文章
jieliang liu · 一月 27, 2021 阅读大约需 13 分钟

在Caché中使用正则表达式

1.关于本文

就像Caché模式匹配一样，正则表达式也可以在Caché中用来识别文本数据中的模式--只是表达能力更强。本文简要介绍了正则表达式，以及在Caché中如何使用它。这里提供的信息基于各种来源，最值得拜读的是Jeffrey Friedl的《掌握正则表达式》一书，当然还有Caché在线文档。本文无意讨论正则表达式的所有可能性和细节。如果你想了解更多，请参考第5章中列出的信息来源。

使用模式进行文本处理有时会变得很复杂。在处理正则表达式时，我们通常有几种实体：我们正在搜索模式的文本、模式本身（正则表达式）和匹配（文本中与模式匹配的部分）。为了便于区分这些实体，本文档中使用了以下约定。

文本样本以单色字体单独列出，不加引号。

```
This is a "text string" in which we want to find "something".
```

除非不明确，否则正文中的正则表达式会以灰色背景显示，如本例。\".*?\".

需要时用不同的颜色突出显示匹配。

```
这是一个"text string"，我们要在其中找到"something"。
```

代码样本会显示在如下的文本框里：

```
set t=" This is a ""text string"" in which we want to find ""something "

set r="\"".*?\""""

w $locate(t,r,,tMatch)
```

2.一些历史（和一些小事）。

在
20世纪40年代初，神经生理学家开发了人类神经系统的模型。几年后，一位数学家用一种代数来描述这些模型，他称之为"正则集"。这种代数的符号被命名为"正则表达式"。

1965年，正则表达式第一次在计算机的范畴内被提及。随着qed，一个作为UNIX操作系统一部分的编辑器，正则
表达式开始传播。该编
辑器后来的版本提供了一个命令序列g/正则表
达式/p（*全局、正则表达式、打印*
），在所有文本行中搜索匹配的正则表达式并输出结果。这个命令序列最终成为独立的UNIX命令行程序"grep"。

今天，许多编程语言都存在正则表达式（RegEx）的各种实现（见3.3节）。

3.Regex 101

就像Caché模式匹配一样，正则表达式也可以用来识别文本数据中的模式--只是表达能力更强。下面的章节概述了正则表达式的组成部分，它们的评估和一些可用的引擎，然后在第4章中详细介绍如何使用。

3.1.正则表达式的组成部分

3.1.1.Regex元字符

以下字符在正则表达式中具有特殊意义。

. ( ) [ ] \ ^ $ |

如果你需要将它们作为字面数使用，你需要使用反斜杠来转义。你也可以使用 \Q <literal sequence> \E显式指定字面序列。

3.1.2.文字

普通文本和转义字符被视为字面，例如：。

- abc — abc
- \f — 换页
- \n — 换行
- \r — 回车
- \v — 标签
- \0+三位数(如：0101) — 八进制数
  Caché (ICU)中使用的regex引擎支持八进制数，最高可达\0377（十进制系统为255）。当你从另一个引擎迁移正则表达式时，
  请确保你了解它如何处理八进制数。
- \x+两位数(如：x41) — 十六进制数
  Caché库确实提供了更多处理十六进制数的选项，请参考文档（链接可以在5.8节找到）。

3.1.3.锚

使用锚点，你可以匹配文本/字符串中的位置，例如：。

- \A 字符串的开始
- \Z 字符串的末端
- ^ 文本或行的开始
- $ 文末或行末
- \b 字词边界
- \B 不字界
- < 词的开头
- #> 词尾

和一些RegEx引擎的行为有所不同，例如，对构成单词的确切定义以及哪些字符被视为单词定界符。

3.1.4.量词

使用正则表达式量词，你可以指定前面的元素可能出现的频率来进行匹配。

- {x}正好出现x次
- {x,y}最小x，最大y的出现次数。
- * 0或更多；相当于{0,}。
- +1或更多；相当于{1,}。
- ? 0或1

量词很"*贪婪*"，它们会尽可能多地抓取字符。假设我们有下面的文本字符串，想找到带引号的文本。

This is "a text" with "four quotes".

由于选择器的贪婪性质，正则表达式"/".*/"会找到太多的文本。

This is "a text" with "four quotes".

在这个例子中，正则表达式.*试图捕捉尽可能多的位于一对引号之间的字符。然而，由于点选择器 (.) 也匹配引号，我们没有得到我们想要的结果。

通
过一些regex引擎（包括Caché使用的引擎），你可以通过添加一个问号来控制量化符的贪婪程度。因此，正则表达式"\".*?\"现在可以匹配引号中的两部分文本--这正是我们要找的。

This is "a text" with "four quotes".

3.1.5.字符类(范围)

方括号用于指定字符的范围或字符集，例如 [a-zA-Z0-9] 或 [abcd] - 在 regex
中，这被
称为字符类。一个
范围可以匹配单个字符，所以范围定义中的字符顺序无关紧要--[dbac]返回的匹配结果与[abcd]相同。

要排除一个字符范围，只需在字符范围定义前面加上^（在方括号内！）。[^abc] 匹配除了a,
b或c以外的任何字符.

一些regex引擎确实提供了预先定义的字符类(POSIX)，例如。

- [:alnum:] [a-zA-z0-9]
- [:alpha:] [a-zA-Z]
- [:blank:] [\t]
- …

3.1.6.Groups (组)

正
则表达式的部分内容可以使用对括号进行分组。这对于将量化符应用于一组选择符，以及从同一regex内（反向引用）和从调用正则表达式的Caché对象脚本代码（捕获缓冲区）中引用分组都很有用。组可以被嵌套。

下
面的regex匹配由一个三位数组成的字符串，后面是一个破折号，然后是3对大写字母和一个数字，后面是一个破折号，然后是与第一部分相同的三位数。

([0-9]{3})-([A-Z][[0-9]){3}-\1

这个例子展示了如何使用*反向引用*
（见下文）不仅匹配结构，而且匹配内容：反向引用（紫色）告诉引擎在结尾处寻找与开头处相同的三位数数字（黄色）。它还演示了如何将量词应用于更复杂的结构（绿色）。

上面的regex将匹配以下字符串。

123-D1E2F3-123

在这些上面是不匹配的。

123-D1E2F3-456(最后三位数与前三位数不同)

123-1DE2F3-123(中间部分不包含三个字母/数字对)

123-D1E2-123(中间部分只包含两个字母/数字对)

组
也会填充所谓的捕获缓冲区（见4.5.1节）。这是一个非常强大的功能，它允许同时匹配和提取信息。

3.1.7. Alternations(交替)

使用管道字符来指定alternations，例如skyfall|done
。这允许匹配更复杂的表达式，如3.1.5节中描述的字符类。

3.1.8.回溯引用

后
面的引用允许您引用以前定义的组（括号中的选择器）。下面的例子显示了一个正则表达式，它匹配三个必须相等的连续字符。

([a-zA-Z])/1/1

后面的引用由\x指定，而x代表第x个括号中的表达式。

3.1.9.优先规则

1. []在()之前
2. +和? 在序列前：ab等于a(b*)，而不是(ab)*。
3. 序列在alternation前：ab|c等于(ab)|c，而不是a(b|c)

3.2.一些理论

正
则表达式的评估通常采用以下两种方法之一来实
现（这里描述是简化的，请参考第5章中提到的文献进行深入讨论）。

1. 文本驱动(DFA - *Deterministic Finite Automaton*) 引擎逐字逐句地检查输入文本，并尝试匹配它目前所拥有的内容。当它真正到达输入文本的结尾时，它宣布成功。
2. Regex-driven (NFA - *Non-deterministic Finite Automaton*) 引擎会逐一检查正则表达式，并尝试将其应用到文本中。当它真正到达（并匹配）最后一个标记时，它宣布成功。

方
法1是确定性的，执行时间只取决于输入文本的长度。正则表达式中选择符的顺序不影响执行时间。

方
法2是非决定性的，引擎会遍历正则表达式中选择符的所有组合，直到找到匹配或遇到错误。因此，当它*没有*
找到匹配项时，

这种方法特别慢（因为它必须遍
历所有可能的组合）。选择符的顺序*确实*
对执行时间有影响。但是，这种方法允许回溯和捕获缓冲区。

3.3.Regex引擎

目
前有很多不同的regex引擎，有些是编程语言或操作系统的内置部分，有些是几乎可以在任何地方使用的库。以下是一些regex引擎，按评估方法分组。

- DFA: grep, awk, lex.
- NFA：Perl、Tcl、Python、Emacs、sed、vi、ICU。

下表是各种编程语言和库中可用的regex功能的比较。

	"+" quantifier	Negated character classes	Non-greedy quantifiers	Shy groups	Recursion	Look-ahead	Look-behind	Backreferences	>9 indexable captures	Directives	Conditionals	Atomic groups	Named capture	Comments	Embedded code	Unicode property support	Balancing groups	Variable-length look-behinds
.NET	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Some	Yes	Yes
Boost.Regex	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Some	No	No
Boost.Xpressive	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	No	No
CL-PPCRE	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Some	No	No
GLib/GRegex	Yes	?	Yes	?	No	?	?	?	?	Yes	Yes	Yes	Yes	Yes	No	Some	No	No
GNU grep	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	?	Yes	Yes	?	Yes	Yes	No	No	No	No
Haskell	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	?	?	?	?	?	No	No	No	No
ICU Regex	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	Yes	No	No
Java	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Some	No	No
JavaScript	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No	No	No	No	No	No	No	No	No
JGsoft	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Some	No	Yes
Lua	Yes	Yes	Yes	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No	No
OCaml	Yes	Yes	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No	No
PCRE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Perl	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
PHP	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No
Python	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No
Qt/QRegExp	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No	No	No	No	No	No	No	No	No
R	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	?	?	?	?	?	?	?	?	?
RE2	Yes	Yes	Yes	Yes	No	No	No	No	Yes	Yes	No	?	Yes	No	No	Some	No	No
RGX	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No
Ruby	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Some	No	No
Tcl	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	Yes	No	No
TRE	Yes	Yes	Yes	Yes	No	No	No	Yes	No	Yes	No	No	No	Yes	No	?	No	No
Vim	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No	No	Yes
XRegExp	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Some	No	No	Yes	Yes	No	Yes	No	No

详情请点击这里：https://en.wikipedia.org/wiki/Comparisonofregularexpressionengines

4.RegEx和Caché

InterSystems Caché使用ICU库来处理正则表达式，Caché在线文档描述了它的许多功能。请参考ICU库的在线文档以了解完整的细节（包括诸如回溯引用等）--ICU的链接可以在5.8节中找到。以下章节旨在为您快速介绍如何使用它。

4.4.$match()和$locate()

在Caché ObjectScript
(COS)中，两个函数$match()和$locate()
提供了对ICU库提供的大部分regex功能的直接访问。$match(String, Regex)
在输入的字符串中搜索指定的Regex模式。当它找到一个匹配的模式时，它返回1，否则它返回0。

例如：

- w $match("baaacd",".*(a)\1/1.*") 返回1。
- w $match("baacd",".*(a)\1/1.*") 返回0。

$locate(String,Regex,Start,End,Value) 就像$match()一样，
以
指定
的regex模
式搜索输入字符串。
然而，$locate()给你更多的控制权，它返
回更多的信息。在Start中，你可以告诉$locate
应该在哪个位置开始搜索输入字符串中的模式。当$locate()
找到一个匹配时，它会返回匹配的第一个
字符的位置，并将End设置为匹配后的下一个字符位置。匹配的内容会在Value中返回。

如果$locate()
没有找到匹配的对象，它将返回0，并且不触及End和Value
（如果指定）的内容。End和Value
是以引用的形式传递的，所以如果你重复使用它（例如在循环中）要小心。

例如：

- w $locate("abcdexyz",".d.",1,e,x) 返回3，e设为6，x设为"cde"

$locate()
执行模式匹配，并且可以同时返回第一个匹配的内容，如果需要提取所有匹配的内容，可以在循环中反复调用$locate()，也可以使用%Regex.Matcher提供的方法。
如果需要提取所有匹配的内容，你可以
在一个循环中重复调用$locate()，或者你可以使用%Regex.Matcher提供的方法（见下一节）。

4.5.%Regex.Matcher.

%Regex.Matcher提供了ICU库的regex功能，就像$match()和$locate()一样。然而，
%Regex.Matcher
还提供了一些高级功能，使更复杂的任务变得非常容易使用。下面的章节将重新审视捕获缓冲区，看看用正则表达式替换字符串的可能性以及控制运行时行为的方法。

4.5.1.捕获缓冲区(Buffers)

正如我们在关于组、回溯引用和$locate()的
章节中已经看到的，正则表达式允许你同时搜索文本中的模式并返回匹配的内容。它的工作原理是将您想要提取的模式的部分放在一对括号中（组）。匹配成功后，捕获缓冲区包含所有匹配的组的内容。请注意，这与$locate()通过其值参数提供的内容略有不同：$locate()返回整个匹配本身的内容，而捕获缓冲区则让您访问匹配的*部分内容*（组）。

要使用它，你需要创建一个 %Regex.Matcher
类的对象，并将正则表达式和输入字符串传递给它。然后你可以调用%Regex.Matcher提供的方法来执行实际工作。

例1（简单组）：

```
set m=##class(%Regex.Matcher).%New("(a|b).*(de)", "abcdeabcde")
```

w m.Locate()返回1

w m.Group(1) 返回 a

w m.Group(2) 返回 de

例2(嵌套组和回溯引用)。

set m=##class(%Regex.Matcher).%New("((a|b).*?(de))(\1)", "abcdeabcde")

w m.Match()返回1

w m.GroupCount返回4

w m.Group(1) 返回 abcde。

w m.Group(2) 返回 a

w m.Group(3) 返回 de

w m.Group(4) 返回 abcde。

(注意嵌套组的顺序--因为开头的括号标志着一个组的开始，所以内部组的索引号比外部组的索引号高)

如
前所述，捕获缓冲区是一个非常强大的功能，因为它们允许您同时匹配模式和提取匹配的内容。如果没有正则表达式，您必须在第一步中找到匹配的内容（例如使用模式匹配操作符），并在第二步中根据一些标准提取匹配的内容（或部分内容）。

如
果您需要对模式中的部分进行分组（例如对该部分应用量化符），但又不想用匹配部分的内容来填充捕获缓冲区，您可以通过在组前加上问号和冒号的方式将组定义为"非捕获"或"害羞"，如下面的例子3。

例3。

set m=##class(%Regex.Matcher).%New("((a|b).*?(?:de))(\1)","abcdeabcde")

w m.Match()返回1

w m.Group(1) 返回 abcde。

w m.Group(2) 返回 a

w m.Group(3) 返回 abcde。

w m.Group(4) 返回 <REGULAR EXPRESSION>zGroupGet+3^%Regex.Matcher.1。

4.5.2.替换

%Regex.Matcher也提供了立即替换匹配内容的方法。 ReplaceAll()和ReplaceFirst()。

set m=##class(%Regex.Matcher).%New(".c.","abcdeabcde")

w m.ReplaceAll("xxxx")　　返回　axxxxeaxxxxe。

w m.ReplaceFirst("xxxx")　　返回　　axxxxeabcde。

你
也可以在替换字符串中引用组。如果我们在上一个例子的模式中添加一个组，我们可以通过在替换字符串中包含$1来引用它的内容。

```
set m=##class(%Regex.Matcher).%New(".(c).","abcdeabcde")."，"abcdeabcde")
```

```
w m.ReplaceFirst("xx$1xx")
```

返回　　axxcxxeabcde。

使用$0在替换字符串中包含匹配的全部内容。

```
w m.ReplaceFirst("xx$0xx")
```

返回　　axxbcdxxeabcde。

4.5.3.操作限制(OperationLimit)

在
3.2节中，我们了解了评估正则表达式的两种方法（DFA和NFA）。Caché中使用的正则表达式引擎是一个非确定性的有限自动机（NFA）。因此，对给定输入字符串评估各种正则表达式的持续时间可能会有所不同。[1]

您可以使用%Regex.Matcher
对象的OperationLimit属性来限制执行单元
的数量（所谓的*簇*
）。执行一个簇的确切持续时间取决于你的环境。通常情况下，一个簇的执行持续时间是非常少的几毫秒。默认情况下，OperationLimit被设置为0（无限制）。

4.6.真实世界的例子：从Perl到Caché的迁移。

本节介绍了从Perl迁移到Caché的过程中与正则表达式有关的部分。Perl
脚本实际上由几十个或多或少复杂的正则表达式组成，这些正则表达式被用来匹配和提取内容。

如
果Caché中没有regex功能，迁移项目就会变成一项重大工作。然而，Caché中的regex功能是可用的，Perl脚本中的正则表达式几乎可以在Caché中使用，而不需要任何改变。

下面是Perl脚本的一部分。

```
sub readJournal{
    my $vcs_file = shift;

    my $vcs = file($vcs_file)->absolute;
    my $domain = undef;
    my $domain_prefix = undef;
    if ($vcs =~ /[\\\/]([^\\^\/]+)[\\\/]ProjectDB[\\\/](.+)[\\\/]archives[\\\/]/i){
                $domain_prefix = uc $1;
                $domain = uc $2;
        }
```

将正则表达式从 Perl 移到 Caché 的唯一改动是 /i 修饰符（使 regex 不区分大小写）--这必须从 regex 的结尾移到开头。

在
Perl中，捕获缓冲区的内容
被复制到特殊的变量中（在上面的Perl代码中是$1和$2
）。在Perl项目中，几乎所有的正则表达式都使用了这种机制。为了类似于这种机制，我们在Caché
对象脚本中写了一个简单的包装方法。它使用 %Regex.Matcher
对文本字符串评估正则表达式，并将捕获缓冲区的内容以列表的形式返回（$lb()）。

由此产生的Caché对象脚本代码如下。

```
如果...RegexMatch(
    tVCSFullName。
    "(?i)[\\/]([^\\^/]+)[\\/]ProjectDB[\\/](.+)[\\/]archives[\\/]",
    .tCaptureBufferList)
    {
        set tDomainPrefix=$zcvt($lg(tCaptureBufferList,1), "U")
        set tDomain=$zcvt($lg(tCaptureBufferList,2), "U")
    }
…
Classmethod RegexMatch(pString as %String, pRegex as %String, Output pCaptureBuffer="") {

    #Dim tRetVal as %Boolean=0。
    set m=##class(%Regex.Matcher).%New(pRegex,pString)
set m.Locate() {
        set tRetVal=1
for i=1:1:m.GroupCount {
            set pCaptureBuffer=pCaptureBuffer_$lb(m.Group(i))
        }
    }
    quit tRetVal
}
```

5.参考资料

5.7.一般资料

概括信息和教程。

- http://www.regular-expressions.info/engine.html

教程和实例。

- http://www.sitepoint.com/demystifying-regex-with-practical-examples/

几种regex引擎的比较。

- https://en.wikipedia.org/wiki/Comparisonofregularexpressionengines

常用表达式cheat sheet。

- https://www.cheatography.com/davechild/cheat-sheets/regular-expressions/pdf/

书。

- Jeffrey E. F. Friedl："掌握正则表达式"(见http://regex.info/book.html)

5.8.Caché在线文件

- 关于Caché中正则表达式的用法概述。 http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GCOSregexp。
- $match()的文档。 http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=RCOSfmatch
- $locate()的文档。 http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=RCOSflocate
- %Regex.Matcher的类引用。 http://docs.intersystems.com/latest/csp/documatic/%25CSP.Documatic.cls?APP=1&LIBRARY=%25SYS&CLASSNAME=%25Regex.Matcher

5.9.ICU

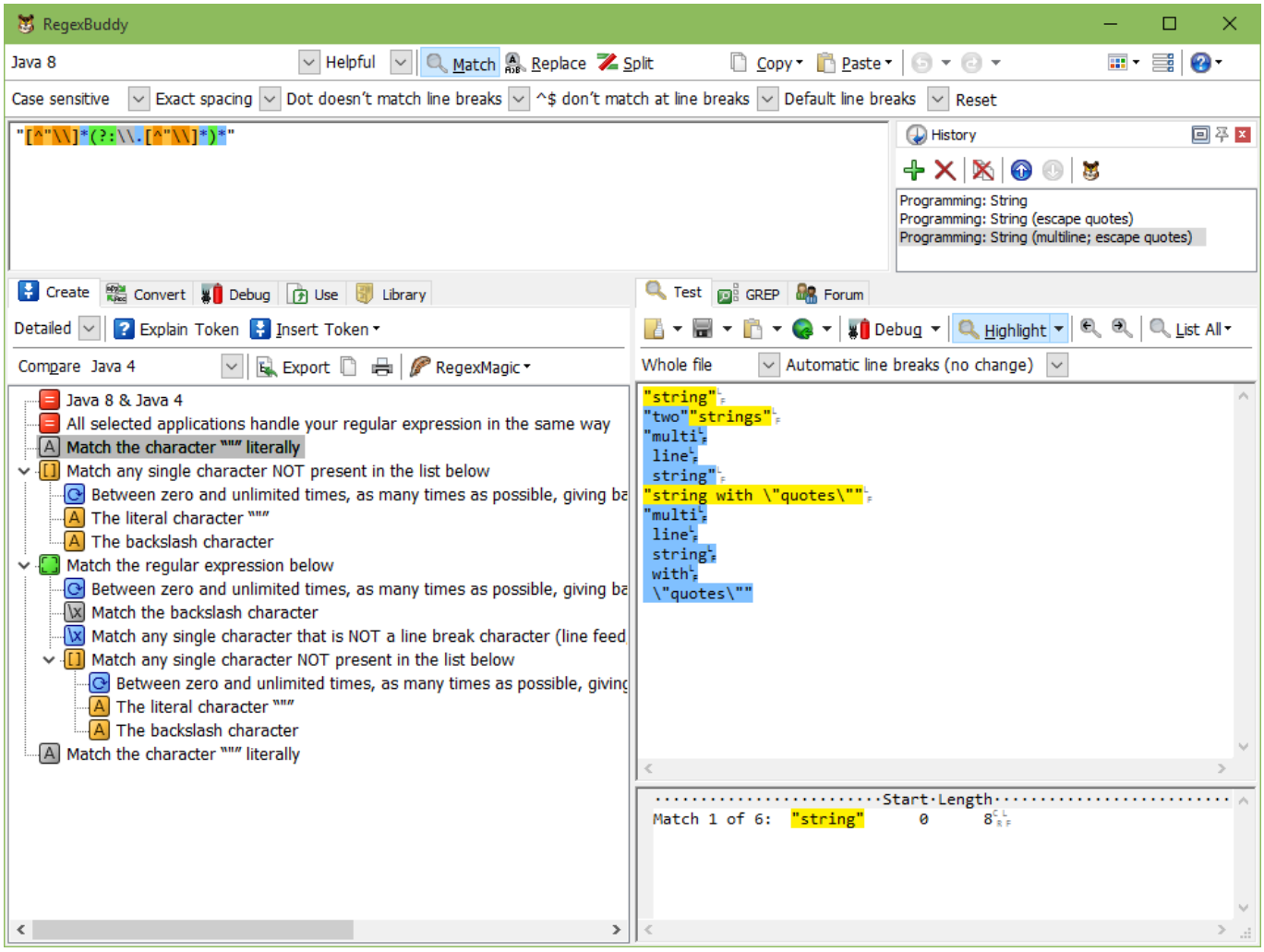
如上所述，InterSystems Caché使用ICU引擎。全面的文档可在网上查阅。

- http://userguide.icu-project.org/strings/regexp
- http://userguide.icu-project.org/strings/regexp#TOC-Regular-Expression-Metacharacters
- http://userguide.icu-project.org/strings/regexp#TOC-Regular-Expression-Operators
- http://userguide.icu-project.org/strings/regexp#TOC-Replacement-Text
- http://userguide.icu-project.org/strings/regexp#TOC-Flag-Options

5.10.工具

有

许多工具支持开发人员创建正则表达式--其中一些是免费的，另一些则有商业许可。我个人的选择是RegexBuddy（http://www.regexbuddy.com/
）--它提供了一套全面的交互式和可视化功能，可以创建和测试不同风味的正则表达式。



#ObjectScript #教程 #Caché #InterSystems IRIS

源URL:
https://cn.community.intersystems.com/post/%E5%9C%A8cach%C3%A9%E4%B8%AD%E4%BD%BF%E7%94%A8%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F