第六章 匹配Unicode和其他字符

有时我们需要匹配ASCII范围之外的字符。ASCII（American Standard Code for Information Interchange，美国信息交换标准代码）定义了英文字符集（A到Z的大写和小写字母，以及控制字符与其他字符）。它的历史已经很久了，早在1968年这个基于拉丁字母的含有128个字符的字符集就得到了标准化。那时个人电脑、VisiCalc、鼠标、Web都还没有出现，而现在我仍然会经常在线查询ASCII字符表。

很多年以前，我刚开始自己的职业生涯的时候，与一位工程师同事在钱包里放了一张ASCII码表。这是以防万一：出门时别忘记带上ASCII码表。

我并不否认ASCII的重要性，但是现在它已经过时了，尤其是我们有了可以表示超过10万个字符的Unicode标准（http://www.unicode.org）。然而，Unicode也没有完全舍弃ASCII，它将ASCII加入了它的基本拉丁（Basic Latin）码表中（参见http://www.unicode.org/charts/PDF/U0000.pdf）。

本章，我们将跳出ASCII的小圈子，投入已经较为普及的Unicode的世界。

本章第一个示例文本是代码库中的voltaire.txt文件，这是法国哲学家伏尔泰（1694-1778）的一段话。

Qu’est-ce que la tolérance? c’est l’apanage de l’humanité. Nous sommes tous pétris de faiblesses et d’erreurs; pardonnons-nous réciproquement nos sottises, c’est la première loi de la nature.

翻译成英文就是：

What is tolerance? It is the consequence of humanity. We are all formed of frailty and error; let us pardon reciprocally each other’s folly—that is the first law of nature.

这段话的意思是：

什么是宽容？它是人性的产物。我们生来都有缺陷和错误，就让我们原谅彼此的蠢行吧！这才是自然的第一法则。

6.1 匹配Unicode字符

有很多方式可以指定Unicode字符（也称为代码点）。为了讲解方便，本书将Unicode字符看做ASCII字符范围以外的字符，但严格说来这并不准确。

先将伏尔泰的名言输入到Regexpal（http://www.regexpal.com）中，然后输入这个正则表达式：

\u00e9

\u之后紧跟十六进制值00e9（这里不区分大小写，即00E9也可以）。00e9对应十进制值233，在ASCII范围（0~127）之外。

注意在Regexpal中字母é（小写e加上一个重音符）被标亮了（参加图6-1）。这是因为在Unicode中é就是代码点U+00E9，所以就被\u00e9匹配。

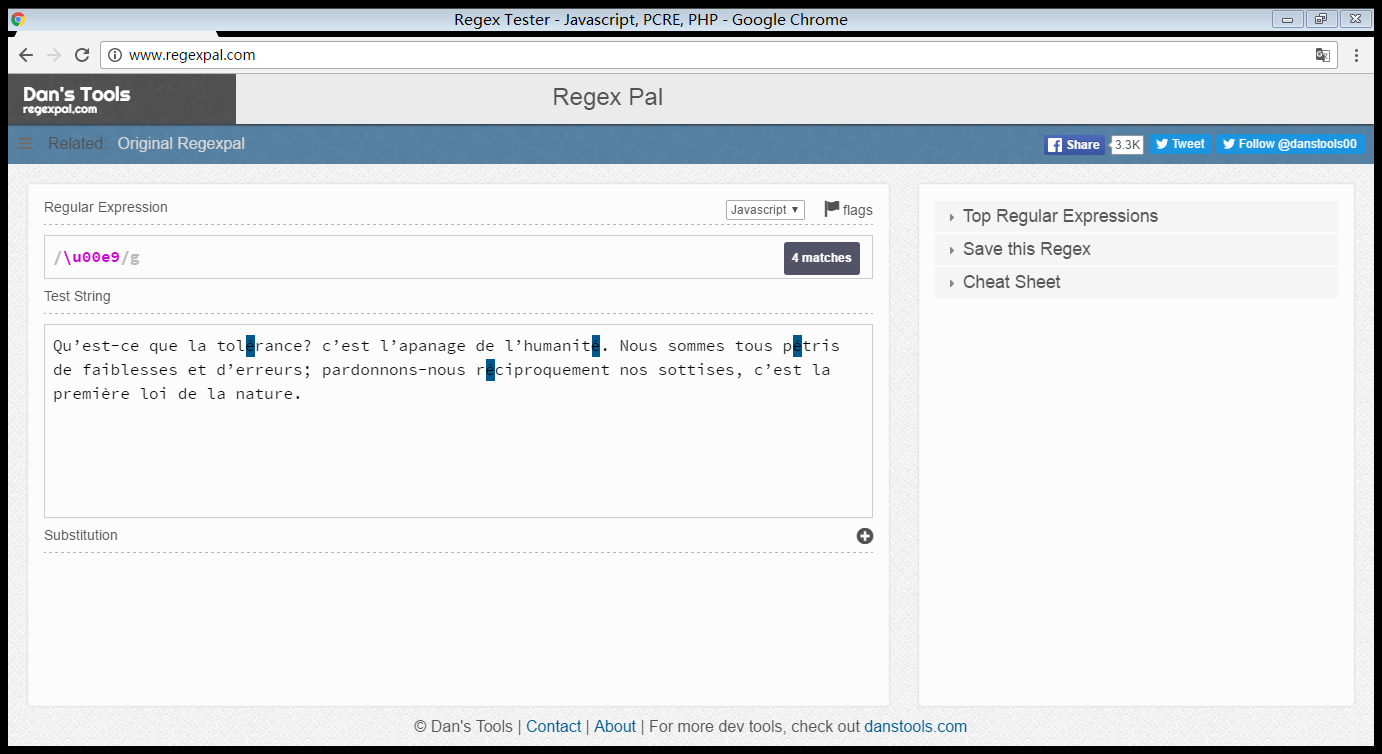


图 6-1：在Regexpal中匹配U+00E9

Regexpal使用的是JavaScript的正则表达式实现。JavaScript也允许你使用下面的语法：

\xe9

请在Regexpal中试一下，看看它是不是跟\u00e9一样也匹配同一个字符。

让我们在另一个正则表达式出来引擎中试一下。请在浏览器中打开http://regexhero.net/tester/。Regex Hero是用.NET编写的且语法稍有不同。将文件basho.txt中的内容放入标签为Target String的文本区域中。其内容包含日本诗人松尾芭蕉（Matsuo Basho，他恰巧是在伏尔泰出生的前一周去世的）的俳句。

以下是该诗的日文版：

古池

蛙飛び込む

水の音

—芭蕉 (1644–1694)

接下来是该诗的英译版：

At the ancient pond

a frog plunges into

the sound of water.

—Basho (1644–1694)

请在标签为Regular Expression的文本区中键入以下内容来匹配日文文本的部分内容：

\u6c60

这是单词pond（池塘）所对应的日文字符的代码点，该字会在下方被标亮（参见图6-2）。

另外，你可以试一下匹配长破折号（—）：

\u2014

或者短破折号（-）:

\u2013

现在再在编辑器中看看这些字符。

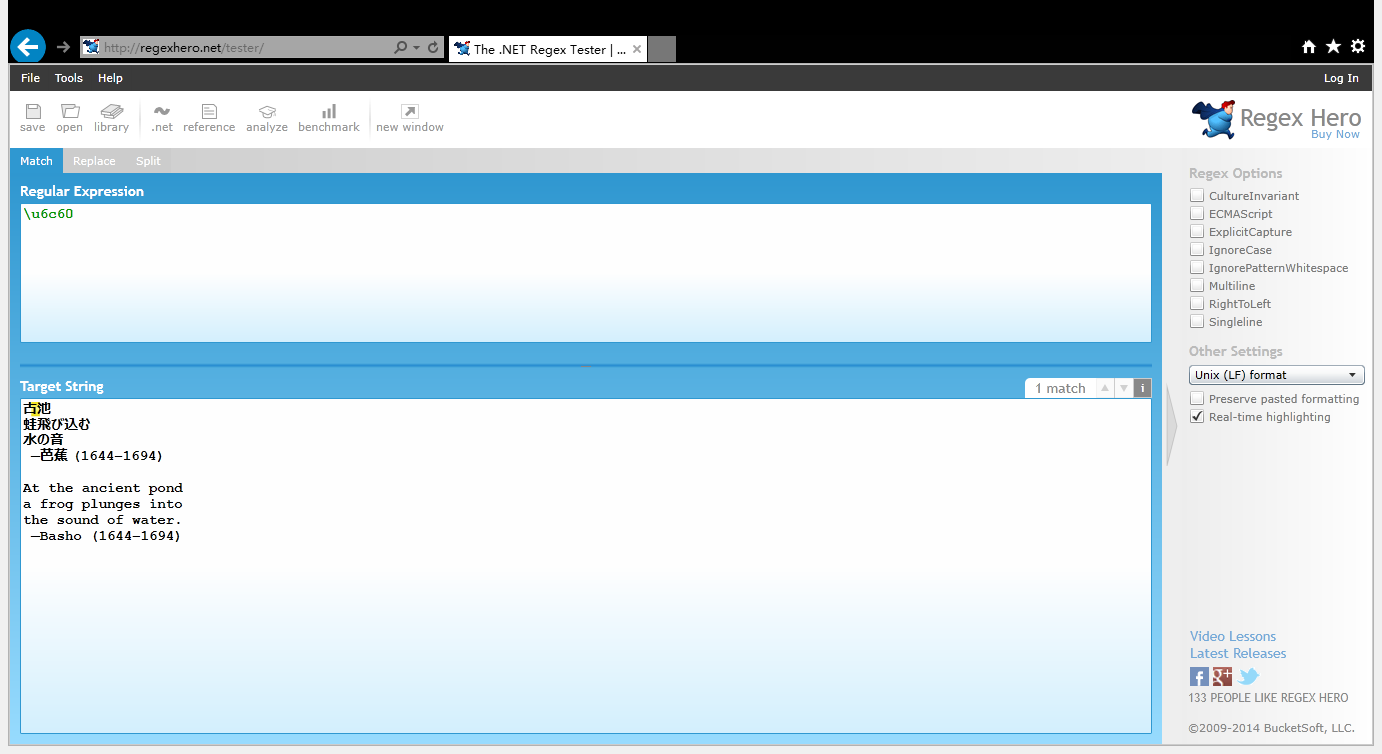


图 6-2：在Regex Hero中匹配U+6c60

使用Vim

如果你的系统里有Vim，可以用它打开basho.txt文件，如下所示：

Vim basho.txt

现在以斜线（/）为起始，输入下面一行来查找：

/\%u6c60

然后键入回车（Enter或Return）。如图6-3所示，光标移到了匹配部分的起始处。表6-1列出了可以设置的选项。在\%之后，你可以使用x或X来匹配0-255（0-FF）范围内的值，使用u匹配256-65535（100-FFFF）范围内的四位十六进制数，还可以用U来匹配65526-2147482647（10000-7FFFFFFF）范围内的八位十六进制数。这样就能涵盖很多编码，其数量远远超过Unicode现有的字符编码数量。

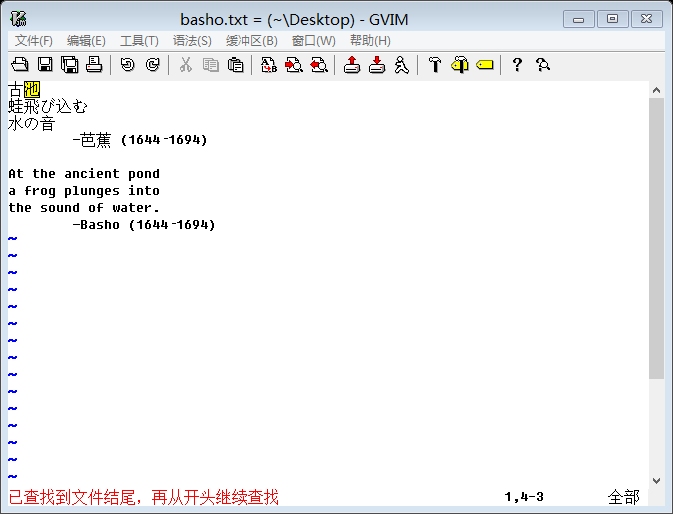


图 6-3：在Vim中匹配U+6c60

表 6-1：在Vim中匹配Unicode编码

|  |  |  |
| --- | --- | --- |
| 首 字 符 | 最大字符数 | 最 大 值 |
| x或X | 2 | 255（FF） |
| u | 4 | 65535（FFFF） |
| U | 8 | 2147483647（7FFFFFFF） |

6.2用八进制数匹配字符

还可以使用八进制数来匹配字符，八进制数以8为基数，使用数字0到7计数。在正则表达式处理器中，就是要在反斜线（\）后加三位数字。

比如，以下八进制数：

\351

等同于：

\u00e9

请在Regexpal中用伏尔泰的文本实验一下。\351匹配é，键入的内容少了一点儿。

6.3 匹配Unicode字符属性

在某些实现（比如像Perl）中，还可以匹配Unicode的字符属性。这些属性包括字符是否是字母、数字或标点符号。

现在介绍一下ack，这是一个用Perl语言编写的命令行工具，它跟grep功能相似（见http：//betterthangrep.com）。系统一般不会附带这个程序，用户需要自己下载并安装。（参见6.6节）。

我们要用ack来处理席勒1785年的作品“An die Freude”的片段（或许你不认识，这是德语）：

An die Freude.

Freude, schöner Götterfunken,

Tochter aus Elisium,

Wir betreten feuertrunken

Himmlische, dein Heiligthum.

Deine Zauber binden wieder,

was der Mode Schwerd getheilt;

Bettler werden Fürstenbrüder,

wo dein sanfter Flügel weilt.

Seid umschlungen, Millionen!

Diesen Kuß der ganzen Welt!

Brüder, überm Sternenzelt

muß ein lieber Vater wohnen.

该片段中有几个有趣的字符，它们不在ASCII的范围内。我们要通过属性来看看这首诗的文本内容。（如果你想知道这段文字的意思，可以将其输入到Google Translate中（http://translate.google.com）.）

在命令行中使用ack时，你可以指定查看那些属性为字母（L）的字符：

ack `\pL` schiller.txt

该命令会将字母都高亮显示。对于小写字母，则要在括号中使用Ll：

ack `\p{Ll}` shiller.txt

括号是必须的。对于大写字母，则是Lu：

ack `\p{Lu}` shiller.txt

要指定不符合某个属性的字符，则使用大写P：

ack `\PL` schiller.txt

这个命令会将非字母字符标亮。

下面的正则表达式用于查找非小写字母：

ack `\P{Ll}` schiller.txt

而这个正则表达式则高亮非大写字母：

ack `\P{Lu}` schiller.txt

在另一个基于浏览器的正则表达式测试程序http://regex.larsolavtorvik.com中也可以这样做。

表6-2列出了在\p{property}或\P{property}中使用的字符属性表名（参见http://www.pcre.org/pcre.txt中的pcresyntax(3)）。还可以用属性对各种人类语言进行匹配。

表 6-2：字符属性

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属 性 | 描 述 | 属 性 | 描 述 | 属 性 | 描 述 |
| C | 其他字符 | M | 标记符号 | Pi | 起始 |
| Cc | 控制字符 | Mc | 空格符号 | Po | 其他标点符 |
| Cf | 格式字符 | Me | 环绕标记 | Ps | 开始标点符 |
| Cn | 未分配字符 | Mn | 非空格标记 | S | 符号 |
| Co | 专用字符 | N | 数字 | Sc | 货币符号 |
| Cs | 替代字符 | Nd | 十进制数字 | Sk | 修饰符号 |
| L | 字母 | Nl | 字母数字 | Sm | 数学符号 |
| Ll | 小写字母 | No | 其他数字 | So | 其他符号 |
| Lm | 修饰字母 | P | 标点符号 | Z | 分隔符 |
| Lo | 其他字母 | Pc | 连接标点符 | Zl | 行分隔符 |
| Lt | 标题大写字母 | Pd | 破折号 | Zp | 段落分隔符 |
| Lu | 大写字母 | Pe | 结束标点符 | Zs | 空格分隔符 |
| L& | Ll、Lu或者Lt | Pf | 最终标点符 |  |  |

6.4 匹配控制字符

如何匹配控制字符呢？虽然你很少会在文本查找控制字符，但知道如何做也不是件坏事。在示例代码库中，你会找到ascii.txt文件（共128行），包含了所有的ASCII字符，一个字符占一行（所以有128行）。当你在该文件中查找时，若找到匹配项就返回一行。这个文件用来试试手挺不错的。

如果你使用grep或者ack在ascii.txt中查找字符串或控制字符，这两个程序可能就将该文件当作二进制文件。如果是这样，在对其执行一个脚本时，若找到匹配程序就会报告“Binary file asci.txt matches”（二进制文件匹配成功）。就这一点需要注意。

在正则表达式中，可以像这样来指定一个控制字符：

\cx

其中x就是你想匹配的控制字符。

例如，要在一个文件中查找空字符，可以使用以下Perl命令：

perl –n –e `print if /\c@/` ascii.txt

如果系统中已经安装了Perl且运行正常，就可以得到以下结果：

0. Null

原因是该行有一个空字符，只是结果中看不到这个字符。

如果你用Vim之外的编辑器打开ascii.txt文件，编辑器就有可能将控制字符从文件中删除，所以最好还是使用Vim编辑器。

还可以用\0来查找空字符。试一下这个命令：

perl –n –e `print if /\0/` ascii.txt

再用以下命令来查找报警字符（BEL）：

perl –n –e `print if /\cG/` ascii.txt

这将返回：

7. Bell

或者还可以使用简写式：

perl –n –e `print if /\a/` ascii.txt

要查找转义字符，则使用：

perl –n –e `print if /\c[` ascii.txt

它的结果是：

27. Escape

或者使用简写式：

perl –n –e `print if /\e/` ascii.txt

如何匹配退格符呢？试一下这个：

perl –n –e `print if /\cH/` ascii.txt

它显示：

8. Backspace

也可以使用分类表达式来查找退格符：

perl –n –e `print if /[\b]` ascii.txt

如果没有括号，\b会被认为是什么呢？是第2章中所学到的单词边界。括号改变了正则表达式处理器对\b的理解方式。在本例中，Perl将其看做一个退格符。

表6-3列出了本章中匹配字符的方法。

|  |  |  |  |
| --- | --- | --- | --- |
| 代 码 | 描 述 | 代 码 | 描 述 |
| \uxxxx | Unicode（四位） | \cx | 控制字符 |
| \xxx | Unicode（两位） | \0 | 空字符 |
| \x{xxxx} | Unicode（四位） | \a | 报警字符 |
| \x{xx} | Unicode（两位） | \e | 转义字符 |
| \ooo | 八进制（基数为8） | [\b] | 退格符 |

该表总结了本章的内容，下一章我们将详细学习量词。

6.5 本章所学

如何使用\uxxxx或者\xxx匹配任意Unicode字符

如何在Vim中使用\%xxx、\%Xxxx、\%uxxxx或者\%Uxxxx匹配任意Unicode字符

如何用八进制格式\ooo匹配0-255范围内字符

如何通过\p{x}来使用Unicode字符属性

如何用\e或\cH匹配控制字符

更多有关如何在命令行中使用Perl的内容（更多的单行Perl命令）

6.6 相关资源

我是使用Vim（http://www.vim.org）将控制字符输入到ascii.txt文件中的。在Vim中，可以在使用Ctrl+V之后输入适当的控制字符序列，比如Ctrl+C是文本结束字符。我还在Ctrl+V后输入字符对应的两位十六进制码。还可以使用二合字母（digraph）来输入控制字符；在Vim中输入:digraph就可看到多个可用编码。要输入二合字母，可在插入（Insert）模式下使用Ctrl+K之后输入两个相连的字母（例如，UN代表空字符）。

RegexHero（http://regexhero.net/tester）是一个基于.NET的正则表达式实现程序，它由Steve Wortham编写，在浏览器中运行。这是个收费程序，但可以免费试用，如果你想购买这个程序，它的价格还算合理（有标准版和专业版两种选择）。

Vim（http://www.vim.org）是Bill Joy于1976年发明的vi编辑器的改进版本。它最初由Bram Moolenaar开发。对于不熟悉它的人来说，它显得过时了，但就像我提到的，它功能非常强大。

工具程序ack（http//betterthangrap.com）是使用Perl语言编写的。它功能上与grep相似且有许多命令行选项，在很多方面都超越了grep。例如，它使用的是Perl正则表达式而不是grep之类的基本正则表达式（不使用-E）。要获取安装指南，请参见http://betterthangrep.com/install/。我使用的是“Install the ack executable.”下面的安装指南。我没有使用curl而只用给出的链接下载了ack，之后就将该脚本复制到Mac和Windows7中运行Cygwin（http://www.cygwin.com）的PC的/usr/bin路径下。

第7章 量词

前面介绍过一些量词，而本章来细致地讲解量词。

在下面的示例中，我们将使用第5章用过的Mac桌面应用程序Reggy。取消勾选底部的Match All（匹配所有）选项。

如果你使用的不是Mac，可以在本书之前提到的某个程序中尝试这些示例。请将triangle.txt中排成直角三角形的数字粘贴到程序中。该文件位于示例代码库中。

7.1 贪心、懒惰和占有

这说的可不是小时候的你，而是在说量词。这些形容词听起来不像什么好特性，但要想把正则表达式用好，却一定要了解量词的这些特性。

量词自身是贪心的。贪心的量词会首先匹配整个字符串。尝试匹配时，它会选定尽可能多的内容，也就是整个输入。量词首次尝试匹配整个字符串，如果失败则回退一个字符后再次尝试。这个过程叫做回溯（backtracking）。它会每次回退一个字符，直到找到匹配的内容或者没有字符可尝试为止。此外，它还记录所有的行为，因此相较另两种方式它对资源的消耗最大。它先“吃”尽所有的字符，然后每次“吐”出一点，慢慢咀嚼消化……你应该明白了。

懒惰（有时也说勉强）的量词则使用另一种策略。它从目标的起始位置开始尝试寻找匹配，每次检查字符串的一个字符，寻找它要匹配的内容。最后，它会尝试匹配整个字符串。要使一个量词成为懒惰的，必须在普通量词后添加一个问号（?）。它每次只“吃”一点。

占有量词会覆盖整个目标然后尝试寻找匹配内容，但它只尝试一次，不会回溯。占有量词就是在普通量词之后添加一个加号（+）。它不“咀嚼”而是直接“吞咽”，然后才想知道“吃”的是什么。下面几节将逐一展示这几种量词。

7.2 用\*、+和?进行匹配

如果你在Reggy中输入了前面的数字，现在就可以开始测试了。首先使用Kleene星号，这一命名是为了纪念正则表达式的发明人Stephen Kleene。如果像下面这样在点号之后使用星号：

.\*

它将会以贪心的方式匹配主文本中的所有字符（数字）。如前所述，.\*匹配任何字符零次或多次。下方文本框中的所有数字都会以另一种颜色高亮显示。一本早期手册对Kleene星号的描述如下：

正则表达式后跟一个“\*”（Kleene星号）表示该正则表达式所匹配的文本接连出现任意次（包括零次）。

现在试一下：

9\*

底部数字9的那一行就会被标亮。现在再试一下：

9.\*

这就会将包含数字9的行和下面的包含数字0的行标亮。因为勾选了Multiline选项（在应用程序窗口底部），点号会匹配行之间的换行符；正常情况下，它不会匹配换行符。

要匹配一个或多个9，尝试：

9+

跟前面有什么不同呢？暂时看不出来，因为在主文本中有九个9。主要区别是+会寻找至少一个9，而\*会寻找零个或多个9。

要匹配零次或一次（可选），可使用：

9?

这只会匹配第一次出现的9。这个数字9是可选的，由于它存在于主文本中，所以它会匹配并被标亮。如果用下面这个表达式：

99?

则第一个9和第二个9都会匹配。

表7-1列出了基本的量词以及它们的一些可能的作用。这些量词默认是贪心的，这意味它们在第一次尝试时会尽可能多地匹配字符。

表 7-1：基本量词

|  |  |
| --- | --- |
| 语 法 | 描 述 |
| ? | 零个或一个（可选） |
| + | 一个或多个 |
| \* | 零个或多个 |

7.3 匹配特定次数

使用花括号可以限制某个模式在某个范围内匹配的次数，未经修饰的量词就是贪心量词。例如：

7{1}

会匹配第一次出现的7。要匹配一个或多个数字7，只要加一个逗号即可：

7{1,}

你可能已经意识到了

7+

和7{1,}

本质上是一样的，而：

7\*

和

7{0,}

也是相同的。另外，

7?

与

7{0,1}

也是一样的。

还可以匹配m到n次，比如：

7{3,5}

会匹配三个、四个以及五个7.

可以看出，花括号（或者说范围语法）是最灵活和精确的量词。表7-2总结了这些特性。

表 7-2：范围语法总结

|  |  |
| --- | --- |
| 语法 | 描述 |
| {n} | 精确匹配n次 |
| {n,} | 匹配n次或更多次 |
| {m,n} | 匹配m至n次 |
| {0,1} | 与?相同（零次或一次） |
| {1,} | 与+相同（一次或更多） |
| {0,} | 与\*相同（零次或更多） |

7.4 懒惰量词

现在让我们把贪心特性放到一边而来看看懒惰量词。理解懒惰特性最好的方式就是看看实际应用。请在Reggy（确定Match All没有被勾选）中尝试用问号（?）匹配零个或者一个5：

5?

第一个5被标亮了。请再加一个?来使量词变成懒惰的：

5??

现在它看起来不匹配任何内容了，其原因是该模式已经是懒惰的了。也就是说，它不会强制匹配第一个5。懒惰的基本特性就是匹配尽可能少的字符——它就是个“懒虫”。

试一下匹配零次或者多次的量词：

5\*?

它也不会匹配任何内容，因为它可以选择匹配最少的次数——零次。

再试下匹配一次或多次：

5+?

看到了吧，懒惰特性使其只匹配了一个5。它只需要做到这个程度就可以了。

使用m和n方式匹配时就更为有趣了。请尝试：

5{2,5}?

只匹配了两个5，而不像贪心量词那样匹配五个。

表 7-3：懒惰量词

|  |  |
| --- | --- |
| 语法 | 描述 |
| ?? | 懒惰匹配零次或一次（可选） |
| +? | 懒惰匹配一次或多次 |
| \*? | 懒惰匹配零次或多次 |
| {n}? | 懒惰匹配n次 |
| {n,}? | 懒惰匹配n次或多次 |
| {m,n}? | 懒惰匹配m至n次 |

7.5 占有量词

占有式匹配很像贪心式匹配，它会选定尽可能多的内容。但与贪心式匹配不同的是它不进行回溯。它不会放弃所找到的内容，它很自私，这也是把它称为占有式（possessive）的原因。它紧紧“抱”住自己所选的内容，一点也不放弃。但占有量词的优点是速度快，因为无需回溯。当然，匹配失败的话也很快。

说实话，本书的例子你很难看出贪心式、懒惰式以及占有式量词的区别。但随着经验的增长，以及对性能的看重，你会发觉其中的不同。

为了理解这一点，我们先尝试匹配以零开头的多个零，然后再匹配以零结尾的多个零。在Reggy中，先确定勾选Match All，然后输入以下以零开头的表达式：

0.\*+

发生什么了？所有的零都被标亮了。存在一个匹配。占有式的匹配看起来和贪心式的匹配是一样的，但没有回溯。可以证明一下。输入这个带有结尾零的表达式：

.\*+0

没有匹配——原因就是没有回溯。它一下就选定了所有的输入，不再回过来查看。它“挥霍”了自己的财产。它一下子没有在结尾找到零，也不知道该从哪里找起。如果将加号去掉，它会找到所有的0，因为它变回贪心式匹配了。

.\*0

当你知道文本中的内容时，就知道在哪里可以找到匹配，这时你应该会使用占有量词。你不在乎它是否会选定所有内容。占有式匹配有助于提高匹配的性能。表7-4列出了占有量词。

表 7-4：占有量词

|  |  |
| --- | --- |
| 语法 | 描述 |
| ?+ | 占有式匹配零次或一次（可选） |
| ++ | 占有式匹配一次或多次 |
| \*+ | 占有式匹配零次或多次 |
| {n}+ | 占有式匹配n次 |
| {n,}+ | 占有式匹配n次或多次 |
| {m,n}+ | 占有式匹配m至n次 |

下一章将介绍环视。

7.6 本章所学

贪心式、懒惰式以及占有式匹配的区别

如何匹配一次或多次（+）

如何进行可选匹配（?，零次或多次）

如何匹配零次或多次（\*）

如何使用{m,n}量词

如何使用贪心、懒惰（勉强）以及占有量词

7.7 相关资源

引文出自Dennis Ritchie和Ken Thompson，QED Text Editor（1970年贝尔实验室出版）p.3（参见http://cm.bell-labs.com/cm/cs/who/dmr/qedman.pdf）。