通过 subject 来引用文本。

自动归类：(?>…)

即使导致匹配失败，也从不回溯组中匹配的文本。例如，（?>[ab]*）\w\w 匹配 aabbcc，担不是 aabbaa。

替换：… | …

允许测试几个子表达式。替换低优先级有时可能造成子表达式比预期的还要大很多，所以最好用括号指定你想要替换的内容。例如，\b(foo|bar)\b 匹配 foo 或 bar。

条件：（?（*if*）*then* |*else*）

If 依赖于具体的实现，不过通常是捕获子表达式或环顾(lookaround)的引用。而 *then* 和 *else* 都是正则表达式模式。当 *if* 条件为真时，引用 *then*，否则用 *else*。例如，(<)?foo(?(1)|(bar)) 匹配 foo 和 foobar。

贪婪计量器：*，+，?，{num, num}

用贪婪计量器来检测一个结构可以应用多少次。尝试进行所有的匹配，但是如果已经成功匹配则可以回溯或放弃匹配。例如,(ab)+匹配所有的 abababababab。

惰性计量器（Lazy quantifiers）：*?，+？，??，{num, num}?

惰性计量器控制一个结构可能应用的次数。但是不像贪婪计量器，它试图进行尽可能少的匹配次数。例如，(an)+?仅匹配一次 banana。

所有的计量器（Possessive quantifiers）：*+，++，?+，{num, num}+

所有的计量器像贪婪计量器，但是锁上（“lock in”）她的匹配，不允许后面分解子匹配的回溯。例如，(ab)++ab 不匹配 abababababab。

Unicode 支持

Unicode字符集给世界上所有语言的每一个字符分配唯一的数字。因为可能字符的数量实在太大了，所以Unicode要求用不止一个字节来表示一个字符。有些正则表达式的实现不支持Unicode字符，因为它们希望1个字节的ASCII字符。Unicode字符的基础支持，从逐字匹配一个Unicode字符串开始。高级支持包括字符集和具体化所有支持Unicode语言的其他的结构。例如，\W既可能匹配è，也可能匹配e。

通用正则表达式

本节包含简单的通用正则表达式模式。为了满足需要，你可能需要进行必要的调整。

这儿只给出了每一个表达式和它们匹配的目标字符串和不匹配的字符串。为了满足需要，你可能需要进行必要的调整。

采用 Perl 风格来书写：

/ 模式 / 形式

s/模式 / 替换 / 形式

诀窍

去掉前导和末尾的空格

s/^\s+//

s/\s+$//

匹配：“ foo bar ”和“foo ”

不匹配：“foo bar”

0 到 999999 之间的数字

/^\d{1,6}$/

匹配：42 和 678234

不匹配：10，000 （带分隔符的 1 万）

验证 HTML 十六进制编码

/^#([a-fA-F0-9]){3}(([a-fA-F0-9]){3})?$/

匹配：#fff、#1a1、#996633

不匹配：#ff 和 FFFFFF

美国社会安全码

/^\d{3}-\d{2}-\d{4}$/
匹配：078-05-1120
不匹配：078051120 和 1234-12-12

美国邮政编码

/^\d{5}(-\d{4})?$/
匹配：94941-3232 和 10024
不匹配：949413232

美国货币

/^\$\(d{1,3}(\,\d{3})*|\d+)(\.\d{2})?$/
匹配：$20 和$15,000.01
不匹配：$1.001 和$.99

匹配日期 MM/DD/YYYY HH:MM:SS

/^\d\d\/\d\d/\d\d\d\d \d\d:\d\d:\d\d$/
匹配：04/03/1978 20:45:38
不匹配：4/03/1978 20:45:38 和 4/3/78

前导路径名

/^.*\//
匹配：/usr/local/bin/apachectl
不匹配：c:\\System\foo.exe
（请参看 MRE 190-192）

点分 IP 地址

/^(\d|[01]?\d\d|2[0-4]\d|25[0-5])\ .
(\d|[01]?\d\d|2[0-4]\d|25[0-5])\ .
(\d|[01]?\d\d|2[0-4]\d|25[0-5])\ .
(\d|[01]?\d\d|2[0-4]\d|25[0-5])$ /

匹配：127.0.0.1 和 224.22.5.110
不匹配：127.1
（请参看 MRE 187-189）

MAC 地址

/^([0-9a-fA-F]{2}:){5}[0-9a-fA-f]{2}$/

匹配：01:23:45:67:89:ab
不匹配：01:23:45 和 0123456789ab

Email

/^[0-9a-zA-Z]([-.\w]*[0-9ª-zA-Z_+])*@([0-9a-zA-Z][-\w]*
[0-9a-zA]\.)+[a-zA-Z]{2,9}$/

匹配：tony@example.com、tomnu@i-e.com 和 tony@mail.example.museum
不匹配：.@example.com、tony@i-.com 和 tony@example.a
（请参看 MRE 70）

HTTP URL

/(http?):\ / \ /([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+
[a-zA-Z]{2,9})
(:\d{1,4})?([-\w\/#~:.?+=&%@~]*)/

匹配：https://example.com 和 http://foo.com:8080/bar.html
不匹配：ftp://foo.com 和 ftp://foo.com/

Perl 5.8

Perl 提供了正则表达式操作符、结构和特征的丰富的集合，而且每一个新版本都有增加。Perl 采用一个传统的 NFA 匹配引擎。为了探索 NFA 引擎背后的规则，请看“正则表达式和模式匹配简介”一节。

本章覆盖 Perl 5.8 版本。部分新的功能是在 Perl 5.10 中才引入的；包括表 8 中的内容。5.6 引入支持 Unicode 的功能，但是直到 5.8 才稳定。绝大多数的功能在 5.004 和之后的版本才有效。

支持的元字符

Perl 支持表 3 到表 7 所列的元字符和元序列。要想学习每一个元字符的扩展定义，请看“正则表达式元字符、模式和结构”一节。

Perl 字符表示	
序列名	序列描述
\a	告警（bell）
\b	空格；仅支持字符类中的空格（其余匹配一个字符边界）
\e	Esc 字符，x1B
\n	换行；Unix 和 Windows 上 x0A，MAC OS 9 上 x0D
\r	回车；Unix 和 Windows 上 x0D，MAC OS 9 上 x0A
\f	换页符，x0C
\t	水平制表符，x09
\octal	由三个或四个八进制数字指定的字符
\xhex	由一个或两个十六进制数字指定的字符
\cchar	命名的控制字符
\N{*name*}	Unicode 标准或 PATH_TO_PERLLIB/unicode/Names.txt 列出的命名字符；要求使用字符名':full'。

表 3 Perl 字符表示

Perl 字符类和类似的结构	
类名	类描述
[...]	列出来的或包含在列表范围的单一字符
[^...]	不在列出来的或不包含在列表范围的单一字符
[:class:]	POSIX 风格的字符类，仅在正则表达式字符类中有效
.	换行之外的任何字符（除非单行模式，/s）
\C	一个字节；但是这可能破坏 Unicode 字符流
\X	跟在任意数量 Unicode 字符之前的基本字符
\w	字字符，\p{IsWord}
\W	非字字符，\p{IsWord}
\d	数字字符，\p{IsDigit}
\D	非数字字符，\p{IsDigit}
\s	空格字符，\p{IsSpace}
\S	非空格字符，\p{IsSpace}
\p{prop}	包含在给定 Unicode 属性、字体和区位的字符
\P{prop}	不包含在给定 Unicode 属性、字体和区位的字符

表 4 Perl 字符类和类似的结构

Perl 锚和 0 宽测试	
序列名	序列描述
^	字符串的开头，或，在多行匹配模式（/m），任意换行之后的位置
\A	搜索字符串的开头，在所有匹配模式
$	搜索字符末尾或字符串末尾换行之前的位置，或在多行模式（/m），任意换行之后的位置
\Z	在任意匹配模式下，字符串末尾或字符串末尾换行之前的位置
\z	任意匹配模式下，字符串末尾位置
\G	当前搜索的开头
\b	字边界
\B	非字边界
(?=...)	正向前（Positive lookahead）
(?!...)	负向前（Negative lookahead）
(?<=...)	正向前；仅仅是锚长度
(?<!...)	负向前；仅仅是锚长度

表 5 Perl 锚和 0 宽测试

Perl 注释和模式变换器	
变换器	变换器描述
/i	大小写敏感匹配
/m	^和$匹配下一个内嵌的\n
/s	Dot(.)匹配换行
/x	忽略空格和允许在模式中使用注释（#）
/o	仅一次编译模式
（?*mode*）	为其余子表达式启动所列的模式（一个或多个 xsmi）
（?-*mode*）	为其余子表达式取消所列的模式（一个或多个 xsmi）
（?*mode*:...）	启动括号内所列的模式（一个或多个 xsmi）
（?-*mode*:...）	取消括号内所列的模式（一个或多个 xsmi）
（?#...）	把子字符串当作注释
#...	在/x模式中，把行的剩余部分当作注释
\u	把下一个字符强制转化为大写
\l	把下一个字符强制转化为小写
\U	把紧接的所有字符都强制转化为大写
\L	把紧接的所有字符都强制转化为小写
\Q	引掉紧接的所有正则表达式元字符
\E	由\U, \L, or \Q 开头的 span 的末尾

表 6 Perl 注释和模式变换器

Perl 归类、捕获、条件和控制	
序列	序列描述
(...)	把子模式和捕获子匹配归到\1，\2，...和$1，$2，...
\n	包含第 n 个被捕获的文本
(?:...)	把子模式分组，但是不捕获子匹配
(?>...)	自动分组
...|...	尝试子模式替换
*	匹配 0 或多次
+	匹配 1 次或多次
?	匹配 1 次或 0 次
{n}	匹配精确的 n 次
{n,}	至少匹配 n 次
{x,y}	至少匹配 x 次，最多 y 次
*?	匹配 0 次或多次，但是尽可能少
+?	匹配 1 次或多次，但是尽可能少
??	匹配 0 次或多次，但是尽可能少

表 7 Perl 归类、捕获、条件和控制

Perl 归类、捕获、条件和控制（续）	
序列	序列描述
{n,}?	至少匹配 n 次，但是尽可能少
{x,y}?	至少匹配 x 次，最多 y 次，但是尽可能少
(?(COND).../...)	匹配 if-then-else 模式，其中 COND 是一个会引或回顾断言的引用整数
(?(COND)...)	匹配 if-then 模式
(?{CODE})	执行内嵌的 Perl 代码
(??{CODE})	从内嵌的 Perl 代码中匹配正则表达式

表 7 Perl 归类、捕获、条件和控制（续）

Perl 5.10 的新功能	
(?<name>...)或(?’name’...)	命名的捕获组
\k<name>或\k’name’	命名捕获组的回引
%+	一个给定名字的最左捕获的 Hash 引用，$+{foo}
%-	一个给定名字的所有捕获的数组的 Hash 引用，$-{foo}[0]
\g{n}或\gn	回引第 n 个捕获
\g{-n}或\g-n	相对的第 n 个前捕获的回引
(?n)	递归第 n 个捕获 buffer
(?&name)	递归第 n 个命名捕获 buffer
(?R)	递归调用完整的表达式
(?(DEFINE)...)	定义一个可以递归的子表达式
(*FAIL)	子匹配失败，强制引擎回溯
(*ACCEPT)	强制引擎接受匹配，即使由更多的模式需要检验
(*PRUNE)	让引擎在当前的起点匹配失败
(*MARK:name)	标记和命名字符串的当前位置。这个位置对于$REGMARK 有效
(*THEN)	当回溯时，转到下一个替换
(*COMMIT)	当回溯时，导致匹配完全失败
/p	加强 ${^PREMATCH} 、 ${MATCH} 和${^POSTMATCH}的有效性的模式变换
\k	从最后的匹配中排除以前的匹配文本

表 8 Perl 5.10 的新功能

正则表达式操作符

Perl 提供了内建的正则表达式操作符 qr//、m//和 s///以及 spilt 函数。每一个操作符接受一个正则表达式模式字符串，通过字符串和变量插值之后，进行编译。

正则表达式一般由正斜杠（ / ）分隔，但是你可以选择任意非字母数字、非空格字符。例如：

qr#...#　　m!...!　　m{...}

s|...|...|　　s[...][...][...]　　s<...>/.../

由斜线（/.../）分隔的匹配，不要求一个前导 m：

/.../　　#和 m/.../ 等效

使用单引号作为分隔符来抑制给变量和\N{name}、\u、\l、\L、\Q 和\E 等结构插值。通常在它们是经过插值之后才传给正则表达式引擎。

qr// (Quote Regex) 引用正则表达式

qr/PATTERN/ismxo

像正则表达式一样引用和编译 PATTERN。返回值可能在后面的模式匹配或替换中使用。如果正则表达式需要重复的插值，那么这样做可以节省一定的时间。匹配模式（或 lack of）、/ismxo 被锁上。

m//(Matching)匹配

m/PATTERN/imsxocg

通过输入字符串来匹配 PATTERN。在这种情况下，返回一个由捕获括号匹配的子字符串的列表或成功匹配返回（1），匹配失败返回（）。在标量上下文中，如果成功返回 1，失败返回空（""）。/imsxo 是可选模式变换器。/cg 是可选匹配变换器。/g 在标量上下文中使匹配从前一个匹配的末尾开始。在列表上下文中，一个/g 匹配返回所有的匹配或返回所有匹配中捕获的子字符串。一个失败的/g 重置匹配的起始位置为字符串的开始，除非匹配属于联合的/cg 模式。

s/// (Substitution) 替换

s/PATTERN/REPLACEMENT/egimosx

在输入字符串中匹配 PATTERN，并用 REPLACEMENT 来替换匹配文本，返回成功的次数。/imosx 是可选的模式变换器。/g 替换 PATTERN 发生的位置的文本。每一个/e 作为 Perl 代码引发一次评价。

Split

split /PATTERN/, EXPR, LIMIT
split /PATTERN/, EXPR
split /PATTERN/
split

返回在 EXPR 中由匹配的 PATTERN 围绕的子字符串的列表。如果 LIMIT 被包含在其中，那么这个列表包括围绕第一个 LIMIT 匹配的子字符串。模式参数是一个匹配操作符，所以如果你想替换分隔符（e.g.,spilt m{PATTERN}），那么就用 m。匹配允许同一个变换器，如 m{}。表 9 列出了后匹配的变量。

Perl 后匹配变量	
变量名	变量描述
$1,$2,…	捕获的子匹配
@-	$-[0]：匹配起点的偏移；$-[n]：$n 起点的偏移
@+	$+[0]：匹配末尾的偏移；$+[n]：$n 末尾的偏移
$+	最后括号的匹配
$'	匹配前的文本。导致所有的正则表达式都变慢。和 substr($input,0,$-[0],])一样
$&	匹配的文本。导致所有的正则表达式都变慢。和 substr($input,$-[0], $+[0] - $-[0])一样
$`	匹配的文本。导致所有的正则表达式都变慢。和 substr($input, $+[0])一样
$^N	最靠近当前捕获括号的文本
$*	如果 true，所有的匹配都被假定为/m 而不是/s
$^R	一个模式匹配中最近执行代码结构的返回值

表 9 Perl 后匹配变量

Unicode 支持

Perl 提供了内建的 Unicode 3.2 的支持，包括完全支持\w、\d、\s 和\b 等元序列。

如果有局部定义那么以下的结构表示当前局部：大小写敏感(i)模式、\L、\l、\U、\u、\w 和\W.

Perl 支持表 3 中的标准 Unicode 属性和表 10 所示的 Perl 指定的组合属性，字体和属性可能有一个前缀 Is，但是并不是要求的。区位要求一个 In 前缀，当且仅当区位名和字体名冲突。

Perl 组合 Unicode 属性	
属性	等价值描述
IsASCII	[\x00-\x7f]
IsAlnum	[\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}]
IsAlpha	[\p{Ll}\p{Lu}\p{Lt}\p{Lo}]
IsCntrl	\p{C}
IsDigit	\p{Nd}
IsGraph	[^\p{C}\p{Space}]
IsLower	\p{Ll}
IsPrint	\P{C}
IsPunct	\p{P}
IsSpace	[\t\n\f\r\p{Z}]
IsUppper	[\p{Lu}\p{Lt}]
IsWord	[_\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}]
IsXDigit	[0-9a-fA-F]

表 10 Perl 组合 Unicode 属性

实例

实例 1.简单匹配

简单匹配实例 Perl 代码

```
# Find Spider-Man, Spiderman, SPIDER-MAN, etc.
my $dailybugle = "Spider-Man Menaces City!";
if ($dailybugle =~ m/spider[- ]?man/i) { do_something( ); }
```

实例 2.匹配、捕获组和 qr

匹配、捕获组和 qr 实例 Perl 代码

```
# Match dates formatted like MM/DD/YYYY, MM-DD-YY,...
my $date = "12/30/1969";
my $regex = qr!^(\d\d)[-/](\d\d)[-/](\d\d(?:\d\d)?)$!;
if ($date =~ $regex) {
print "Day= ", $1,
" Month=", $2,
" Year= ", $3;
}
```

实例 3. 简单替换

简单替换实例 Perl 代码

```
# Convert <br> to <br /> for XHTML compliance
my $text = "Hello World! <br>";
$text =~ s#<br>#<br />#ig;
```

实例 4. 高难度替换

高难度替换实例 Perl 代码

```
# urlify - turn URLs into HTML links
$text = "Check the web site, http://www.oreilly.com/catalog/
regexppr.";
$text =~
s{
\b # start at word boundary
( # capture to $1
(https?|telnet|gopher|file|wais|ftp) :
# resource and colon
[\w/#~:.?+=&%@!\-] +? # one or more valid
# characters
# but take as little as
# possible
)
(?= # lookahead
[.:?\-] * # for possible punctuation
(?: [^\w/#~:.?+=&%@!\-] # invalid character
| $ ) # or end of string
)
}{<a href="$1">$1</a>}igox;
```

其他资源

- O’Reilly 出版的 Larry Wall et al. 著作《Perl 编程》是 Perl 的标准手册。
- O’Reilly 出版的 Jeffrey E. F. Friedl 著作《精通正则表达式》第三版 283-364 页包括了 Perl 正则表达式的详细信息。
- 大多数 Perl 发行包中的 perlre 是 Perl 的帮助文档。

Java (java.util.regex)

在 Java1.4 中通过 Sun 的 java.util.regex 包来介绍正则表达式。虽然在以前的版本中有可与之媲美的包，但是现在 Sun 的是标准。Sun 提供的软件包采用的是传统的 NFA 匹配引擎。如果想进一步了解传统的 NFA 引擎背后的规则，请看“正则表达式和模式匹配”一节。本章包括 Java1.5 和 1.6 中的正则表达式。

支持的元字符

java.util.regex 支持表 11 到表 14 中列出来的元字符和元序列。关于每一个元字符的详述，请看“正则表达式元字符、模式和结构”一节。

Java 字符表示	
序列名	序列描述
\a	告警
\b	空格，\x08，只有在字符类中有效
\e	Esc 字符，\x1B
\n	换行，\x0A
\r	回车，\x0D
\f	分页，\x0C
\t	水平制表符(tab)，\x09
\0octal	通过 1、2 或 3 个八进制码数指定的字符
\xhex	通过 2 个十六进制数指定的字符
\uhex	通过 4 个十六进制数指定的 Unicode 字符
\cchar	命名的控制字符

表 11 Java 字符表示

Java 字符类和类似（class-like）结构	
字符类	类描述
[…]	列出来的或包含在列表范围的单一字符
[^…]	不在列出来的或不包含在列表范围的单一字符
.	除行终止（除非是 DOTALL 模式）之外的任意字符
\w	字字符，[a-zA-Z0-9_]
\W	非字字符，[^a-zA-Z0-9_]
\d	数字字符，[0-9]
\D	非数字字符，[^0-9]
\s	空格字符，[\t\n\f\r\x0B]
\S	非空格字符，[^ \t\n\f\r\x0B]
\p{prop}	包含在给定 POSIX 字符类，Unicode 属性和 Unicode 区位中的字符
\P{prop}	不包含在给定 POSIX 字符类，Unicode 属性和 Unicode 区位中的字符

表 12　Java 字符类和类似（class-like）结构

Java 锚和其他 0 宽测试	
序列名	序列描述
^	字符串的开头，或，在多行匹配模式（MULTILINE），任意换行之后的位置
\A	在任意匹配模式，搜索字符串的开头
$	字符串末尾，或在多行模式（MULTILINE），任意换行之前的位置
\Z	在任意匹配模式下，字符串末尾或字符串末尾换行之前的位置
\z	任意匹配模式下，字符串末尾
\b	字边界
\B	非字边界
\G	当前搜索的开头
(?=…)	正向前(Positive lookahead)
(?!…)	负向前(Negative lookahead)
(?<=…)	正向后(Positive lookbehind)
(?<!...)	负向后(Negative lookbehind)

表 13 Java 锚和其他 0 宽测试

Java 注释和模式转换器		
转换器/序列	模式字符	转换器描述
Pattern.UNIX_LINES	d	把\n 作为终止符
Pattern.DOTALL	s	点号匹配包括行终止符在内的任意字符
Pattern.MULTILINE	m	^和$匹配下一个内嵌的行终止符
Pattern.COMMENTS	x	忽略空格，并允许以#开头的注释
Pattern.CASE_INSENSITIVE	i	大小写不敏感的 ASCII 码字符匹配
Pattern.UNICODE_CASE	u	大小写不敏感的 UNICODE 码字符匹配
Pattern.CANON_EQ		Unicode“canonical equivalence”模式，其中基础字符中的字符，或序列和可视化表示的联合字符视为相等
(?*mode*)		为其余子表达式启动所列的模式（一个或多个 idmsux）
(?-*mode*)		为其余子表达式取消所列的模式（一个或多个 idmsux）
(?*mode*:...)		启动括号内所列的模式（一个或多个 idmsux）
(?-*mode*:...)		取消括号内所列的模式（一个或多个 idmsux）
#...		在/x 模式中，把行内剩余部分当作注释

表 14 Java 注释和模式转换器

Java 归组、捕获、条件和控制	
序列	序列描述
(...)	把子模式和捕获子匹配归到\1，\2，...和$1，$2，...
\n	包含第 n 个被捕获的文本
$n	在替换字符串中，包含第 n 个捕获组中匹配的文本
(?:...)	把子模式分组，但是不捕获子匹配
(?>...)	自动分组
...|...	尝试子模式替换
*	匹配 0 或多次
+	匹配 1 次或多次
?	匹配 1 次或 0 次
{n}	匹配精确的 n 次
{n,}	至少匹配 n 次
{x,y}	至少匹配 x 次，最多 y 次
*?	匹配 0 次或多次，但是尽可能少
+?	匹配 1 次或多次，但是尽可能少
??	匹配 0 次或多次，但是尽可能少

表 15 Java 归组、捕获、条件和控制

Java 归组、捕获、条件和控制（续）	
序列	序列描述
{n,}?	至少匹配 n 次，但是尽可能少
{x,y}?	至少匹配 x 次，最多 y 次，但是尽可能少
*+	匹配 0 此或多次，并且从不回溯
++	匹配 1 此或多次，并且从不回溯
?+	匹配 0 此或 1 次，并且从不回溯
{n}+	至少匹配 n 次，并且从不回溯
{n,}+	至少匹配 n 次，并且从不回溯
{x,y}+	至少匹配 x 次，最多匹配 y 次，并且从不回溯

表 15 Java 归组、捕获、条件和控制（续）

正则表达式类和接口

正则表达式主要包含在两个类，java.util.regex.Pattern 和 java.util.regex.Matcher；一个异常类，java.util.regex.PatternSyntaxException；和一个接口，CharSequence。而且 String 类实现了 CharSequence 的接口，来提供基础的模式匹配方法。编译之后的 Pattern 对象，可以适用于任意的 CharSequcene。一个 Matcher 是一个可以扫面模式发生一次或多次的全状态对象，适用于 String（或实现 ChaeSequcene 的任意对象）。

在正则表达式String中避免使用反斜杠。所以，\n（换行）在Java正则表达式中用\\n表示。

java.lang.String

描述

模式匹配方法。

方法

boolean matches(String *regex*)

如果 regex 匹配整个字符串则返回 true。

string[] split(String *regex*)

返回一个 regex 边缘的子字符串的数组。

String[] split(String *regex*, int *limits*)

返回一个 regex 第 limit-1 个匹配边缘的子字符串的数组。

String replaceFirst(String *regex*, String *replacement*)

用 *replacement* 替换由 *regex* 第一此匹配的子字符串。

String replaceAll(String *regex*, String *replacement*)

用 *replacement* 替换 *regex* 匹配的所有子字符串。

java.util.regex.Pattern

描述

正则表达式模式的模型。

方法

static Pattern compile(String *regex*)

从 regex 中构建一个 Pattern 对象。

static Pattern compile(String *regex*, int *flags*)

从 regex 和 OR'd 模式转换器常量 *flags* 中构建一个 Pattern 对象。

Int flags()

返回 Pattern 的模式转换器的个数。

Matcher matcher(CharSequence *input*)

构建一个使 Pattren 匹配 *input* 的 Matcher 对象。

static boolean matches(String *regex*, CharSequence *input*)

如果 *regex* 匹配整个 *input* 字符串，则返回 true。

String Pattern()

返回创建这个 Pattern 的正则表达式。

static String quote(String *text*)

转义 *text*，这样操作符就可以逐字匹配了。

String[] split(CharSequence *input*)

返回一个 Pattern 在 *input* 中的匹配周围的子字符的数组。

String[] split(CharSequence input, int limit)

返回在字符串 *input* 中 *limit* 首次匹配的 Pattern 周围的子字符串的一个数组。

java.util.regex.Matcher

描述

模型化一个正则表达式匹配器和模式匹配结果。

方法

Matcher appendReplacement(StringBuffer *sb*, String *replacement*)

在匹配和 *replacement* 之前向 *sb* 填充子字符串。

StringBuffer appendTial(StringBuffer *sb*)

在匹配之后想 *sb* 填充子字符串。

int end()

匹配之后的第一个字符的索引。

int end(int *group*)

由 *group* 捕获的文本之后第一个字符的索引。

boolean find()

查找输入字符串中下一个匹配。

boolean find(int *start*)

查找 *start* 之后的下一个匹配。

String group()

被此模式匹配的文本。

String group(int *group*)

由捕获组 *group* 捕获的文本。

int groupCount()

Pattern 在那个捕获组的个数。

boolean hasAnchoringBounds()

如果 Matcher 采用锚边界，所以锚操作在区域边界上匹配成功，则返回 trus，而不仅在目标字符串的开头和结尾。

boolean hasTransparentBounds()

如果 Matcher 采用透明边界，所以回环（lookaround）操作在当前搜索边界之外可见，则结果为 true。默认为 false。

boolean hitEnd()

如果最后一个匹配试图检查超过输入末尾时返回 true。多个输入可能产生一个很长的匹配的标志。

boolean lookingAt()

如果模式匹配输入的开头则返回 true。

boolean matches()

如果 Pattern 匹配了整个输入字符串则返回 true。

Pattern pattern()

返回被 Matcher 用的 Pattern 对象。

static String quoteReplacement(String *string*)

在替换中转移特殊字符。

Matcher region(int *start*, int *end*)

返回匹配（matcher），并在字符串中从开头的 *start* 到 *end* 之间执行进一步的匹配。

int regionStart()

返回搜索区域的起始偏移量。默认为 0。

int regionEnd()

返回搜索区域的末尾偏移量。默认为到目标字符串的长度。

String replaceAll(String *replacement*)

用 *replacement* 来替换所有的匹配字符串。

String replaceFirst(String *replacement*)

用 *replacement* 替换第一个匹配。

boolean requireEnd()

如果最后一个匹配的成功依赖于输入的末尾则返回 true。在扫描过程中，这是多个输入可能已经导致失败的一个标致。

Matcher reset()

重置匹配（matcher），所以下一个匹配从输入字符串的起点开始。

int start()

被匹配的第一个字符的索引。

int start(int *group*)

在被捕获的子字符串 *group* 中第一个被匹配的字符的索引。

MatchResult toMatchResult()

为最近的匹配的一个 MatchResult 对象。

String toString()

为了调试返回一个匹配的字符串表示。

Matcher useAnchorBounds(boolean *b*)

如果为 true 则设置 Matcher 采用锚边界，这样锚操作从当前搜索边界的开头和结尾进行匹配，不是字符串的开始和结尾。默认为 true。

Matcher usePattern(Pattern *p*)

替换 Matcher 的模式，但是保持其他部分的匹配状态。

Matcher useTransparentBounds(boolean *b*)

如果为 true 则设置 Matcher 采用透明边界，这样等回环（lookaround）在当前搜索边界之外可见。默认为 false。

java.util.regex.PatternSyntaxException

描述

抛出异常表明正则表达式中有语法错误。

方法

PatternSyntaxException(String *desc*, String *regex*, int *index*)

构造本类的一个实例。

String getDescription()

返回错误的描述。

int getIndex()

返回错误的索引。

String getMessage()

返回包括错误描述、索引、正则表达式模式和模式中错误发生的位置的描述的多行错误消息。

String getPattern()

返回抛出异常的正则表达式模式。

java.lang.CharSequence

描述

定义只读的接口，这样正则表达式就可以使用于字符。

方法

char charAt(int *index*)

返回基于 0 位置 *index* 的字符。

int length()

返回序列中字符的个数。

CharSequence subSequence(int *start*, int *end*)

返回一个包括 *start* 索引而不包括 *end* 索引的子序列。

String toString()

返回序列的字符串表示。

Unicode 支持

这个包支持 Unicode4.0，但是\w、\W、\d、\D、\s 和\S 仅支持 ASCII。你可以使用等效的 Unicode 属性\p{L}、\P{L}、\p{Nd}、\P{Nd}、\p{Z}和\p{z}。字符边界序列\b 和\B 之别 Unicode。

请看表 2，了解支持的 Unicode 属性和区位。这个包只支持简短的属性名，比如\p{Lu}，而不支持\p{Lowercase_Letter}。区位的名字要求以 In 为前缀，仅支持没有空格和下划线的名字，例如，\p{InGreekExtended}，而不支持\p{In_Greek_Extended}或\p{InGreek Extended}。

实例

实例 5 简单匹配

简单匹配

```
// Find Spider-Man, Spiderman, SPIDER-MAN, etc.

    public class StringRegexTest {
        public static void main(String[] args) throws Exception {
```

```
        String dailyBugle = "Spider-Man Menaces City!";
        //regex must match entire string
        String regex = "(?i).*spider[- ]?man.*";
            if (dailyBugle.matches(regex)) {
            System.out.println("Matched: " + dailyBugle);
            }
        }
    }
```

实例 6 匹配和捕获组

匹配和捕获组

```
// Match dates formatted like MM/DD/YYYY, MM-DD-YY,...
import java.util.regex.*;
public class MatchTest {
    public static void main(String[] args) throws Exception {
        String date = "12/30/1969";
        Pattern p =
        Pattern.compile("^(\\d\\d)[-/](\\d\\d)[-/](\\d\\d(?:\\d\\d)?)$");
        Matcher m = p.matcher(date);
        if (m.find( )) {
            String month = m.group(1);
            String day = m.group(2);
            String year = m.group(3);
            System.out.printf("Found %s-%s-%s\n", year, month, day);
        }
    }
}
```

注意：由于 JDK1.6 不支持 System.out.printf（sting, string, string, string）方法，所以需要适当的修改才能编译通过。修改后的代码如下：

匹配和捕获组（更改后的代码）

```
import java.util.regex.*;
public class MatchTest {
    public static void main(String[] args) throws Exception {
        String date = "12/30/1969";
        String[] result;
        Pattern p = Pattern.compile("^(\\d\\d)[-/](\\d\\d)[-/](\\d\\d(?:\\d\\d)?)$");
        Matcher m = p.matcher(date);
        if (m.find()) {
            String month = m.group(1);
```

```
                String day = m.group(2);
                String year = m.group(3);

                result = new String[3];
                result[0] = year;
                result[1] = month;
                result[2] = day;
                // System.out.printf("Found %s-%s-%s\n", year, month, day);
                System.out.printf("Found %s-%s-%s\n", result);
            }
        }
}
```

实例 7 简单替换

简单替换

```
/Example -. Simple substitution
//Convert <br> to <br /> for XHTML compliance
import java.util.regex.*;

public class SimpleSubstitutionTest {
public static void main(String[] args) {
        String text = "Hello world. <br>";
        Pattern p = Pattern.compile("<br>", Pattern.CASE_INSENSITIVE);
        Matcher m = p.matcher(text);
        String result = m.replaceAll("<br />");
        System.out.println(result);
        }
}
```

实例 8 高难度替换

高难度替换

```
  import java.util.regex.*;

  public class Urlify {
        public static void main (String[ ] args) throws Exception {
        String text = "Check the web site, http://www.oreilly.com/
        catalog/regexppr.";
        String regex =
        "\\b # start at word\n"
        + " # boundary\n"
```

```
        + "( # capture to $1\n"
        + "(https?|telnet|gopher|file|wais|ftp) : \n"
        + " # resource and colon\n"
        + "[\\w/\\#~:.?+=&%@!\\-] +? # one or more valid\n"
        + " # characters\n"
        + " # but take as little\n"
        + " # as possible\n"
        + ")\n"
        + "(?= # lookahead\n"
        + "[.:?\\-] * # for possible punc\n"
        + "(?: [^\\w/\\#~:.?+=&%@!\\-] # invalid character\n"
        + "| $ ) # or end of string\n"
        + ")";
        Pattern p = Pattern.compile(regex,
        Pattern.CASE_INSENSITIVE + Pattern.COMMENTS);
        Matcher m = p.matcher(text);
        String result = m.replaceAll("<a href=\"$1\">$1</a>");
        System.out.println(result);
    }
}
```

其他资源

- O'Reilly 出版的 Jeffrey E. F. Friedl 著作《精通正则表达式》第三版 365-403 页包括了 Java 正则表达式的详细信息。
- Sun 在线帮助文档 http://java.sun.com/javase/6/docs/api/java/util/regex/package-summary.html

.NET and C#

微软的.NET 框架为.NET 的所有实现提供了一个一致而功能强大的正则表达式类的集合。以下章节罗列了.NET 正则表达式的语法，核心的.NET 类，和 C#实例。微软的.NET 采用了传统的 NFA 匹配引擎。如果想进一步了解传统的 NFA 引擎背后的规则，请看“正则表达式和模式匹配”一节。

支持的元字符

.NET 支持表 16 到表 21 中列出来的元字符和元序列。关于每一个元字符的详述，请看“正则表达式元字符、模式和结构”一节。

.NET 字符表示	
序列名	序列描述
\a	告警
\b	空格，\x08，只有在字符类中有效
\e	Esc 字符，\x1B
\n	换行，\x0A
\r	回车，\x0D
\f	分页，\x0C
\t	水平制表符(tab)，\x09
\v	垂直制表符（tab），\x0B
\0octal	通过 1、2 或 3 个八进制码数指定的字符
\xhex	通过 2 个十六进制数指定的字符
\uhex	通过 4 个十六进制数指定的 Unicode 字符
\cchar	命名的控制字符

表 16 .NET 字符表示

.NET 字符类和类似（class-like）结构	
字符类	类描述
[…]	列出来的或包含在列表范围的单一字符
[^…]	不在列出来的或不包含在列表范围的单一字符
.	除行终止（除非是单行模式，s）之外的任意字符
\w	字字符，[\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}\p{Pc}]或 ECMAScript 模式中[a-zA-Z_0-9]
\W	非字字符，[\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}\p{Pc}]或 ECMAScript 模式中[^a-zA-Z_0-9]
\d	数字字符，\p{Nd}或 ECMAScript 模式中[0-9]
\D	非数字字符，\p{Nd}或 ECMAScript 模式中[^0-9]
\s	空格字符，[\f\n\r\t\v\x85]或 ECMAScript 模式中[\f\n\r\t\v]
\S	非空格字符，[^ \f\n\r\t\v\x85]或 ECMAScript 模式中[^ \f\n\r\t\v]
\p{prop}	包含在给定 POSIX 字符类，Unicode 属性和 Unicode 区位中的字符
\P{prop}	不包含在给定 POSIX 字符类，Unicode 属性和 Unicode 区位中的字符

表 17　.NET 字符类和类似（class-like）结构

.NET 锚和其他 0 宽测试	
序列名	序列描述
^	字符串的开头，或，在多行匹配模式（MULTILINE），任意换行之后的位置
\A	在任意匹配模式，搜索字符串的开头
$	字符串末尾，或在多行模式（MULTILINE），任意换行之前的位置
\Z	在任意匹配模式下，字符串末尾或字符串末尾换行之前的位置
\z	任意匹配模式下，字符串末尾
\b	\w 和\W 之间的字边界
\B	非字边界
\G	前一个匹配的末尾
(?=…)	正向前(Positive lookahead)
(?!…)	负向前(Negative lookahead)
(?<=…)	正向后(Positive lookbehind)
(?<!...)	负向后(Negative lookbehind)

表 18 .NET 锚和其他 0 宽测试

.NET 注释和模式转换器		
转换器/序列	模式字符	转换器描述
Singleline	s	点号匹配包括行终止符在内的任意字符
Multiline	m	^和$匹配下一个内嵌的行终止符
IgnorePatternWhitespace	x	忽略空格，并允许以#开头的注释
IgnoreCase	i	在当前文化（current culture）中大小写不敏感匹配
CultureInvariant	i	在当前文化（current culture）大小写不敏感码字符匹配
ExplicitCapture	n	允许命名的捕获组，但是把括号当作非捕获组
Compiled		编译正则表达式
RightToLeft		从右向左搜索，起点位置的左边开始。具有未定义和不可预测的语义
ECMAScript		加强 IgnoreCase 或 Multiline 中的 ECMAScript 编译
(?imnsx-imnsx)		为模式剩下的部分启动或关闭匹配标识
(?imnsx-imnsx:...)		为子表达式剩下的部分启动或关闭匹配标识
(?#...)		把子字符串当作注释
#...		在/x 模式中，把行内剩余部分当作注释

表 19　.NET 注释和模式转换器

.NET 归组、捕获、条件和控制	
序列	序列描述
(...)	把子模式和捕获子匹配归到\1，\2，...和$1，$2，...
\n	包含第 n 个被捕获的文本
$n	在替换字符串中，包含第 n 个捕获组中匹配的文本
(?<*name*>...)	把捕获的子字符串归到 *name* 组
(?:...)	把子模式分组，但是不捕获子匹配
(?>...)	自动分组
...|...	尝试子模式替换
*	匹配 0 或多次
+	匹配 1 次或多次
?	匹配 1 次或 0 次
{n}	匹配精确的 n 次
{n,}	至少匹配 n 次
{x,y}	至少匹配 x 次，最多 y 次
*?	匹配 0 次或多次，但是尽可能少
+?	匹配 1 次或多次，但是尽可能少
??	匹配 0 次或多次，但是尽可能少
{n,}?	至少匹配 n 次，但是尽可能少
{x,y}?	至少匹配 x 次，最多 y 次，但是尽可能少

表 20　.NET 归组、捕获、条件和控制

.NET 替换序列	
序列	序列描述
$1,$2,...	捕获的子匹配
$(*name*)	命名捕获组捕获的文本
$’	匹配(match)之前的的文本
$&	匹配文本(Text of match)
$`	匹配之后的文本
$+	最后加括号的匹配
$_	原始输入的拷贝

表 21 .NET 替换序列

正则表达式类和接口

.NET 在 System.Text.RegularExpressions 模块中定义了正则表达式的支持。RegExp()构造器处理正则表达式的创建，其余的 RegExp 方法处理模式匹配。Group 和 Match 类包含每一个匹配相关的信息。

C#的原始字符串语法，@" "，允许定义正则表达式，而不需要转义内嵌的反斜杠。

Regex

这个类处理正则表达式的创建和模式匹配。多个方法允许在不创建 RegExp 对象的情况下进行模式匹配。

方法

public Regex(string *pattern*)

public Regex(string *pattern*, RegexOptions *options*)

返回一个基于 *pattern* 和可选模式变换器，*options*，的正则表达式对象。

public static void CompileToAssembly(RegexCompilationInfo[] *regexinfos*,
System.Reflection.AssemblyName *assemblyname*)

public static void CompileToAssembly(RegexCompilationInfo[] *regexinfos*,
System.Reflection.AssemblyName *assemblyname*)

public static void CompileToAssembly(RegexCompilationInfo[] *regexinfos*,
System.Reflection.AssemblyName *assemblyname*,
System.Reflection.Emit.CustomAttributeBuilder[] *attributes*)

public static void CompileToAssembly(RegexCompilationInfo[] *regexinfos*,
System.Reflection.AssemblyName *assemblyname*,
System.Reflection.Emit.CustomAttributeBuilder[] *attributes*,
string *resourceFile*)

把一个或多个 Regex 编译成汇编。*regexinfos* 数组描述将要包含的正则表达式。汇编文件名是 *assemblyname*。*attributes* 数组定义了汇编文件的属性。*resourceFile* 是汇编中包括的一个 Win32 源文件。

public static stringEscape(string *str*)

返回一个转义了正则表达式、英镑符号（#）和空格的字符串。

public static bool IsMatch(string *input*, string *pattern*)
public static bool IsMatch(string *input*, string *pattern*, RegExpOptions *options*)
public static bool IsMatch(string *input*)
public static bool IsMatch(string *input*, int *startat*)

返回单一匹配输入字符串 *input* 的成功。这个方法的静态版本要求正则表达式 *pattern*。参数 options 允许使用可选的模式变换器（和 OR'd 一起）。*startat* 参数定义了字符串 *input* 中开始匹配的起点。

public static Match Match(string *input*, string *pattern*)
public static Match Match(string *input*, string *pattern*, RegExpOptions *options*)
public Match Match(string *input*)
public Match Match(string *input*, int *startar*, int *length*)

执行这对输入字符串 *input* 的一个单一匹配，并返回 Match 对象中匹配相关的信息。这个方法的静态版本要求使用正则表达式 *pattern*。参数 *options* 允许使用可选的模式变换器（和 OR'd 一起）。参数 *startat* 和 *length* 分别定义了字符串 *input* 中开始匹配的起点，和执行匹配的起点之后字符的个数。

public static MatchCollection Matches(string *input*, string *pattern*)
public static MatchCollection Matches(string *input*, string *pattern*,
RegExpOptions *options*)
public MatchCollection Matches(string *input*)
public MatchCollection Matches(string *input*, int *startat*)

返回输入字符串中所有的匹配，并返回 MatchCollection 对象中匹配相关的信息。这个方法的静态版本要求正则表达式参数 pattern。参数 options 允许使用可选的模式变换器（和 OR'd 一起）。参数 *startat* 定义了字符串 *input* 中开始匹配的起点。

public static string Replace(string *input*, *pattern*, MatchEvaluator *evluator*)
public static string Replace(string *input*, *pattern*, MatchEvalutor *evaluator*,
RegExpOptions *options*)
public string Replace(string *input*, MatchEvaluator *evaluator*)
public string Replace(string *input*, MatchEvaluator *evaluator*, int *count*)
public string Replace(string input, MatchEvaluator *evaluator*, int *count*, int *startat*)
public string Replace(string *input*, string *replacement*)
public string Replace(string *input*, string *replacement*, int *count*)
public string Replace(string *input*, string *replacement*, int *count*, int *startat*)

返回其中的匹配都被替换为 *replacement* 的字符串或 MatchEvaluator 对象的一个调用。字符串 *replacement* 允许使用"回引"（backreferences）通过$n 和${*name*}语法捕获文本。参数 *options* 允许使用可选的模式变换器（和 OR'd 一起）。参数 *count* 限制替换的次数。参数

startat 定义了字符串 *input* 中开始匹配的起点。

public static string[] Split(string *input*, string *pattern*)
public static string[] Split(string *input*, string *pattern*, RegexOptions *options*)
public static string[] Split(string *input*)
public static string[] Split(string *input*, int *count*)
public static string[] Split(string *input*, int *count*)
public static string[] Split(string *input*, int *count*, int *startat*)

返回一个被 *regex* 模式分开的字符串的数组。如果被指定了，最多返回 *count* 个字符串。可以通过参数 *startat* 指定字符串 *input* 中开始匹配的起点。

Match

属性

public bool *Success*

描述匹配是否成功。

public string *Value*

匹配的文本。

public int *Length*

被匹配的文本中字符的个数。

public int *Index*

以 0 作为基准的匹配起点的字符索引。

public GroupCollection *Group*

一个 GroupCollection 对象，Groups[0].value 包含了整个匹配的文本，每一个附加的 *Groups* 元素包含被捕获组匹配的文本。

方法：

public Match NextMatch()

返回输入字符串中 *regex* 的下一个匹配的一个 Match 对象。

pubic virtual stringRsult(string *result*)

返回被前一个匹配替换的特殊的替换序列的结果 *result*。

public static Match Synchronized(Match *inner*)

返回标识 *inner* 的一个 Match 对象，但是对多线程也安全。

Group

属性

public bool *Success*

如果组在匹配中，则返回 true。

public string *Value*

被这个组捕获的文本。

public int *Length*

被这个组捕获的字符个数。

public int *Index*

以 0 作为基准被组捕获的文本起点的字符索引。

Unicode 支持

.NET 支持 Unicode 3.1,包括完全支持\w、\d、\s 序列。通过启动 ECMAScript 模式可以将匹配的字符局限在 ASCII。大小写不敏感的匹配受限于 Thread.CurrentCulture 定义的当前语言，除非设置了 CultureInvariant。

.NET 支持标准 Unicode 属性（参看表 2）和区位。只支持简短的属性命名形式。区位名要求以 Is 作为前缀，而且必须使用简短的命名形式，其中不能出现空格和下划线。

实例

实例 9 简单匹配

简单匹配

```
//Find Spider-Man, Spiderman, SPIDER-MAN, etc.

namespace Regex_PocketRef
{
    using System.Text.RegularExpressions;
    class SimpleMatchTest
    {
        static void Main( )
        {
            string dailybugle = "Spider-Man Menaces City!";
            string regex = "spider[- ]?man";
            if (Regex.IsMatch(dailybugle, regex, RegexOptions.
            IgnoreCase)) {
            //do something
        }
    }
}
```

实例 10 匹配和归组

匹配和归组

```
//Match dates formatted like MM/DD/YYYY, MM-DD-YY,...
using System.Text.RegularExpressions;
class MatchTest
{
      static void Main( )
      {
            string date = "12/30/1969";
            Regex r = new Regex( @"^(\d\d)[-/](\d\d)[-/](\d\d(?:\d\d)?)$" );
            Match m = r.Match(date);
            if (m.Success) {
                  string month = m.Groups[1].Value;
                  string day = m.Groups[2].Value;
                  string year = m.Groups[3].Value;
            }
      }
}
```

实例 11 简单替换

简单替换

```
//Convert <br> to <br /> for XHTML compliance
using System.Text.RegularExpressions;
class SimpleSubstitutionTest
{
      static void Main( )
      {
            string text = "Hello world. <br>";
            string regex = "<br>";
            string replacement = "<br />";
            string result =
            Regex.Replace(text, regex, replacement, RegexOptions.IgnoreCase);
      }
}
```

实例 12 高难度替换

高难度替换

```
//urlify - turn URLs into HTML links
using System.Text.RegularExpressions;
public class Urlify
{
    static Main ( )
    {
        string text = "Check the web site, http://www.oreilly.com/
        catalog/regexppr.";
        string regex = @"\b # start at word boundary( # capture to $1
                    (https?|telnet|gopher|file|wais|ftp) :
                    # resource and colon
                    [\w/#~:.?+=&%@!\-] +? # one or more valid
                    # characters
                    # but take as little as
                    # possible)
                    (?= # lookahead
                    [.:?\-] * # for possible
                    # punctuation
                    (?: [^\w/#~:.?+=&%@!\-] # invalid character
                    | $ ) # or end of string
                    )";
        Regex r = new Regex(regex, RegexOptions.IgnoreCase
                | RegexOptions.IgnorePatternWhitespace);
                string result = r.Replace(text, "<a href=\"$1\">$1</a>");
    }
}
```

其他资源

- O’Reilly 出版的 Jesse Liberty 著作《C#编程》对 C#，.NET 和正则表达式做了一个整体的介绍。
- O’Reilly 出版的 Jeffrey E. F. Friedl 著作《精通正则表达式》第三版 399-432 页包括了.NET 正则表达式的详细信息。
- 微软在线帮助文档 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconregularexpressionslanguageelements.asp.

PHP

本章涵盖了 PHP4.3 和 preg 支持的 5.1.4 Perl-style（Perl 风格）正则表达式。两者都是基于 PCRE 6.6 类库。preg 采用传统的 NFA 匹配引擎。如果想进一步了解传统的 NFA 引擎背后的规则，请看“正则表达式和模式匹配”一节。

支持的元字符

PHP 支持表 22 到表 26 中列出来的元字符和元序列。关于每一个元字符的详述，请看“正则表达式元字符、模式和结构”一节。

PHP 字符表示	
序列名	序列描述
\a	告警（bell），\x07
\b	空格，\x08，只有在字符类中有效
\e	Esc 字符，\x1B
\n	换行，\x0A
\r	回车，\x0D
\f	分页，\x0C
\t	水平制表符(tab)，\x09
\0octal	通过 1、2 或 3 个八进制码数指定的字符
\xhex	通过 1 个或 2 个十六进制数指定的字符
\x{hex}	通过十六进制码指定的字符
\cchar	命名的控制字符

表 22　PHP 字符表示

PHP 字符类和类似（class-like）结构	
字符类	类描述
[…]	列出来的或包含在列表范围的单一字符
[^…]	不在列出来的或不包含在列表范围的单一字符
[:*class*:]	POSIX 风格的字符类（只有在 regex 字符类中有效）
.	除行终止（除非是单行模式，/s）之外的任意字符
\C	1 字节（然而，这可能会破坏 Unicode 字符流）
\w	字字符，[a-zA-Z0-9_]
\W	非字字符， [^a-zA-Z0-9_]
\d	数字字符， [0-9]
\D	非数字字符，[^0-9]
\s	空格字符，[\n\r\f\t]
\S	非空格字符，[^ \n\r\f\t]

表 23　PHP 字符类和类似（class-like）结构

PHP 锚和其他 0 宽测试	
序列名	序列描述
^	字符串的开头，或，在多行匹配模式（/m），任意换行之后的位置
\A	在任意匹配模式(mode)，搜索字符串的开头
$	字符串末尾，或在多行模式（/m），任意换行之前的位置
\Z	在任意匹配模式下，字符串末尾或字符串末尾换行之前的位置
\z	任意匹配模式下，字符串末尾
\G	当前匹配的开头
\b	\w 和\W 之间的字边界
\B	非字边界
(?=…)	正向前(Positive lookahead)
(?!…)	负向前(Negative lookahead)
(?<=…)	正向后(Positive lookbehind)
(?<!...)	负向后(Negative lookbehind)

表 24　PHP 锚和其他 0 宽测试

PHP 注释和模式转换器	
模式名	模式描述
i	大小写不敏感匹配
m	^和$匹配下一个内嵌的行终止符（\n）
s	点号匹配换行
x	忽略空格，并允许以#开头的注释
U	转换计量器的贪婪性：* 变成懒惰，而 *?成为贪婪
A	迫使匹配从主体字符长的起始开头
D	迫使$匹配字符串的末尾，而不是换行符之前这将被多行模式重写。
u	把正则表达式和主题字符串视为 UTF-8 字符
(?*mode*)	为其余子表达式启动所列的模式（一个或多个 imsxU）
(?-*mode*)	为其余子表达式取消所列的模式（一个或多个 imsxU）
(?*mode*:...)	启动括号内的模式（xsmi）
(?-*mode*:...)	取消括号内的模式（xsmi）
(?#...)	把子字符串当作注释
#...	在/x 模式中，把行内剩余部分当作注释
\Q	引掉（注释掉）后续的正则表达式
\E	终止一个从\Q 开始的 span

表 25　PHP 注释和模式转换器

PHP 归组、捕获、条件和控制	
序列	序列描述
(...)	把子模式和捕获子匹配归到\1，\2，...和$1，$2，...
(?P<*name*>...)	在替换字符串中，包含第 n 个捕获组中匹配的文本
\n	包含第 n 个被捕获的文本
(?:...)	把子模式分组，但是不捕获子匹配
(?>...)	自动分组
...|...	尝试子模式替换
*	匹配 0 或多次
+	匹配 1 次或多次
?	匹配 1 次或 0 次
{n}	匹配精确的 n 次
{n,}	至少匹配 n 次
{x,y}	至少匹配 x 次，最多 y 次
*?	匹配 0 次或多次，但是尽可能少
+?	匹配 1 次或多次，但是尽可能少
??	匹配 0 次或多次，但是尽可能少
{n,}?	至少匹配 n 次，但是尽可能少
{x,y}?	至少匹配 x 次，最多 y 次，但是尽可能少
*+	匹配 0 此或多次，并且从不回溯
++	匹配 1 此或多次，并且从不回溯
?+	匹配 0 此或 1 次，并且从不回溯
{n}+	至少匹配 n 次，并且从不回溯
{n,}+	至少匹配 n 次，并且从不回溯
{x,y}+	至少匹配 x 次，最多匹配 y 次，并且从不回溯
(?(*condition*)...|...)	匹配 if-then-else 模式。*condition* 可以是捕获组、回环或向后结构的个数
(?(*condition*)...)	匹配 if-then-else 模式。*condition* 可以是捕获组、回环或向后结构的个数

表 26　PHP 归组、捕获、条件和控制

模式匹配函数

PHP 为正则表达式提供了很多独立的函数。当创建一个正则表达式字符串时，需要转义内嵌的反斜杠；反斜杠在传给正则表达式引擎之前已经被解释了。

array preg_grep(string *pattern*, array *input*)

返回一个包含 *input* 中被 *pattern* 匹配的每一个元素的数组。

int preg_match_all(string *pattern*, string *subject*, array *matches*[, int *flags*])

搜索 *subject* 中被 *pattern* 匹配的所有的匹配。被匹配的子字符串被放置到 *matches* 数组中。*matches* 的第一个元素是一个包含每一个全匹配的文本的数组。*matches* 的每一个附加的元素 n 包含每一个全匹配的第 *n* 个捕获组匹配。所以，例如 *matches*[7][3]包含 *subject* 中 *pattern* 第 4 个匹配中被第 7 捕获组捕获的文本。

可以通过 PREG_SET_ORDER 标识来明确 *matches* 的先后顺序。PREG_SET_ORDER 设置一个更直观的顺序，这样 *matches* 中的每一个元素就是与一个匹配相关的数组。每一个数组元素 0 是完成的匹配，附加的元素与捕获组有关。附加的标识 PREG_OFFSET_CAPTURE 引发每一个包含一个字符串的数组被 *subject* 中包含相同字符串和起始字符的二元数组替换。

int preg_match(string *pattern*, string *subejct*[,array *matches*[, int *flags*]])

如果在 *subject* 中有 *pattern* 匹配则返回 1，否则返回 0。如果提供了 *matches* 数组，则匹配的字符串被 *matches*[0]替换，每一个捕获组在子序列元素中被替换。一个允许的标识 PREG_OFFSET_CAPTURE，引发 *matches* 中的每一个元素被包含匹配中字符串和起点字符位置的二元数组替换。

string preg_quotr(string *str*[, string *delimiter*])

返回一个所有的正则表达式和元字符都被转义的 str。如果你在自己的正则表达式中用到分隔符，则提供 delimiter 参数，并且需要在 str 中转义分隔符。

mixed preg_replace_callback(mixed *pattern*, callback *callback*, mixed *subject*[, int *limit*])

返回被 callback 替换的每一个发生的 *pattern* 的 *subject* 的文本。*callback* 需要一个参数，一个包含被匹配的文本的数组，和捕获组中每一个匹配。如果提供了 *limit*，则最多执行 *limit* 次替换。

如果 pattern 是一个数组，则每一个元素都将被 *callback* 替换。如果 *subject* 是一个数组，则函数在每一个元素上进行迭代。

mixed preg_replace(mixed *pattern*, mixed *replacement*, mixed *subject*[, int *limit*])

返回每一个发生的 *pattern* 都被 *replacement* 替换的 *subject* 的文本。如果提供了 *limit*，则函数最多执行 *limit* 次替换。替换的字符串可能和匹配，或使用$n(preferred)，或\n(deprecated)的捕获组匹配有关。如果 *pattern* 有/e 变换器，则 *replacement* 被子替换解析，并且作为 PHP 代码执行。

如果 *pattern* 是一个数组，则每一个元素都将被 *callback* 替换。如果 *replacement* 是一个数组，相关的元素在 *replacement* 中。如果 *subject* 是一个数组，则函数在每一个元素上进行迭代。

array preg_split(string *pattern*, string *subjects*[, int *limit* [, int *flags*]])

返回被 pattern 分开的字符串的一个数组。如果指定了 *limit*，则 *preg_init* 最多返回 *limit* 个子字符串。如果 *limit* 被指定为-1，则相当于“没有任何限制”，允许设置标识。可用的标识有：PREG_SPLIT)NO_EMPTY，仅返回非空的部分；PREG_SPLIT_DELIM_CAPTURE，

返回每一个被拆分的子字符串之后被捕获的子匹配；PREG_SPLIT_OFFSET_CAPTURE，返回第一个元素是匹配，而第二个元素是匹配在 *subject* 中的偏移的一个二元数组。

实例

实例 13 简单匹配

实例 13 简单匹配

```
//Find Spider-Man, Spiderman, SPIDER-MAN, etc.

$dailybugle = "Spider-Man Menaces City!";
$regex = "/spider[- ]?man/i";
if (preg_match($regex, $dailybugle)) {
      //do something
}
```

实例 14 匹配和归组

实例 14 匹配和归组

```
//Match dates formatted like MM/DD/YYYY, MM-DD-YY,...

      $date = "12/30/1969";
      $p = "!^(\\d\\d)[-/](\\d\\d)[-/](\\d\\d(?:\\d\\d)?)$!";
      if (preg_match($p,$date,$matches) {
            $month = $matches[1];
            $day = $matches[2];
            $year = $matches[3];
      }
...
```

实例 15 简单替换

实例 15 简单替换

```
//Convert <br> to <br /> for XHTML compliance

$text = "Hello world. <br>";
$pattern = "{<br>}i";
echo preg_replace($pattern, "<br />", $text);
```

实例 16 高难度替换

实例 16 高难度替换

```
//urlify - turn URLs into HTML links

$text = "Check the web site, http://www.oreilly.com/catalog/
regexppr.";
$regex =
"{ \\b # start at word\n"
. " # boundary\n"
. "( # capture to $1\n"
. "(https?|telnet|gopher|file|wais|ftp) : \n"
. " # resource and colon\n"
. "[\\w/\\#~:.?+=&%@!\\-]+? # one or more valid\n"
. " # characters\n"
. " # but take as little as\n"
. " # possible\n"
. ")\n"
. "(?= # lookahead\n"
. "[.:?\\-]* # for possible punct\n"
. "(?:[^\\w/\\#~:.?+=&%@!\\-] # invalid character\n"
. "|$) # or end of string\n"
. ") }x";
echo preg_replace($regex, "<a href=\"$1\">$1</a>", $text);
```

其他资源

- PHP 在线帮助文档, http://www.php.net/pcre。
- O’Reilly 出版的 Jeffrey E. F. Friedl 著作《精通正则表达式》第三版 439-482 页包括了 PHP 正则表达式的详细信息。

Python

Python 在 re 模块中提供了一个丰富的，perl 风格的正则表达式语法。re 采用传统的 NFA 匹配引擎。如果想进一步了解传统的 NFA 引擎背后的规则，请看“正则表达式和模式匹配”一节。

本章将涵盖 Python 2.3.5 中包含的 re 版本，虽然相似形式的模块自从 Python 1.5 中就已经有了。

支持的元字符

Python 支持表 27 到表 31 中列出来的元字符和元序列。关于每一个元字符的详述，请看“正则表达式元字符、模式和结构”一节。

Python 字符表示	
序列名	序列描述
\a	告警（bell），\x07
\b	空格，\x08，只有在字符类中有效
\e	Esc 字符，\x1B
\n	换行，\x0A
\r	回车，\x0D
\f	分页，\x0C
\t	水平制表符(tab)，\x09
\v	垂直制表，\x0B
\0octal	通过 3 个八进制码数指定的字符
\xhh	通过 2 个十六进制数指定的字符
\uhhhh	通过 4 个十六进制码指定的字符
\Uhhhhhhhh	通过 8 个十六进制码指定的字符

表 27　Python 字符表示

Python 字符类和类似（class-like）结构	
字符类	类描述
[…]	列出来的或包含在列表范围的单一字符
[^…]	不在列出来的或不包含在列表范围的单一字符
.	除换行符（除非是 DOTALL 模式）之外的任意字符
\w	字字符，[a-zA-Z0-9_]（除非是 LOCAL 或 UNICODE 模式）
\W	非字字符， [^a-zA-Z0-9_]（除非是 LOCAL 或 UNICODE 模式）
\d	数字字符， [0-9]
\D	非数字字符，[^0-9]
\s	空格字符，[\t\n\r\f\v]
\S	非空格字符，[^ \t\n\r\f\v]

表 28　Python 字符类和类似（class-like）结构

Python 锚和其他 0 宽测试	
序列名	序列描述
^	字符串的开头，或，在多行匹配模式（/m），任意换行之后的位置
\A	在任意匹配模式(mode)，搜索字符串的开头
$	字符串末尾，或在多行模式（/m），任意换行之前的位置
\Z	在任意匹配模式下，字符串末尾或字符串末尾换行之前的位置
\z	任意匹配模式下，字符串末尾
\b	\w 和\W 之间的字边界
\B	非字边界
(?=…)	正向前(Positive lookahead)
(?!…)	负向前(Negative lookahead)
(?<=…)	正向后(Positive lookbehind)
(?<!...)	负向后(Negative lookbehind)

表 29　Python 锚和其他 0 宽测试

Python 注释和模式转换器		
转化器/序列	模式字符	转换器描述
I or IGNORECASE	i	大小写不敏感匹配
L or LOCAL	L	引发\w,\W,\b 和\B 采用字母数字的当前位置定义
M or MULTILINE or(?m)	m	^和$匹配下一个内嵌的行终止符（\n）
S or DOTALL or (?s)	s	点号匹配换行
U or UNICODE or(?u)	u	引发\w,\W,\b 和\B 采用字母数字的 Unicode 定义
X or VERBOSE or(?x)	x	忽略空格，并允许模式中以#开头的注释
(?*mode*)		为其余子表达式启动所列的模式（一个或多个 iLmsux）
(?#...)		把子字符串当作注释
#...		在 VERBOSE 模式中，把行内剩余部分当作注释

表 30　Python 注释和模式转换器

Python 归组、捕获、条件和控制	
序列	序列描述
(...)	把子模式和捕获子匹配归到\1，\2，...和$1，$2，...
(?P<*name*>...)	把子模式和捕获子匹配归到命名捕获组 name
(?P=*name*)	被命名组 name 更早匹配的文本匹配。
\n	包含第 n 个更早子匹配的结果
(?:...)	归组子模式，但是不捕获子匹配
...|...	尝试子模式替换
*	匹配 0 或多次
+	匹配 1 次或多次
?	匹配 1 次或 0 次
{n}	匹配精确的 n 次
{x,y}	至少匹配 x 次，最多 y 次
*?	匹配 0 次或多次，但是尽可能少
+?	匹配 1 次或多次，但是尽可能少
??	匹配 0 次或多次，但是尽可能少
{x,y}?	至少匹配 x 次，最多 y 次，但是尽可能少

表 31　Python 归组、捕获、条件和控制

re 模块对象和函数

re 模块定义了所有的正则表达式功能。模式匹配可以直接由模块函数完成，通过编译成为正则表达式的模式，可以用来重复匹配。匹配和捕获组相关的信息通过匹配对象进行遍历。

Python原始语法，r′′ 或r " " 允许直接指定正则表达式模式，而不需要对反斜杠进行转义。原始字符串(raw-string)模式，r′ \n′ 和正则表达式\\n等效。Python还为多行正则表达式提供三个引号的原始字符串语法，如，r′′′ text′′′ 或r " " " text " " " 。

模块函数

re 模块定义了一下函数和一个异常。

compile(*pattern* [, *flags*])

通过可选的模式变换器，*flags*，返回一个正则表达式对象。

match(*pattern*, *sting*[,*flags*])

在 *string* 中搜索 *pattern*，如果成功，则返回一个匹配对象，否则返回 None.

split(*pattern*, *string*[, *maxsplit*=0])

在 *pattern* 上拆分 *string*，并通过 *maxsplit* 来限制拆分的个数。同时，返回捕获括号中的子匹配。

sub(*pattern*, *repl*, *string*[, *count*=0])

返回一个 *pattern* 全部或 *count* 次被 *repl* 替换的字符串。*repl* 可以是一个字符串，也可以是一个带一个匹配对象参数的函数。

subn(*pattern*, *repl*, *strng*[, *count*=0])

执行 sub()，但是返回一个新字符串的一元组和替换的次数。

findall(*pattern*, *string*)

返回 *string* 中 *pattern* 的所有匹配。如果 *pattern* 有捕获组，则返回子匹配的列表或子匹配一元组列表。

finditer(*pattern*, *string*)

通过 *string* 中 *pattern* 的匹配返回一个迭代。对于每一个匹配，迭代器返回一个匹配对象。

escape(*string*)

返回一个带反斜杠的字母数字的字符串，这样 *string* 就可以被逐个匹配。

exception error

如果在编译或匹配过程中有错误发生，那么就会产生一个异常。如果传给函数的不是一个有效正则表达式的字符串，那么这是经常会发生的。

RegExp

正则表达式是通过 re.*compile* 函数创建的。

flags

返回一个对象被编译时用到的 *flags* 参数或 0。

groupindex

返回映射符号组名和符号个数的一个字典。

pattern

返回对象被编译时用的一个模式字符串。

match(*string*[, *pos*[, *endpos*]])
search(*string*[, *pos* [, *endpos*]])
split(*string*[, *maxsplit*=0])
sub(*repl*, *string*[, *count*=0])
subn(*repl*, *string*[, *count*=0])
findall(*string*)

和 *re* 模块的函数有些相似，只是模式是隐含的（没有明确指定）。*pos* 和 *endpos* 匹配匹配指定字符串的起始和终止索引。

匹配对象

匹配对象由 *match* 和 *find* 函数创建。

pos
endpos

pos 和 *endpos* 的值传给 *search* 或 *match*。

re

正则表达式对象，它的 *match* 或 *search* 返回这个对象。

string

被传给 *match* 和 *search* 的字符串。

group([*g1*,*g2*,...])

从捕获组中返回一个或多个子匹配。Groups 可能是与捕获组相关的数字，或与命名捕获组相关的字符串。Group 0 与整个匹配有关。如果没有提供任何的参数，这个函数返回整个匹配。没有匹配的捕获组返回一个 None。

groups([*default*])

返回所有捕获组的结果的一个元组。没有匹配的捕获组有一个 None 或 *default* 值。

groupdict([])

返回一个由组名为关键子的命名捕获组字典。没有匹配的捕获组有一个 None 或 *default* 值。

start([*group*])

被 *group* 匹配的子字符串的起点索引（如果没有 *group* 则是整个被匹配的字符串的起始索引）。

end([group])

被 *group* 匹配的子字符串的结尾索引（如果没有 *group* 则是整个被匹配的字符串的末尾索引）。

span([group])

返回 *group* 起始和终止索引的一个元组（如果没有 *group* 则返回被匹配的字符串）。

expand([grounp])

返回一个通过在 *template* 上执行反斜杠替换所选择的一个字符串。字符转义、数字回引

(numeric backreference)和命名回引(named backreference)被扩充。

lastgroup

最后匹配捕获组的名字，或如果没有匹配或组没有名字则为 None

lastindex

最后匹配捕获组的索引，如果没有匹配则为 None

Unicode 支持

re 提供了有限的 Unicode 支持。字符串可能包含 Unicode 字符，而且独立的 Unicode 字符可以通过\u 来指定。加之，Unicode 标识使\w、\W、\b 和\B 识别所有的 Unicode 字母数字。然而，re 没有为 Unicode 属性和区位或分类提供支持。

实例

实例 17 简单匹配

实例 13 简单匹配

```
#Find Spider-Man, Spiderman, SPIDER-MAN, etc.
import re
dailybugle = 'Spider-Man Menaces City!'
pattern = r'spider[- ]?man.'
if re.match(pattern, dailybugle, re.IGNORECASE):
    print dailybugle
```

实例 18 匹配和归组

实例 18 匹配和归组

```
#Match dates formatted like MM/DD/YYYY, MM-DD-YY,...
import re
date = '12/30/1969'
regex = re.compile(r'^(\d\d)[-/](\d\d)[-/](\d\d(?:\d\d)?)$')
match = regex.match(date)
if match:
    month = match.group(1) #12
    day = match.group(2) #30
    year = match.group(3) #1969
...
```

实例 19 简单替换

实例 19 简单替换

```
#Convert <br> to <br /> for XHTML compliance

import re
text = 'Hello world. <br>'
regex = re.compile(r'<br>', re.IGNORECASE);
repl = r'<br />'
result = regex.sub(repl,text)
```

实例 20 高难度替换

实例 20 高难度替换

```
//urlify - turn URLs into HTML links

import re
text = 'Check the web site, http://www.oreilly.com/catalog/regexppr.'
pattern = r'''
      \b                                  # start at word boundary
      (                                   # capture to \1
      (https?|telnet|gopher|file|wais|ftp) :
                                          # resource and colon
      [\w/#~:.?+=&%@!\-] +?               # one or more valid chars
                                          # take little as possible
      )
      (?=                                 # lookahead
      [.:?\-] *                           # for possible punc
      (?: [^\w/#~:.?+=&%@!\-]             # invalid character
      | $ )                               # or end of string
      )'''

regex = re.compile(pattern, re.IGNORECASE+ re.VERBOSE)
result = regex.sub(r'<a href="\1">\1</a>', text)
```

其他资源

- Python 在线帮助文档, http://www.python.org/doc/current/lib/module-re.html。

RUBY

Ruby 在 Regexp 和 String 类中提供了 Perl 风格的正则表达式的一个子集。Ruby 采用了传统的 NFA 匹配引擎。如果想进一步了解传统的 NFA 引擎背后的规则，请看“正则表达式和模式匹配”一节。

Ruby 1.9 中介绍了一种新的正则表达式引擎，其中包含了许多新的功能。这些功能作为 Oniguruma 库的一部分，在早些版本中就可用了。以下的内容主要包括 Ruby1.8.6，不过包括了 Ruby1.9 中主要的功能，并有相关的标记。

支持的元字符

Ruby 支持表 32 到表 37 中列出来的元字符和元序列。关于每一个元字符的详述，请看“正则表达式元字符、模式和结构”一节。

Ruby 字符表示	
序列名	序列描述
\a	告警（bell），\x07
\b	空格，\x08，只有在字符类中有效
\e	Esc 字符，\x1B
\n	换行，\x0A
\r	回车，\x0D
\f	分页，\x0C
\t	水平制表符(tab)，\x09
\v	垂直制表符（tab），\x0B
\0octal	通过 2 个八进制码数指定的字符
\xhex	通过 2 个十六进制数指定的字符
\cchar	命名的控制字符

表 32 Ruby 字符表示

Ruby 字符类和类似（class-like）结构	
字符类	类描述
[…]	列出来的或包含在列表范围的单一字符
[^…]	不在列出来的或不包含在列表范围的单一字符
.	除行终止（除非是单行模式，s）之外的任意字符
\w	字字符
\W	非字字符
\d	数字字符
\D	非数字字符
\s	空格字符，[\f\n\r\t\v]
\S	非空格字符，[^ \f\n\r\t\v]

表 33　Ruby 字符类和类似（class-like）结构

Ruby 锚和其他 0 宽测试	
序列名	序列描述
^	字符串的开头，或，在多行匹配模式（MULTILINE），任意换行之后的位置
\A	在任意匹配模式，搜索字符串的开头
$	字符串末尾，或在多行模式（MULTILINE），任意换行之前的位置
\Z	在任意匹配模式下，字符串末尾或字符串末尾换行之前的位置
\z	任意匹配模式下，字符串末尾
\b	\w 和\W 之间的字边界
\B	非字边界
\G	前一个匹配的末尾
(?=…)	正向前(Positive lookahead)
(?!…)	负向前(Negative lookahead)

表 34　Ruby 锚和其他 0 宽测试

转换器/序列	转换器描述
m	点号匹配包括行终止符在内的任意字符
x	忽略空格，并允许以#开头的注释
l	在当前文化（current culture）中大小写不敏感匹配
n	关闭（取消）通配符处理
o	仅执行一次#{...}，默认在 regex 执行时执行
(?imns-imns)	为模式剩下的部分启动或关闭匹配标识
(?imns-imns:...)	为子表达式剩下的部分启动或关闭匹配标识
(?#...)	把子字符串当作注释
#...	在/x 模式中，把行内剩余部分当作注释
(?<=…)	正向后(Positive lookbehind)
(?<!...)	负向后(Negative lookbehind)

表 35 Ruby 注释和模式转换器

Ruby 归组、捕获、条件和控制	
序列	序列描述
(...)	把子模式和捕获子匹配归到\1，\2，...和$1，$2，...
(?<*name*>...)	命名的捕获组模式满足\k<name>
\n	在一个正则表达式中，匹配前第 n 个被捕获的子匹配
$n	在替换字符串中，包含第 n 个捕获组中匹配的文本
\k<name>	在一个替换字符串中，包含命名的字匹配 name
(?:...)	把子模式分组，但是不捕获子匹配
(?>...)	自动分组
...|...	尝试子模式替换
*	匹配 0 或多次
+	匹配 1 次或多次
?	匹配 1 次或 0 次
{n}	匹配精确的 n 次
{n,}	至少匹配 n 次
{x,y}	至少匹配 x 次，最多 y 次
*?	匹配 0 次或多次，但是尽可能少
+?	匹配 1 次或多次，但是尽可能少
??	匹配 0 次或多次，但是尽可能少
{n,}?	至少匹配 n 次，但是尽可能少
{x,y}?	至少匹配 x 次，最多 y 次，但是尽可能少

表 36　Ruby 归组、捕获、条件和控制

Ruby 替换序列	
序列	序列描述
$1,$2,...	捕获的子匹配
$(*name*)	命名捕获组捕获的文本
$’	匹配(match)之前的的文本
$&	匹配文本(Text of match)
$`	匹配之后的文本
$+	最后加括号的匹配

表 37　Ruby 替换序列

面向对象的接口

Ruby 通过 Regex 和 MatchData 类以及 String 类内建的一些方法提供了一个面向对象的正则表达式接口。

Ruby 还提供/.../和=~操作符，来提供 Perl 风格的操作语法。操作符/.../是 Regexp.new 的替代词，而=~是 String#match 的替代词。/.../操作通常被用来给一个方法传递 Regex 对象，

例如，"foo, bar, frog".split(/,\s*/)。

String

描述

String 对象包含一些针对正则表达式模式匹配和替换的方法，和一些以字符串操纵为主的以正则表达式为参数的方法。

实例方法（Instance Methods）

string =~ regexp => fixnum or nil

匹配 regexp，并返回匹配开始的位置或 nil（空）。

regexp = = = string => boolean

用在 case-when 语句中，如果 regexp 匹配字符串，则返回 true。

gsub(*pattern*, *replacement*) => new_string

gsub(*pattern*){ |*match*| *block* } => new_string

返回所有发生的 *pattern* 都被 *replacement* 替换了的字符串，或 *block* 的值。否则，执行和 Regexp#sub 等效的操作。

gsub!(*pattern*, *replacement*) => string or nil

gsub!(*pattern*){ |*match*| *block*} => string or nil

在适当的位置执行 String#sub 替换，并返回字符串；如果替换没有执行则返回 nil。

index(*regexp* [, *offset*]) => fixnum or nil

返回 regexp 第一个匹配的索引；如果没有找到，则返回 nil。*offset* 指定在字符串中开始搜索的位置。

match(*pattern*) => matchdata or nil

在字符串中应用 *pattern* 或 *regexp* 对象。如果匹配返回 *matchdata*，否则返回 nil。

rindex(*regexp*[,*fixnum*]) => fixnum or nil

返回 *regexp* 第一个匹配的索引；如果没有找到，则返回 nil。offset 指定在字符串中开始搜索的位置。这个点右边的字符将不被考虑。

scan(regexp) => array

scan(regex) {|match, ... | block} = > string

在字符串中进行迭代，返回一个匹配数组，或者，如果 *regexp* 包含匹配组，则返回一个数组的数组。

[regexp] => substring or nil

[regexp, fixnum] => substring or nil

slice(*regexp*) => substring or nil

slice(*regexp*, *fixnum*) => substring or nil

返回匹配的子串或 nil。如果指定了 fixnum 则返回相关的字匹配。

slice!(*regexp*) => new_str or nil

删除字符串中匹配的部分，并返回被删除的字符串，或，如果没有找到匹配，则返回 nil。

split(*pattern*=$;, [*limit*]) => anArrary

根据分隔符把字符串分成多个子串。这些子字符串既可以是字符串，也可以是 Regexp 对象。如果指定了 *limit*，则最多返回 *limit* 个匹配。如果没有指定 *limit*，那么忽略空的子字符串。如果 *limit* 是一个负数，则返回所有的子字符串，包括空的子字符串。

sub(*regexp*, *replacement*) => new_string
sub(*regexp*){|*match*| *block*} => new_string

返回所有发生的 *pattern* 都被 *replacement* 替换了的字符串，或 *block* 的值。*replacement* 可以应用\1,\2,...,\n。*block* 可以引用特殊的匹配变量$1,$2,$`,$&和$'。

sub!(*pattern*, *replacement*) => string or nil
sub!(*pattern*){|*match*| *block*} => string or nil

在适当的位置执行 String#sub 替换，并返回字符串；如果替换没有执行则返回 nil。

Regexp

描述

保持一个用于字符匹配一个模式的正则表达式。

类的方法

escape(*string*) => escaped_string
quote(*string*) => escaped_string

转义一个正则表达式，这样用于正则表达式模式中的时候不会被解释。

last_match = > matchdata
last_match(*n*) => string

返回最后成功匹配的 MatchData 或 MatchData 对象中第 *n* 个字段。

Regexp.new(*pattern* [, *options* [, *lang*]]) => regexp
Regexp.compile(*pattern* [, *options* [, *lang*]]) => regexp

从正则表达式模式中创建一个新的 Regexp 对象。*options* 可以是 Regexp::EXTENDED，Regexp::IGNORECASE 和 Regexp::MULTILINE 的一个 OR'd 联合。*lang* 参数加强了对 Regexp: 'n', 'N', = none, 'e', 'E' = EUC, 's' , 'S' = SJIS, 'u', 'U' = UTF-8 的支持。

Regexp::union([*pattern*]*) => new_str

通过由替换符连接的给定模式的联合创建一个 Regexp 对象。其中模式既可以是模式字符串，也可以是 Regexp 对象。

实例方法

regexp = = second_regexp = > boolean
regexp.eql?(*second_regexp*) = > boolean

如果两个 Regexp 对象给予相同标识的模式，而且拥有相同的字符集编码和模式选项，则返回 true。

match(*string*) => matchdata or nil

返回描述匹配的一个 MatchData 对象。如果没有匹配，则返回 nil。

casefold? => true or false

如果整个模式都设置了 IGNORECASE，则返回 true。

inspect => string

返回表示 Regexp 对象的一个字符串。

kcode => string

返回 Regexp 对象的字符集编码。

options => fixnum

依据创建 Regexp 时的可选项，返回 *bits* 集合。这个 *bits* 可以在创建新的 Regexp 时作为参数传递。

source => string

返回原来的模式字符串。

to_s => string

返回包含采用(?imns-imns:...)注释的正则表达式和可选项的一个字符串。

MatchDate

描述

保持一个成功匹配的结果，包括被匹配的字符串和匹配组中的字匹配。

实例方法

[*i*] => string

[*start*, *length*] => array

[*range*] => array

以一个数组访问匹配结果。第 0 个元素是整个被匹配的字符串，元素 1 到 *n* 是子匹配。

begin(*n*) => integer

返回字符串中第 *n* 个子匹配的起点的偏移量。

captures => array

返回捕获数组，等效于 MatchData#to_a。

end(*n*) => integer

返回字符串中第 *n* 个子匹配的末尾的偏移量。

length = > integer

size => integer

返回元素的个数，包括匹配数组中完全匹配和子匹配。

offset(*n*) => array

返回包括递 n 个子匹配起点和终点的偏移量的二元数组。

post_match => string

返回原始字符串中当前匹配之后的部分（和$`等效）。

pre_match => string

返回原始字符串中当前匹配之前的部分（和$`等效）。

select([*index*]*) => array

通过 index 来访问子匹配，返回相关值的一个数组。

string => original_string

返回传给匹配的字符串的一个副本。

to_a => array

返回匹配的数组。

to_s => 返回整个匹配的字符串。

Unicode 支持

Ruby 有一些 UTF-8 的支持，但是必须通过编码$KCODE = “UTF8”，才能使用这种结构。同时，支持 ASCII 之外的\w，\d、\s、\b 等 Unicode 字符。通过给 Regexp.new 传递一个语言参数，就可以使用多字节的 regex 处理了，同时，可以通过/n 来关闭多字符变换器。

实例

实例 21 简单匹配

实例 21 简单匹配

```
#Find Spider-Man, Spiderman, SPIDER-MAN, etc.

dailybugle = 'Spider-Man Menaces City!'

if dailybugle.match(/spider[- ]?man./i)

        puts dailybugle

end
```

实例 22 匹配和归组

实例 22 匹配和归组

```
#Match dates formatted like MM/DD/YYYY, MM-DD-YY,...
date = '12/30/1969'
regexp = Regexp.new('^(\d\d)[-/](\d\d)[-/](\d\d(?:\d\d)?)$')
if md = regexp.match(date)
        month = md[1] #12
        day = md[2] #30
        year = md[3] #1969
end
```

实例 23 简单替换

实例 23 简单替换

```
#Convert <br> to <br /> for XHTML compliance

text = 'Hello world. <br>'
regexp = Regexp.new('<br>', Regexp::IGNORECASE)
result = text.sub(regexp, "<br />")
```

实例 24 高难度替换

实例 24 高难度替换

```
#urlify - turn URLs into HTML links
text = 'Check the web site, http://www.oreilly.com/catalog/
regexppr.'
regexp = Regexp.new('
    \b                                  # start at word boundary
    (                                   # capture to \1
    (https?|telnet|gopher|file|wais|ftp) :
                                        # resource and colon
    [\w/#~:.?+=&%@!\-] +?               # one or more valid chars
                                        # take little as possible
    )
    (?= # lookahead [.:?\-] *           # for possible punc
    (?: [^\w/#~:.?+=&%@!\-]             # invalid character
    | $ )                               # or end of string
    )', Regexp::EXTENDED)

result = text.sub(regexp, '<a href="\1">\1</a>')
```

JavaScript

JavaScript 在 1.2 版本中提供了和 Perl 相似的正则表达式支持。本章涵盖 1.5 到 1.7 版本中 ECMS 标准定义的 JavaScript 正则表达式内容。支持的实现包括微软 IE5.5+和 Firefox 1.0++。JavaScript 采用传统的 NFA 匹配引擎。如果想进一步了解传统的 NFA 引擎背后的规则，请看“正则表达式和模式匹配”一节。

支持的元字符

JavaScript 支持表 38 到表 42 中列出来的元字符和元序列。关于每一个元字符的详述，请看“正则表达式元字符、模式和结构”一节。

JavaScript 字符表示	
序列名	序列描述
\o	空字符（Null），\x00
\b	空格，\x08，只有在字符类中有效
\n	换行，\x0A
\r	回车，\x0D
\f	分页，\x0C
\t	水平制表符(tab)，\x09
\xhh	通过 2 个十六进制码指定的字符
\uhhhh	通过 4 个十六进制码指定的字符
\cchar	命名的控制字符

表 38　JavaScript 字符表示

JavaScript 字符类和类似（class-like）结构	
字符类	类描述
[…]	列出来的或包含在列表范围的单一字符
[^…]	不在列出来的或不包含在列表范围的单一字符
.	除行终止之外的任意字符，[^\x0A\x0D\u2028\u2029]
\w	字字符，[a-zA-Z0-9_]
\W	非字字符，　[^a-zA-Z0-9_]
\d	数字字符，　[0-9]
\D	非数字字符，[^0-9]
\s	空格字符
\S	非空格字符

表 39　JavaScript 字符类和类似（class-like）结构

JavaScript 锚和其他 0 宽测试	
序列名	序列描述
^	字符串的开头，或，在多行匹配模式（/m），任意换行之后的位置
$	字符串末尾，或在多行模式（/m），任意换行之前的位置
\b	字边界
\B	非字边界
(?=…)	正向前(Positive lookahead)
(?!…)	负向前(Negative lookahead)

表 40　PHP 锚和其他 0 宽测试

JavaScript 注释和模式转换器	
模式名	模式描述
m	^和$匹配下一个内嵌的行终止符（\n）
i	大小写不敏感匹配

表 41　JavaScript 注释和模式转换器

JavaScript 归组、捕获、条件和控制	
序列	序列描述
(...)	把子模式和捕获子匹配归到\1，\2，...和$1，$2，...
\n	包含第 n 个被捕获的文本
$n	在替换字符串中，包含第 n 个捕获组中匹配的文本
(?:...)	把子模式分组，但是不捕获子匹配
(?>...)	自动分组
...|...	尝试子模式替换
*	匹配 0 或多次
+	匹配 1 次或多次
?	匹配 1 次或 0 次
{n}	匹配精确的 n 次
{n,}	至少匹配 n 次
{x,y}	至少匹配 x 次，最多 y 次
*?	匹配 0 次或多次，但是尽可能少
+?	匹配 1 次或多次，但是尽可能少
??	匹配 0 次或多次，但是尽可能少

表 42　JavaScript 归组、捕获、条件和控制

JavaScript 归组、捕获、条件和控制（续）	
序列	序列描述
{n,}?	至少匹配 n 次，但是尽可能少
{x,y}?	至少匹配 x 次，最多 y 次，但是尽可能少
*+	匹配 0 此或多次，并且从不回溯
++	匹配 1 此或多次，并且从不回溯
?+	匹配 0 此或 1 次，并且从不回溯
{n}+	至少匹配 n 次，并且从不回溯
{n,}+	至少匹配 n 次，并且从不回溯
{x,y}+	至少匹配 x 次，最多匹配 y 次，并且从不回溯

表 42　JavaScript 归组、捕获、条件和控制（续）

模式匹配函数和对象

JavaScript 在 String 对象，以及复杂模式匹配 RegExp 对象中提供了习惯的模式匹配方法。JavaScript 采用反斜杠来转义。因此，正则表达式引擎约定采用双转义（比如，用\\w，而不是\w）。也可以使用正则表达式子面上的语法/pattern/img。

String

Strings 支持模式匹配的四种约定方法。每一个方法使用一个 pattern 参数。这个参数既可以是一个 RegExp 对象，也可以是一个包含正则表达式模式的字符串。

方法

search(*pattern*)

在字符串中进行 *pattern* 匹配，返回第一个匹配子字符串的起始位置或-1。

replace(*pattern*, *replacement*)

搜索 *pattern* 的匹配，并用 *replacement* 替换匹配的子字符串。如果 *pattern* 是全局模式集，那么替换 *pattern* 的所有的匹配。*replacement* 可能包含 *pattern* 中被第 n 个捕获组匹配的文本替换的$n 结构。

match(*pattern*)

在字符串中进行 *pattern* 匹配，返回第一个数组位置或-1。数组中的元素 0 包含全匹配。附加的元素包含捕获组的子匹配。在全局模式（g）中，包含 *pattern* 的所有匹配，但是不包括捕获组织匹配。

split(*pattern*, *limit*)

返回一个被 pattern 切分的字符串的数组。如果包括 *limit*，那么这个数组最多包含 *limit* 个子字符串。如果 *pattern* 包含捕获组，那么捕获的子字在每一个拆分子字符串之后作为元素返回。

RegExp

构建一个正则表达式，并且包含模式匹配的方法。

构造器

new RegExp(*pattern*, *attributes*)
/*pattern*/*attributes*

RegExp 对象既可以通过 REgExp()构造器创建，也可以通过特殊的语法/.../l 来创建。参数 *pattern* 是必须的正则表达式模式，而参数 *attributes* 却是一个可选字符串，包含任意的变换器,g, i, 或 m。参数 *pattern* 也可以是一个 RegExp 对象，但是必须指定 *attributes* 参数。

这个构造器，可能会抛出两个异常。如果 *pattern* 是畸形的，或如果 *attributes* 包含非法的变换器，那么就会抛出 *SyntaxError* 异常。如果 *pattern* 是一个 RegExp 对象，而忽略了 *attributes* 参数，那么就会抛出 *TypeError* 异常。

实例属性

global

标识 RegExp 是否有 *g* 属性的一个 Boolean 属性。

ignorecase

标识 RegExp 是否有 i 属性的一个 Boolean 属性。

lastIndex

最后匹配的字符位置。

multiline

标识 RegExp 是否有 *m* 属性的一个 Boolean 属性。

source

用于创建这个对象的下一个模式。

方法

exec(*text*)

搜索 *text*，如果搜索成功则返回字符串的一个数组，如果失败返回 null。数组的元素 0 被整个正则表达式匹配的子字符串。附加的元素与捕获组有关。

如果设置了全局标识，那么 *lastIndex* 被设置为匹配之后的位置，或者，如果没有匹配则赋值为 0。连续的 exec()或 test()调用将从 lastIndex 开始。注意，*lastIndex* 是正则表达式的一个属性，而不是被搜索的字符串的属性。如果你在全局模式下搜索多个字符串，那么需要手工的重置 *lastIndex*。

test(*text*)

如果 RegExp 对象匹配 *text*，则返回 true。在全局模式下，test()执行和 exec()一样的操作：一系列的调用从 *lastIndex* 开始，即
使被应用到不同的字符串。

实例

实例 25 简单匹配

实例 25 简单匹配

```
//Find Spider-Man, Spiderman, SPIDER-MAN, etc. var dailybugle =
"Spider-Man Menaces City!";

//regex must match entire string
var regex = /spider[- ]?man/i;
if (dailybugle.search(regex)) {
      //do something
}
```

实例 26 匹配和归组

实例 26 匹配和归组

```
//Match dates formatted like MM/DD/YYYY, MM-DD-YY,...
var date = "12/30/1969";
var p = new RegExp("^(\\d\\d)[-/](\\d\\d)[-/](\\d\\d(?:\\d\\ d)?)$");
var result = p.exec(date);
if (result != null) {
      var month = result[1];
      var day = result[2];
      var year = result[3];...
```

实例 27 简单替换

实例 27 简单替换

```
//Convert <br> to <br /> for XHTML compliance

String text = "Hello world. <br>";
var pattern = /<br>/ig;
test.replace(pattern, "<br />");
```

实例 28 高难度替换

实例 28 高难度替换

```
//urlify - turn URLs into HTML links

var text = "Check the web site, http://www.oreilly.com/
catalog/regexppr.";
        var regex =
           “\\b”                                    // start at word boundary
        + "(" // capture to $1
        +"(https?|telnet|gopher|file|wais|ftp) :"   // resource and colon
        +"[\\w/\\#~:.?+=&%@!\\-]+?"                 // one or more valid chars
                                                    // take little as possible
        +")"
        +"(?="                                      // lookahead
        "[.:?\\-]*"                                 // for possible punct
        "(?:[^\\w/\\#~:.?+=&%@!\\-]"                // invalid character
        "|$)"                                       // or end of string
        ")";
text.replace(regex, "<a href=\"$1\">$1</a>");
```

其他资源

- O’Reilly 出版的 David Flanagan 著作《JavaScript 权威指南》是所有 javascript，包括正则表达式在内的参考手册。

PCRE

Perl 兼容的正则表达式（Perl Compatible Regular Expression, PCRE）库，是 Philip Hazel 开发的，对任何人都免费的，开源的 C 语言正则表达式库。PCRE 已经被整合到 PHP、Apache Web Server 2.0、KDE、Exim、Analog 和 Postfix。这些程序的用户可以可以使用表 43 到表 47 中所列出的 PCRE 支持的元字符。

PCRE 采用传统的 NFA 匹配引擎。如果想进一步了解传统的 NFA 引擎背后的规则，请看“正则表达式和模式匹配”一节。

本节涵盖 PCRE 7.0 正则表达式相关的知识。这个版本的目标是仿效 Perl 5.8 风格的正则表达式，但是也包含了一些直到 Perl 5.10 采用的一些功能和特性。

无论是支持或不支持 UTF-8，还是支持或不支持 Unicode 属性 PCRE 都能被编译。以下的列表和图表假设这两个功能都有效。

支持的元字符

PCRE 支持表 43 到表 47 中列出来的元字符和元序列。关于每一个元字符的详述，请看“正则表达式元字符、模式和结构”一节。

PCRE 字符表示	
序列名	序列描述
\a	告警（bell），\x07
\b	空格，\x08，只有在字符类中有效
\e	Esc 字符，\x1B
\n	换行，\x0A
\r	回车，\x0D
\f	分页，\x0C
\t	水平制表符(tab)，\x09
\0octal	通过 1、2 或 3 个八进制码数指定的字符
\xhex	通过 1 个或 2 个十六进制数指定的字符
\x{hex}	通过十六进制码指定的字符
\cchar	命名的控制字符
\p{prop}	包含被定 Unicode 区位或属性的字符
\P{prop}	不包含被定 Unicode 区位或属性的字符

表 43　PCRE 字符表示

PCRE 字符类和类似（class-like）结构	
字符类	类描述
[…]	列出来的或包含在列表范围的单一字符
[^…]	不在列出来的或不包含在列表范围的单一字符
[:*class*:]	POSIX 风格的字符类（只有在 regex 字符类中有效）
.	除行终止（除非是单行模式，PCRE_DOTALL）之外的任意字符
\C	1 字节（然而，这可能会破坏 Unicode 字符流）
\w	字字符，[a-zA-Z0-9_]
\W	非字字符， [^a-zA-Z0-9_]
\d	数字字符， [0-9]
\D	非数字字符，[^0-9]
\s	空格字符，[\n\r\f\t\v]
\S	非空格字符，[^ \n\r\f\t\v]

表 44　PCRE 字符类和类似（class-like）结构

PCRE 锚和其他 0 宽测试	
序列名	序列描述
^	字符串的开头，或，在多行匹配模式（PCRE_MULTILINE），任意换行之后的位置
\A	在任意匹配模式(mode)，搜索字符串的开头
$	字符串末尾，或在多行模式（PCRE_MULTILINE），任意换行之前的位置
\Z	在任意匹配模式下，字符串末尾或字符串末尾换行之前的位置
\z	任意匹配模式下，字符串末尾
\G	当前匹配的开头
\b	\w 和\W 之间的字边界
\B	非字边界
(?=…)	正向前(Positive lookahead)
(?!…)	负向前(Negative lookahead)
(?<=…)	正向后(Positive lookbehind)
(?<!...)	负向后(Negative lookbehind)

表 45　PCRE 锚和其他 0 宽测试

PCRE 注释和模式转换器		
模式转换器/序列	模式名	模式转换器描述
PCRE_CASELESS	i	当码点（codepoints）小于 256 时，大小写不敏感匹配。当 Unicode 属性得到支持时，对 UTF-8 模式有效。
PCRE_MULTILINE	m	^和$匹配下一个内嵌的行终止符（\n）
PCRE_DOTALL	s	点号匹配换行
PCRE_EXTENDED	x	忽略空格，并允许以#开头的注释
PCRE_UNGREEDY	U	转换计量器的贪婪性：* 变成懒惰，而 *?成为贪婪
PCRE_ANCHORED		迫使匹配从主体字符长的起始开头
PCRE_DOLLAR_ENDONLY		迫使$匹配字符串的末尾，而不是换行符之前这将被多行模式重写。
PCRE_NO_AUTO_CAPUTRE		禁止括号捕获函数
PCRE_UTF8		把正则表达式和主题字符串视为 UTF-8 字符
PCRE_AUTO_CALLOUT		插入自动标注
PCRE_DUPNAMES		允许重复命名组
PCRE_FIRSTLINE		锚的模式必须匹配主体第一个换行之前
PCRE_NEWLINE_CR PCRE_NEWLINE_LF PCRE_NEWLINE_CRLF PCRE_NEWLINE_ANY		指定换行符序列
PCRE_NOTBOL		主体的提点，而不是行的开头
PCRE_NOTEOL		主体的终点，而不是行的末尾
PCRE_NOTEMPTY		一个空行不是一个合法的匹配
PCRE_NO_UTF8_CHECK		UTF-8 字符串非法
PCRE_PARTIAL		当到达输入字符串末尾而匹配失败的时候返回 PCRE_PARTIAL 而不是 PCRE_NOT_MATCH
（?*mode*）		为其余子表达式启动所列的模式（一个或多个 imsxU）
（?-*mode*）		为其余子表达式取消所列的模式（一个或多个 imsxU）
（?*mode*:...）		启动括号内所列的模式（一个或多个 imsx）
（?-*mode*:...）		取消括号内所列的模式（一个或多个 imsx）
\E		由\Q 开头的 span 的末尾
\Q		引掉紧接的所有正则表达式元字符
（?#...）		把子字符串当作注释
#...		在PCRE_EXTENDED模式中，把行的剩余部分当作注释

表 46　PCRE 注释和模式转换器

PCRE 归组、捕获、条件和控制	
序列	序列描述
(...)	把子模式和捕获子匹配归到\1，\2，...和$1，$2，...
(?P<*name*>...), (?<name>),(?'name')	在替换字符串中，包含第 n 个捕获组中匹配的文本
(?P=name), \k<name>, \k'name'	回引（backreference）一个命名捕获
\n, \gn, \g{n}	包含从括号捕获组或命名捕获组中第 n 个子匹配的结果
(?:...)	把子模式分组，但是不捕获子匹配
(?>...)	自动分组
...|...	尝试子模式替换
*	匹配 0 或多次
+	匹配 1 次或多次
?	匹配 1 次或 0 次
{n}	匹配精确的 n 次
{n,}	至少匹配 n 次
{x,y}	至少匹配 x 次，最多 y 次
*?	匹配 0 次或多次，但是尽可能少
+?	匹配 1 次或多次，但是尽可能少
??	匹配 0 次或多次，但是尽可能少
{n,}?	至少匹配 n 次，但是尽可能少
{x,y}?	至少匹配 x 次，最多 y 次，但是尽可能少
*+	匹配 0 此或多次，并且从不回溯
++	匹配 1 此或多次，并且从不回溯
?+	匹配 0 此或 1 次，并且从不回溯
{n}+	至少匹配 n 次，并且从不回溯
{n,}+	至少匹配 n 次，并且从不回溯
{x,y}+	至少匹配 x 次，最多匹配 y 次，并且从不回溯
(?(*condition*)...|...)	匹配 if-then-else 模式。*condition* 可以是捕获组、回环或向后结构的个数
(?(*condition*)...)	匹配 if-then-else 模式。*condition* 可以是捕获组、回环或向后结构的个数

表 47　PCRE 归组、捕获、条件和控制

PCRE API

使用 PCRE 的应用程序需要通过 pcre.h 来察看 API 原型，并通过-lpcre 编译引入准确的库文件，libpcre.a。

绝大多数的功能都包含在函数 pcre_compile()中，它预备一个正则表达式的数据结构，和 pcre_exec()，它执行模式匹配。虽然 PCRE 提供了 pcre_free_substring()和 pcre_free_substring_list()函数，但是你必须负责释放内存。

PCRE API Synopsis

PCRE API 概述

pcre *pcre_compile(const char **pattern*, int *options*, const char ***errptr*, int **erroffset*,
const unsigbed char **tableptr*)

通过可选模式变换器 *options* 和由 *pcre_maketables*()创建的 *tableptr*，来编译 *pattern*。返回编译之后的 *regex* 或 NULL 和 *errptr* 指向的错误消息，以及 *erroffset* 指向的 *pattern* 中错误的发生的位置。

int pcre_exec(const pcre **code*, const pcre_extra **extra*, const char **subject*, int *length*,
int *startoffset*, int *options*, int **ovector*, int *ovecsize*)

在已经编译的正则表达式，*code* 和提供的 *length* 长度的输入字符串，*subject* 上执行模式匹配。成功匹配的结果存储在 *ovector* 中。在 *ovector* 中的第一个和第二个元素包含总匹配的第一个字符和紧跟总匹配末尾的字符。每一个附加的元素对，直到 *ovector* 长度的三分之二，包含起始字符的位置和捕获组子匹配之后的字符。可选的参数 *options*，包含模式变换器，pcre_extra 包含一个 pcre_study()调用的结果。

pcre_extea *pcre_study(const pcre **code*, int *options*, const char ***errptr*)

返回加快通过 *code* 对 pcre_exec()调用的信息。因为目前没有提供可选项，所以需要置 *options* 为 0。如果发生了一个错误，那么 *errptr* 指向一个错误的消息。

int pcre_copy_named_sustring(const pcre **code*, const char **subject*, int **ovector*,
int *stringcount*, const char **stringname*, char **buffer*,
int *buffersize*)

把被命名的捕获组匹配的子字符串 *stringname* 拷贝到 *buffer* 中。*stringcount* 是替换到 *ovector* 中的子字符串的个数，通常是 pcre_exec()返回的结果。

int pcre_copy_sustring(const char **subject*, int **ovector*,int *stringcount*,
const char **stringname*, char **buffer*, int *buffersize*)

把被命名的捕获组匹配的子字符串 *stringname* 拷贝到 *buffer* 中。*stringcount* 是替换到

ovector 中的子字符串的个数，通常是 pcre_exec()返回的结果。

int pcre_get_named_sustring(const pcre *_code_, const char *_subject_, int *_ovector_,
int _stringcount_, const char *_stringname_, const char **stringptr)

创建一个新的字符串，指向包含被命名捕获组 *stringname* 匹配的子字符串的 *stringptr*。返回子字符串的长度。*stringcount* 是替换到 *ovector* 中的子字符串的个数，通常是 pcre_exec()返回的结果。

int pcre_get_stringnumber(const pcre * _code_, const char *_name_)

返回与命名捕获组 *name* 有关的捕获组的个数。

int pcre_get _sustring(const char *_subject_, int *_ovector_,
int _stringcount_, const char *_stringnumber_, sonst char **stringptr)

创建一个新的字符串，指向包含被计数的捕获组 *stringname* 匹配的子字符串的 *stringnumber*。返回子字符串的长度。*stringcount* 是替换到 *ovector* 中的子字符串的个数，通常是 pcre_exec()返回的结果。

int pcre_get_substring_list(const char *_subject_, int *_ovector_, int _stringcount_,
const char ***_listptr_)

返回指向所有被捕获的子字符串的指针列表，*listptr*。

void pcre_fress_substring(cinst char *_stringptr_)

释放指向 *stringptr*，以及被 pcre_get_named_substring()或 pcre_get_substring_list()分配的内存。

void pcre_free_substring_list(const char **_stringptr_)

释放指向 *stringptr*，以及被 pcre_get_substring_list()分配的内存。

const unsigned char *pcre_maketables(*void*)

为当前位置创建字符表。

int pcre_fullinfo(const pcre *_code_, const pcre_extra *_extra_, int *what*, void *_where_)

放置由 *what* 指定到 *where* 的正则表达式的信息。*what* 可以取的值有：
PCRE_INFO_BACKREFMAX, PCRE_INFO_ CAPTURECOUNT, PCRE_INFO_FIRSTBYTE, PCRE_INFO_FIRSTTABLE, PCRE_INFO_LASTLITERAL, PCRE_INFO_NAMECOUNT, PCRE_INFO_ NAMEENTRYSIZE, PCRE_INFO_NAMETABLE, PCRE_INFO_OPTIONS, PCRE_ INFO_SIZE, and PCRE_INFO_STUDYSIZE。

int pcre_config(int what, void *_where_)

放置由 *what* 指定到 *where* 的 build-time 可选项。*what* 可以取的值有：
PCRE_CONFIG_UTF8, PCRE_ CONFIG_NEWLINE, PCRE_CONFIG_LINK_SIZE, PCRE_CONFIG_POSIX_ MALLOC_THRESHOLD, and PCRE_CONFIG_MATCH_LIMIT.

char *pcre_version(*void*)

返回一个指向包含 PCRE 版本和发布日期的字符串。

void *(**pcre_malloc*)(size_t)

PCRE 进行 malloc()调用的入口点。

void (*pcre_free)(void *)

PCRE 进行 pcre_free()调用的入口点。

int (*pcre_callout)(pcre_callout_block *)

可以被设置给一个匹配过程中，将被调用的标注函数。

Unicode 支持

PCRE 提供了基础 Unicode 5.0 的支持。当一个模式用 PCRE_UTF8 标识编译，那么这个模式将使用 Unicode 文本。然而，PCRE 基于一个默认的表检测字符的大小写和属性，来确定是一个字母还是数字。可以通过不同位置的表来替换它。例如：

```
setlocale(LC_CTYPE, " fr " ) ;
tables = pcre_maketables() ;
re = pcre_compile(..., tables) ;
```

实例

实例 29 简单匹配

实例 29 简单匹配

```
#include <stdio.h>
#include <string.h>
#include <pcre.h>
#define CAPTUREVECTORSIZE 30        /* should be a multiple of 3 */
int main(int argc, char **argv)
{
    pcre *regex;
    const char *error;
    int erroffset;
    int capturevector[CAPTUREVECTORSIZE];
    int rc;
```

```
    char *pattern = "spider[- ]?man";
    char *text ="SPIDERMAN menaces city!";

    /* Compile Regex */
    regex = pcre_compile(
                pattern,
                PCRE_CASELESS,          /* OR'd mode modifiers */
                &error,                 /* error message */
                &erroffset,     /* position in regex where error occurred */
                NULL);          /* use default locale */
    /* Handle Errors */
    if (regex = = NULL)
    {
        printf("Compilation failed at offset %d: %s\n", erroffset,error);
        return 1;
    }

    /* Try Match */
    rc = pcre_exec(
        regex,                  /* compiled regular expression */
        NULL,                   /* optional results from pcre_study */
        text,                   /* input string */
        (int)strlen(text),      /* length of input string */
        0,                      /* starting position in input string */
        0,                      /* OR'd options */
        capturevector,          /* holds results of capture groups */
        CAPTUREVECTORSIZE);

    /* Handle Errors */
    if (rc < 0) { switch(rc)
    {
        case PCRE_ERROR_NOMATCH:
            printf("No match\n");
            break;
        default:
            printf("Matching error %d\n", rc);
            break;
    }
    return 1;
    }
return 0;
}
```

实例 30 匹配和归组

实例 30 匹配和归组

```
#include <stdio.h>
#include <string.h>
#include <pcre.h>

#define CAPTUREVECTORSIZE 30                /* should be a multiple of 3 */
int main(int argc, char **argv)
{
    pcre *regex;
    const char *error;
    int erroffset;
    int capturevector[CAPTUREVECTORSIZE];
    int rc, i;
    char *pattern = "(\\d\\d)[-/](\\d\\d)[-/](\\d\\d(?:\\d\\d)?)";
    char *text ="12/30/1969";

    /* Compile the Regex */
    re = pcre_compile(
        pattern,
        PCRE_CASELESS,          /* OR'd mode modifiers */
        &error,                 /* error message */
        &erroffset,             /* position in regex where error occurred */
        NULL);                  /* use default locale */

        /* Handle compilation errors */
    if (re = = NULL)
    {
        printf("Compilation failed at offset %d: %s\n",erroffset, error);
        return 1;
     }

    rc = pcre_exec( regex, /* compiled regular expression */
        NULL,                   /* optional results from pcre_study */
        text,                   /* input string */
        (int)strlen(text),      /* length of input string */
        0,                      /* starting position in input string */
        0,                      /* OR'd options */
        capturevector,          /* holds results of capture groups */
        CAPTUREVECTORSIZE);

    /* Handle Match Errors */
    if (rc < 0) { switch(rc)
```

```
    {
        case PCRE_ERROR_NOMATCH:
            printf("No match\n");
            break;
        /*
        Handle other special cases if you like
        */
        default: printf("Matching error %d\n", rc); break;
    }
    return 1;                                   /* Match succeded */
    printf("Match succeeded\n");

    /* Check for output vector for capture groups */
    if (rc = = 0)
    {
        rc = CAPTUREVECTORSIZE/3;
        printf("ovector only has room for %d captured substrings\n",rc - 1);
    }

    /* Show capture groups */
    for (i = 0; i < rc; i++)
    {
        char *substring_start = text + ovector[2*i];
        int substring_length = capturevector[2*i+1]-capturevector[2*i];
        printf("%2d: %.*s\n", i, substring_length, substring_start);
    }
    return 0;
}
```

其他资源

- PCRE 的 C 源代码和在线帮助文档, http://www.pcre.org。

Apache Web Server

Apache web server 2.0 采用了基于 PCRE 库的 Perl 风格正则表达式。Apache web server 2.0 采用传统的 NFA 匹配引擎。如果想进一步了解传统的 NFA 引擎背后的规则，请看“正则表达式和模式匹配”一节。

Apache 有很多的命令正则表达式。本章涵盖 Apache 2.2（和 2.0 最兼容）和最常用的命令：RewriteRule、LocationMatch、DirectoryMatch、FilesMatch、ProxyMatch 和 AliasMatch。

支持的元字符

Apache 支持表 48 到表 52 中列出来的元字符和元序列。关于每一个元字符的详述，请看“正则表达式元字符、模式和结构”一节。

Apache 字符表示	
序列名	序列描述
\octal	通过 3 个八进制码数指定的字符
\xhex	通过 1 个或 2 个十六进制数指定的字符
\x{hex}	通过十六进制码指定的字符
\cchar	命名的控制字符

表 48　Apache 字符表示

Apache 字符类和类似（class-like）结构	
字符类	类描述
[…]	列出来的或包含在列表范围的单一字符
[^…]	不在列出来的或不包含在列表范围的单一字符
[:*class*:]	POSIX 风格的字符类（只有在 regex 字符类中有效）
.	除行终止（除非是单行模式，/s）之外的任意字符
\C	1 字节（然而，这可能会破坏 Unicode 字符流）
\w	字字符，[a-zA-Z0-9_]
\W	非字字符，　[^a-zA-Z0-9_]
\d	数字字符，　[0-9]
\D	非数字字符，[^0-9]
\s	空格字符，[\n\r\f\t]
\S	非空格字符，[^ \n\r\f\t]

表 49　Apache 字符类和类似（class-like）结构

Apache 锚和其他 0 宽测试	
序列名	序列描述
^	字符串的开头
$	字符串末尾，或在多行模式（/m），任意换行之前的位置
\b	\w 和\W 之间的字边界
\B	非字边界
(?=…)	正向前(Positive lookahead)
(?!…)	负向前(Negative lookahead)
(?<=…)	正向后(Positive lookbehind)
(?<!...)	负向后(Negative lookbehind)

表 50　Apache 锚和其他 0 宽测试

Apache 注释和模式转换器	
模式名	模式描述
NC	大小写不敏感匹配
(?*mode*)	为其余子表达式启动所列的模式（一个或多个 imsxU）
(?-*mode*)	为其余子表达式取消所列的模式（一个或多个 imsxU）
(?*mode*:...)	启动括号内的模式（xsmi）
(?-*mode*:...)	取消括号内的模式（xsmi）
(?#...)	把子字符串当作注释
#...	在/x 模式中，把行内剩余部分当作注释
\Q	引掉（注释掉）后续的正则表达式
\E	终止一个从\Q 开始的 span

表 51　Apache 注释和模式转换器

Apache 归组、捕获、条件和控制	
序列	序列描述
(...)	把子模式和捕获子匹配归到\1，\2...
(?P<*name*>...)	在替换字符串中，包含第 n 个捕获组中匹配的文本
\n	包含第 n 个被捕获的文本
(?:...)	把子模式分组，但是不捕获子匹配
(?>...)	自动分组
...|...	尝试子模式替换
*	匹配 0 或多次
+	匹配 1 次或多次
?	匹配 1 次或 0 次
{n}	匹配精确的 n 次
{n,}	至少匹配 n 次
{x,y}	至少匹配 x 次，最多 y 次
*?	匹配 0 次或多次，但是尽可能少
+?	匹配 1 次或多次，但是尽可能少
??	匹配 0 次或多次，但是尽可能少
{n,}?	至少匹配 n 次，但是尽可能少
{x,y}?	至少匹配 x 次，最多 y 次，但是尽可能少
*+	匹配 0 此或多次，并且从不回溯
++	匹配 1 此或多次，并且从不回溯
?+	匹配 0 此或 1 次，并且从不回溯
{n}+	至少匹配 n 次，并且从不回溯
{n,}+	至少匹配 n 次，并且从不回溯
{x,y}+	至少匹配 x 次，最多匹配 y 次，并且从不回溯
(?(*condition*)...|...)	匹配 if-then-else 模式。*condition* 可以是捕获组、回环或向后结构的个数
(?(*condition*)...)	匹配 if-then-else 模式。*condition* 可以是捕获组、回环或向后结构的个数

表 52　Apache 归组、捕获、条件和控制

RewriteRule

重写引擎强化了基于正则表达式的 URLs 重写。这个功能通过 RewriteEngine On 定向来实现。绝大多数的重写是单行的 RewriteRule 或一个 RewriteCond 紧跟一个 RewriteRule 的联合。

RewriteRule pattern substitution [[*FLAG1*, *FLAG2*, ...]]

如果 URL 被 *pattern* 成功匹配，那么把 URL 重写为 *substitution*。*substitution* 可以包含 *RewriteRule* 模式的回引（$N），最后匹配的 *RewriteCond* 模式的回引（*%N*），以及规则中条件测试字符串（test-strings）中的一些变量（%{*VARNAME*}），和映射函数调用（${*mapname:key|default*}）。表 53 中列出的可选标识，当匹配发生时引发服务器的一系列行为。

ReeorteCond teststring pattern

为了应用一个 *RewriteRule*，定义一个测试条件（表 54）。*RewriteRule* 前面的多个 *RewriteCond* 之间通过隐含的 AND 来连接，除非被指定为 *OR*。 *teststring* 可以包含 *RewriteRule* 模式的回引（$N），最后匹配的 *RewriteCond* 模式的回引（*%N*），以及规则中条件测试字符串（test-strings）中的一些变量（%{*VARNAME*}），和映射函数调用（${*mapname:key|default*}）。

表 55 中列出了对重写有影响的一些变量。

Apache RewriteRule 标识	
模式	模式描述
C	下一个规则的链(chain)。如果规则匹配，则实行重写，以及紧接的被链的重写，否则中止链。
CO=NAME:VAL:*domain* [:lifetime[:path]]	设置 cookie。
E=*VAR:VAL*	设置一个环境变量
F	禁止；发送 403 代码
G	已经转移；发送 401 代码
H=*Content-handler*	发送内容的操作者
L	最后的规则；不使用任何更多的重写规则。
N	下一个规则；在新的重写 URL 中使用这个规则。

表 54　Apache RewriteRule 标识

Apache RewriteRule 标识（续）	
模式	模式描述
NC	没有大小写；使用大小写不敏感的匹配。
NE	让 URL 转义规则的应用程序不能对规则的输出进行转义
NS	如果请求是内部的子请求，则跳过处理
P	停止重写处理，转而处理一个内部代理请求的结果
PT	转移到下一个操作者，设置请求结构，这样 Alias,ScriptAlias 和 Redirect 可可以作用于结果。
QSA	填充查询字符串
R[=*Code*]	重定向到一个新的 URL 可选码。默认码为 302。
S=*num*	跳过后续的 *num* 个规则。
T=*MIME-type*	设置 MIME 类型

表 54　Apache RewriteRule 标识（续）

Apache RewriteCond 标识	
模式	模式描述
NC	没有大小写；使用大小写不敏感的匹配。
OR	用一个局部的 OR 来连接规则条件，而不是隐含的 AND。

表 55 Apache RewriteCond 标识

Apache Server 变量	
HTTP 头部	连接和请求
HTTP_USER_AGENT	REMOTE_ADDR
HTTP_REFERER	REMOTE_HOST
HTTP_COOKIE	REMOTE_PORT
HTTP_FORWARDED	REMOTE_USER
HTTP_HOST	REMOTE_IDENT
HTTP_PROXY_CONNECTION	REQUEST_METHOD
HTTP_ACCEPT	SCRIPT_FILENAME
Server internals	PATH_INFO
DOCUMENT_ROOT	AUTH_TYPE

表 56　Apache Server 变量

Apache Server 变量（续）	
HTTP 头部	连接和请求
SERVER_ADMIN	**Date and time**
SERVER_ADDR	TIME_YEAR
SERVER_PORT	TIME_MON
SERVER_PROTOCOL	TIME_DAY
Special	TIME-MIN
API_VERSION	*TIME_WDAY*
THE_REQUEST	*TIME*
REQUEST_URI	
REQUEST_FILENAME	
IS_SUBREQ	
HTTPS	

表 56　Apache Server 变量（续）

Matching Directives

另外一些 Apache 命令有效的利用了正则表达式。以下是最通用的几个命令。

AliasMatch pattern file-path|directory-path

把 URLs 映射到文件系统的一个位置。使用子匹配变量$1...$*n* 来访问文件路径结果中的子匹配。

<DirectoryMath *pattern*>...</DirectiryMatch>

当文件系统目录匹配 *pattern* 时，使用最近的一个定向。

<FilesMatch *pattern*>...</FilesMatch>

当文件匹配 *pattern* 时，使用最近的一个定向。

<LocationMatch *pattern*>...</LocationMatch>

当 URL 匹配 *pattern* 时，使用最近的一个定向。

<ProxyMatch *pattern*>...</ProxyMatch>

当 URL 匹配 *pattern* 时，使用最近的一个定向。

实例

实例 31　简单匹配

实例 31 简单匹配

```
# Rewrite /foo to /bar
RewriteEngine On
RewriteRule ^/foo$ /bar
```

实例 32 匹配和归组

实例 32 匹配和归组

```
# Rewrite pretty url as script parameters
RewriteRule ^/(\w+)/(\d+) /index.php?action=$1&id=$2
```

实例 33 重写条件

实例 33 重写条件

```
# Limit admin url to internal IP addresses

RewriteCond %{REMOTE_ADDR} !192.168.\d*.\d*
RewriteCond %{PATH_INFO} ^admin
RewriteRule .* - [F]
```

实例 34 重定向

实例 34 重定向

```
# Make sure admin urls are served over SSL
RewriteCond %{SERVER_PORT} !^443$
RewriteRule ^/admin/(.*)$ https://www.example.com/admin/$1
[L,R]
```

Vi Editor

vi 程序是 Unix 系统上主流的文本编辑器。Vim 是扩展了正则表达式支持的流行 vi 编辑器。两个都采用 DFA 匹配引擎。如果想进一步了解传统的 DFA 引擎背后的规则，请看“正则表达式和模式匹配”一节。

支持的元字符

vi 支持表 56 到表 60 中列出来的元字符和元序列。关于每一个元字符的详述，请看“正则表达式元字符、模式和结构”一节。

vi 字符表示	
序列名	序列描述
仅 Vim（文本编辑器）支持	
\b	空格，\x08
\e	Esc 字符，\x1B
\n	换行，\x0A
\r	回车，\x0D
\t	水平制表符(tab)，\x09

表 56　vi 字符表示

vi 字符类和类似（class-like）结构	
字符类	类描述
[…]	列出来的或包含在列表范围的单一字符
[^…]	不在列出来的或不包含在列表范围的单一字符
[:*class*:]	POSIX 风格的字符类（只有在 regex 字符类中有效）
.	除行终止（除非是单行模式，/s）之外的任意字符
仅 Vim 支持	
\w	字字符，[a-zA-Z0-9_]
\W	非字字符，[^a-zA-Z0-9_]
\a	字母，[a-zA-Z]
\A	非字母，[^a-zA-Z]
\h	字字符的开头，[a-zA-Z_]
\H	非字字符的开头，[^a-zA-Z_]
\d	数字字符，[0-9]

表 57　vi 字符类和类似（class-like）结构

vi 字符类和类似（class-like）结构（续）	
字符类	类描述
\D	非数字字符，[^0-9]
\s	空格字符，[\t]
\S	非空格字符，[^ t]
\x	十六进制数，[a-fA-F0-9]
\X	非十六进制数，[^a-fA-F0-9]
\o	八进制数，[0-7]
\O	非八进制数，[^0-7]
\l	小写字母，[a-z]
\L	非小写字母，[^a-z]
\u	大写字母，[A-Z]
\U	非大写字母，[^A-Z]
\i	由 isident 定义的标识字符
\I	任意非数字的标识字符
\k	通常由语言模型设置，由 iskeyword 定义的键盘字符
\K	任意非键盘字符
\f	由 isfname 定义的文件名字符。依赖具体操作系统的实现。
\F	任意非数字的文件名字符
\p	由 isprint 定义的可打印字符。通常是 x20-x7E

表 57　vi 字符类和类似（class-like）结构（续）

vi 锚和其他 0 宽测试	
序列名	序列描述
^	出现在正则表达式起始位置时，表示行的开头，否则匹配自身
$	出现在正则表达式末尾位置时，表示行的末尾，否则匹配自身
\<	字边界的起始（比如，一个符号或空格字符和一个字字符）
\>	字边界的末尾

表 58　vi 锚和其他 0 宽测试

vi 注释和模式转换器	
模式名	模式描述
:set ic	为所有的搜索和替换启动大小写敏感模式
:set noic	取消（关闭）大小写敏感模式
\u	强制替换字符串中下一个字符为大写
\l	强制替换字符串中下一个字符为小写
\U	强制替换字符串中所有字符为大写
\L	强制替换字符串中所有字符为小写
\E 或\e	终止一个从\U 或\L 开始的 span

表 59　vi 注释和模式转换器

vi 归组、捕获、条件和控制	
序列	序列描述
\(...\)	把子模式和捕获子匹配归到\1，\2，...
\n	包含第 n 个早被捕获的文本。在 regex 模式或替换字符串中能够有效
*	匹配 0 或多次
仅 Vim 支持	
+	匹配 1 次或多次
\=	匹配 1 次或 0 次
{n}	匹配精确的 n 次
{n,}	至少匹配 n 次
{,n}	最多匹配 n 次
{x,y}	至少匹配 x 次，最多 y 次

表 60　vi 归组、捕获、条件和控制

模式匹配

搜索

/pattern ?pattern

光标移动到文件中被 *pattern* 匹配的下一个位置。一个 ?*pattern* 从后向前搜索。前向搜索时可用 n 键，重复搜索操作，但后向搜索时用 N 键来执行重复搜索。

替换

:[*addr1*[,*addr2*]]s/*pattern*/*replacement*/[*cgp*]

在 *address* 范围内用 *replacement* 来替换被 *pattern* 匹配的文本。如果没有指定位置范围（*address*），那么默认为当前行。每一个 *address* 既可以是一个行号，也可以是通用的行。如果指定了 *addr1*，那么替换从指定的行开始（或者第一个匹配的行），到文件末尾为止，或由 *addr2* 指定的行（或匹配的）。下表是 *address* 的一些位置速记符。

替换选项	
选项	选项描述
C	每个替换之前的提示
g	替换一行内所有的匹配
p	打印替换之后的行

位置速记符	
位置	相关描述
.	当前行
$	文件最后一行
%	整个文件
‘t	位置 t
/...[/]	下一个被 pattern 匹配的行
?...[?]	前一个被 pattern 匹配的行
\/	下一个被最后搜索匹配的行
\?	前一个被最后搜索匹配的行
\&	下一个替换模式匹配的行

实例

实例 35 vi 中简单搜索

实例 35 vi 中简单搜索

```
Find spider-man, Spider-Man, Spider Man

/[Ss]pider[- ][Mm]an
```

实例 36 Vim 中简单搜索

实例 36 Vim 中简单搜索

```
Find spider-man, Spider-Man, Spider Man, spiderman, SPIDERMAN, etc.

:set ic /spider[- ]\=man
```

实例 37 vi 中简单替换

实例 37 vi 中简单替换

```
Globally convert <br> to <br /> for XHTML compliance.

:set ic
: % s/<br>/<br \/>/g
```

实例 38 Vim 中简单替换

实例 38 Vim 中简单替换

Globally convert
 to
 for XHTML compliance.

: % s/
/<br \/>/ig

实例 39 Vim 中高难度单替换

实例 39 Vim 中高难度单替换

Urlify: Turn URLs into HTML links

: % s/\(https\=:\/\/[a-z_.\\w\/\\#~:?+=&;%@!-]*\)/< a href=" \1">\1<\/a>/ic

其他资源

- O’Reilly 出版 Linda Lamb 的著作《学习 vi 编辑器》第六版，是 vi 编辑器和流行的 vi 克隆编辑器的相关参考手册。
- Oleg Raisky, 有关 Vim 正则表达式语法概述, http://www.geocities.com/volontir/。

Schell Tools

awk, sed 和 egrep 是 Unix 中文本处理相关的 Shell 工具。awk 采用了 DFA 匹配引擎。egrep 依据说使用的功能在 DFA 和 NFA 引擎之间进行切换（实现了两中引擎）。sed 采用传统的 NFA 匹配引擎。如果想进一步了解传统的 NFA 引擎背后的规则，请看“正则表达式和模式匹配”一节。

本章涵盖 GNU egrep 2.4.2，一个文本行搜索程序；GNU sed 3.2，一个脚本编辑命令工具；和 GNU awk 3.1，一种文本处理语言。

支持的元字符

awk, sed 和 egrep 支持表 61 到表 65 中列出来的元字符和元序列。关于每一个元字符的详述，请看“正则表达式元字符、模式和结构”一节。

Shell 字符表示		
序列名	序列描述	支持工具
\a	告警（bell）	awk, sed
\b	空格，只有在字符类中有效	awk,
\f	分页	awk, sed
\n	换行	awk, sed
\r	回车	awk, sed
\t	水平制表符	awk, sed
\v	垂直制表符	awk, sed
\ooctal	通过 1、2 或 3 个八进制码数指定的字符	sed
\octal	通过 1、2 或 3 个八进制码数指定的字符	awk
\xhex	通过 1 个或 2 个十六进制数指定的字符	awk, sed
\x{hex}	通过十六进制码指定的字符	awk, sed
\ddecimal	通过 1 个或 2 个十进制数指定的字符	awk, sed
\cchar	命名的控制字符	sed
\b	空格	awk
\matecharacter	转义元字符，所以它仅表示自身	awk, sed, egrep

表 61　Shell 字符表示

Shell 字符类和类似（class-like）结构		
字符类	类描述	支持工具
[…]	列出来的或包含在列表范围的单一字符	awk, sed, egrep
[^…]	不在列出来的或不包含在列表范围的单一字符	awk, sed, egrep
.	除换行之外的任意字符	awk, sed, egrep
\w	字字符，[a-zA-Z0-9_]	egrep, sed
\W	非字字符， [^a-zA-Z0-9_]	egrep, sed
[:*prop*:]	匹配 POSIX 字符集内的任意字符	awk, sed
[^[:pro*p*:]]	匹配不在 POSIX 字符集内的任意字符	awk, sed

表 62　Shell 字符类和类似（class-like）结构

Shell 锚和其他 0 宽测试		
序列名	序列描述	支持工具
^	匹配字符串的开头，即时有内切的换行	awk, sed, egrep
$	匹配字符串末尾，即时有内切的换行	awk, sed, egrep
\<	匹配字边界的起始位置	egrep
\>	匹配字边界的末尾	egrep

表 63　Shell 锚和其他 0 宽测试

Shell 注释和模式转换器		
模式名	模式描述	支持工具
flag:i 或 I	ASCII 字符大小写不敏感匹配	sed
命令行选项：-i	ASCII 字符大小写不敏感匹配	egrep
set IGNORECASE 为非 0（none-zero）	Unicode 字符大小写不敏感匹配	awk

表 64　hell 注释和模式转换器

Shell 归组、捕获、条件和控制		
序列	序列描述	支持工具
(*PATTERN*)	归组	awk
(*PATTERN*\)	组和捕获字匹配，填写\1,\2,...,\9	sed
\n	包含第 n 个被捕获子匹配	sed
...|...	替换；匹配一个或其他	awk, sed, egrep
贪婪计量器		
*	匹配 0 或多次	awk, sed, egrep
+	匹配 1 次或多次	awk, sed, egrep
?	匹配 1 次或 0 次	awk, sed, egrep
\{n\}	匹配精确的 n 次	sed, egrep
\{n,\}	至少匹配 n 次	sed, egrep
\{x,y\}	至少匹配 x 次，最多 y 次	sed, egrep

表 65　Shell 归组、捕获、条件和控制

egrep

grep [*options*] *pattern files*

grep 在文件搜索 *pattern* 发生的位置，并打印出每一个匹配的行。

实例

grep 实例

```
$ echo 'Spiderman Menaces City!' > dailybugle.txt
$ egrep -i 'spider[- ]?man' dailybugle.txt
Spiderman Menaces City!
```

sed

sed '[*address1*][,*address2*]s/*pattern*/*replacement*/[*flags*]' *files*

sed -f *script files*

默认的情况下 sed 把替换应用于 *files* 中的每一行。每一个 *address* 既可以是一个行号，也可以是一个正则表达式。当在正则表达式中使用时，必须在双方斜杆之内定义（/.../）。

如果指定了 *address1*，那么替换从指定的行开始,或者第一个匹配的行，到文件末尾为止，或由 *address2* 指定或匹配的行。&和\n 将在 *replacement* 中作为匹配结果被解释。

&被 *pattern* 匹配的文本替换。\n 与当前匹配中捕获组（1...9）有关。以下是可用的标识：

n　替换一行内第 *n* 个匹配，*n* 在 1 到 512 之间。

g　替换一行中所发生的所有 *pattern*。

p　打印所有成功替换的行。

w *file*　把成功的替换写到文件中。

实例

把时间格式从 MM/DD/YYYY 改为 DD.MM.YYYY

```
$ echo 12/30/1969' |
  sed 's!\([0-9][0-9]\)/\([0-9][0-9]\)/\([0-9]\{2,4\}\)!
  \2.\1.\3!g'
```

awk

awk *'instruction' files*
awk –f *script files*

包含在 *instruction* 或 *script* 中脚本必须由一系列的/pattern/ {action}对组成。*action* 码应用于被 *pattern* 的每一行。awk 还为模式匹配提供了一些函数。

函数

match(*text*, *pattern*)

如果 *pattern* 匹配 *text*,返回 *text* 中匹配的起始位置。一个成功的匹配还设置变量 RSTART 的值为匹配起始位置，设置变量 RLENGTH 的值为匹配中字符的个数。

gsub(*pattern*, *replacement*, *text*)

用 *replacement* 来替换 *text* 中每一个 *pattern* 匹配，并返回替换的次数。如果没有指定 *text* 则默认为$0。

sub(*pattern*, *replacement*, *text*)

用 *replacement* 来替换 *text* 中第一个 *pattern* 匹配，并返回替换的次数。一个成功的替换返回 1，一个不成功的替换返回 0。如果没有指定 *text* 则默认为$0。

实例

创建一个 awk 文件，并通过命令行执行这个文件

```
$ cat sub.awk
{
        gsub(/https?:\/\/[a-z_.\\w\/\\#~:?+=&;%@!-]*/,
        "<a href=\"\&\">\&</a>");
        print
}
$ echo "Check the web site, http://www.oreilly.com/
catalog/repr" | awk -f sub.awk
```

其他资源

- O’Reilly 出版的 Dale Dougherty 和 Arnold Robbins 著作《sed 和 awk》是相关工具的参考手册。