Regular Expressions

Gregory Adam

Contents

- Introduction
- A small program
- The engine
- The Object model
- Operator precedence
- Pattern
- Links
- Example
- .Net bits

Introduction

- A means to match strings of text
- Eg: **fox** in “The quick brown fox jumps over the lazy dog”
- Written in a formal language
  - Mathematical model
  - Generates a parser (interpreted or compiled)
- Many dialects

A small program

```
local obj, s
obj = createobject('VBScript.RegExp')
obj.Global = true        && default is false
obj.IgnoreCase = false   && default = false
obj.Multiline = false    && default false
obj.Pattern = 'fox'

s = 'The quick brown fox jumps over the lazy dog'

&& Test
?'Test: ', obj.Test(s)   && true
?

&& Replace
?'Replace: ', obj.Replace(s, 'fax')
&& The quick brown fax jumps over the lazy dog
?

&& Execute
local matches, match, i
matches = obj.Execute(s)

? 'occurrences: ', matches.Count     && 1
for i = 0 to matches.Count - 1 && zero based
    ? 'occurrence :', i
    match = matches.item[i]

    ? 'Index: ', match.FirstIndex        && 16 - zero based
    ? 'Length: ', match.Length           && 3
    ? 'Value:',match.Value               && fox
endfor
```

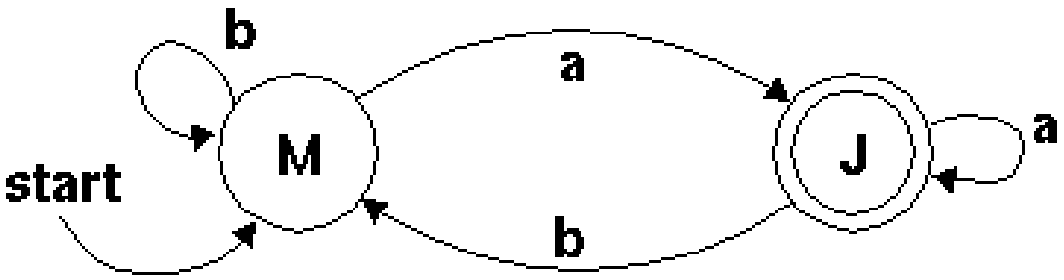
```
Test:  .T.

Replace:  The quick brown fax jumps over the lazy dog

occurrences:            1
occurrence :            0
Index:          16
Length:           3
Value: fox
```

The Engine

- Is a state machine
- Source: http://perl.plover.com/Regex/ http://perl.plover.com/Regex/article.html
- Sample: ^(a|b)*a$



The object model – Properties

Name	Type	Default	
Pattern	string		Defines the regular expression
IgnoreCase	bool	false	True = ignore upper and lower case
Global	bool	false	Find all matches or only the first one

MultiLine	bool	false	False:
			^ matches beginning of the string
			$ matches the end of the string
			
			True:
			^ matches beginning of the string and each position following \n or \r
			$ matches the end of the string and each position before \n or \r

The object model - Methods

Name	Type	Arguments	
Test	bool	(stringToTest)	Tests whether a string matches the pattern Eg: if(regExObj.Test(“abcd”))
Replace	string	(stringToTest, replaceText)	Replaces every successful match with replaceText
Execute	Matches collection	(stringToTest)	Returns a collection of all the matches

The Matches Collection

Name	Type	
Count	Int	The number of matches
Item[i]	Match object	One match item - zero based - see table below

Name	Type	
FirstIndex	Int	Zero based offset of the match
Length	Int	Length of the match
Value	String	Matched value (text)
SubMatches	Collection	One submatch per parenthesis Numbered left to right

Name	Type	* Submatches need version 5.5 of VBScript (IE 5.5)
Count	Int	The number of subMatches Matches.Item[0].SubMatches[0 to Count-1]

Example

- Pattern : ((a+)(b+)) Parentheses: $(_0(_1a+)(_2b+))$
- String: aab abb
- Matches.Item[0].Value : aab
  - SubMatches[0] : aab
  - SubMatches[1] : aa
  - SubMatches[2] : b
- Matches.Item[1].Value : abb
  - SubMatches[0] : abb
  - SubMatches[1] : a
  - SubMatches[2] : bb

Operator precedence

Name		Associativity	
Parentheses	( )		
Quantifiers	* ? + {}	None	
Concatenation	ab	Left	A followed by b
Alternation	|	Left	ab|cd = ab or cd

Associativity is the order of evaluation when the operators are of equal precedence.
- *addition/substraction is left associative*: 1 - 2 + 3 is evaluated as (1-2)+3
- *assignment operators are right associative in C# and C* :

```
a = b = c = 2;  //  a = (b = (c = 2)));
```

Law	
r|s = s|r	| is commutative
r|(s|t) = (r|s)|t	| is associative
r(st) = (rs)t	Concatenation is associative
r(s|t) = rs|rt (r|s)t = rt|rs	Concatenation distributes over |

Pattern – Character escapes

\t	Matches a tab
\r	Matches a carriage return
\n	Matches a line feed
\v	Matches a vertical tab
\f	Matches a form feed
\octal	Use octal representation. Max 3 octal digits, eg \o123
\xhex	Use hex, eg \xc0
\cchar	Matches a control char: eg \cC matches CTRL+C or 0x03
\unnnn	Matches a char represented by 4 hex chars
****num	Matches *num*, where *num* is a positive integer. A reference back to remembered matches. For example, "(.)\1" matches two consecutive identical characters
\	The char following is not a special char Eg: a* matches a*

Pattern – Character classes (1)

		pattern	
[char_group]	Matches any single char in char_group Is by default case sensitive	[ae]	“a” and “e” in “lane”
[^char_group]	Matches any char that is NOT in char_group Only right after the opening [	[^ae] [\^a] or [a^] [[] [^[] []] [^]]	“l” and “n” in “lane” “^” and “a” in “a^” “[“ in “[a” “a” in “[a” “]” in “[a]” “[“ and “a” in “[a]”
[first-last]	Character range: matches any char in the range from first to last Note: [a-z] does not include accented chars	[A-C]	“A”, “B”, “C” in “ABCD”
.	Matches any char except \n	f.x	“fox” in “Atoutfox”

Pattern – Character classes (2)

		pattern	
\w	Matches any word character including underscore. Equivalent to [A-Za-z0-9_]	\w	“a”, “1”, “4” in “a1+4”
\W	Matches any non-word character. Equivalent to [^A-Za-z0-9_]	\W	“+” in “a1+4”
\s	Matches any white space including space, tab, form-feed, etc. Equivalent to [\f\n\r\t\v]	\w\s	"D " in "ID A1.3"
\S	Matches any nonwhite space character. Equivalent to [^ \f\n\r\t\v]	\s\S	“ x” in “int x”
\d	Matches a digit character. Equivalent to [0-9].	\d	"4" in "4 = IV"
\D	Matches a non-digit character. Equivalent to [^0-9].	\D	" ", "=", " ", "I", "V" in "4 = IV"

Anchors (zero-width assertions)

- Cause a match to succeed or fail depending on the current position in the string
- The engine does not advance in the string
- No character is consumed

		pattern	
^	The match must start at the beginning of the string or line.	^\d{3}	"901-" in "901-333-"
$	The match must occur at the end of the string or before \n at the end of the line or string	-\d{3}$	“"-333" in "-901-333"
\b	The match must occur on a boundary between a \w (alphanumeric) and a \W (nonalphanumeric) character.	fox\b	“fox” Atout fox
\B	The match must not occur on a \b boundary.	\Bfox	“fox” in Atout fox

Backreference construct

- Matches the value of a numbered subexpression

		pattern	
(subexpression)	Captures the matched subexpression and assigns it a zero-based ordinal number	(ba)\1	“baba" in “alibaba“
	Note: there must be an exact match of the previously matched subexpression.	(b\w)\1	“baba" in “alibaba“ Nothing in “alibaby”
		(b\w){2}	“baba" in “alibaba“ “baby” in “alibaby”
		(b\w)\1\1 (b\w)\1{2}	“bababa” in “alibababa” “bababa” in “alibababa”

Quantifiers

- specifies how many instances of the previous element must be present in the input string for a match to occur.
- Previous element :
  - can be a character
  - a group
  - a character class

Quantifiers

		pattern	
*	Matches the previous element zero or more times	\d*	“” in “abc” (4 times) “123” in “abc123”
+	Matches the previous element one or more times	\d+	No match in “abc” “123” in “abc123”
?	Matches the previous element zero or one times Makes the previous element optional: same as {0,1}	ab?c ab{0,1}c	“ac” in“ac” “abc” in “abc”
{n}	Matches the previous element exactly n times.	(ba){2}	“baba” in “alibababa”
{n,}	Matches the previous element at least n times	(ba){2,}	“bababa” in “alibababa”
{n,m}	Matches the previous element at least n times, but no more than m times.	(ba){1,2}	“baba” in “alibababa”

Greed

- Always returns the longest possible match

Pattern - Global	Input	Output
<.+>	<a><b><c>	<a><b><c>
<[^>]+>	<a><b><c>	<a> <b> <c>

Substitutions (1)

– Regular expression arguments supported in replacement patterns

		pattern	Replacement pattern	Input	Output
$number	Substitutes the substring matched by group number.	(\w+)(\s*)(\w+)	$3$2$1	"one two"	"two one"
$$	Substitutes a literal "$"	(\w+)(\s*)(\w+)	$3$$$1	"one two"	"two$one"
$&	Substitutes a copy of the whole match.	(\w+)(\s*)(\w+)	$& + $&	"one two"	"one two + one two"

Substitutions (2)

		pattern	Replacement pattern	Input	Output
$`	Substitutes all the text of the input string before the match	(\d+)	$`	"one 2 three"	“one one three"
$'	Substitutes all the text of the input string after the match	(\d+)	$’	"one 2 three"	"“one three three"
$+	Substitutes the last group that was captured	(\w+)(\s*)(\w+)	$+	"abc def 123"	“def 123"
$_	Substitutes the entire input string	(\w+)(\s*)(\w+)	$_	"abc def 123"	"abc def 123 123"

.Net – bits (1)

- Regex class has static methods for
  - Match() (vbscript: Execute(), Global = false)
  - Matches() (vbscript: Execute(), Global = true)
  - IsMatch() (vbscript: Test())
  - Replace()
  - Caches 15 recently used patterns (compiled)
    - Change CacheSize to modify the number

.Net – bits (2)

- Groups can be accessed by (SubMatches in vbscript)
  - Non-negative integer (0-match.Groups.Count-1)
    - Match.Groups[i].Value
    - Groups[0] is the entire matched expression
  - A name in case of named groups
    - (?<FileStem>\w+)
    - Match.Groups("FileStem").Value
  - Named groups can be nested
    - (?<FileName>(?<FileStem>\w+)\.(?<FileExtension>\w+))

.Net – bits (3)

- Positive/negative lookahead/lookbehind zero-width assertions

(?= subexpression)	Zero-width positive lookahead assertion.
(?! subexpression)	Zero-width negative lookahead assertion.
(?<= subexpression)	Zero-width positive lookbehind assertion.
(?<! subexpression)	Zero-width negative lookbehind assertion.

.Net – bits (4)

- Positive/negative lookahead/lookbehind zero-width assertions
- Example : Password constains
  - At least one digit
  - At least one upper-case char
  - At least two lower-case char
  - + sign is not allowed in the password
  - At least 6 chars long

.Net – bits (5)

```
static void Main()
{
    string pattern = @"^(?=.*\d)(?=.*\p{Lu})(?=.*\p{Ll}.*\p{Ll})(?!.*\+).{6,}$";
    // (?=.*\d)             zero-width positive lookeahead.  assure at least one digit
    // (?=.*\p{Lu})         zero-width positive lookeahead.  assure at least one upper-case char
    // (?=.*\p{Ll}.*\p{Ll}) zero-width positive lookeahead.  assure at least two lower-case char
    // (?!.*\+)             zero-with negative look-ahead.  assure no + sign in password
    // .{6,}                at least 6 chars long

    string[] test = {
                     "",                 // false
                     "1",                // false
                     "aa",               // false
                     "abA1+1",           // false
                     "abA1=1",           // true
                     "JeanFrancois1",    // true
                     "Jean+Francois1",   // false
                     "René37"            // true     \p{Ll} matches accented char
                     };

    foreach (var s in test)
    {
        var m = Regex.IsMatch(s, pattern);
        Console.WriteLine("{0}: {1}", m, s);
    }
    Console.ReadLine();

}
```

.Net – bits (6) – Greed Quantifiers

		pattern	Input	Output
*?	Matches the previous element zero or more times, but as few times as possible.	<.*?>	<><b><c>	<> <b> <c>
+?	Matches the previous element one or more times, but as few times as possible.	<.+?>	<><b><c>	<><b> <c>
??	Matches the previous element zero or one time, but as few times as possible	<.??>	<><b><c>	<> <b> <c>

.Net – bits (6) – Greed Quantifiers

		pattern	Input	Output
{n}?	Matches the preceding element exactly n times.	<.{1}?>	<><b><c>	<b> <c>
{n,}?	Matches the previous element at least n times, but as few times as possible.	<.{1,}?>	<><b><ccc>	<><b> <ccc>
{n,m}?	Matches the previous element between n and m times, but as few times as possible.	<.{0,3}?> <.{1,3}?>	<><b><ccc> <><b><ccc>	<> <b> <ccc> <><b> <ccc>

Links (1)

- http://regexlib.com/ samples – 2834
- http://www.codeproject.com/KB/dotnet/expresso.aspx Tool to build regular expressions
- http://msdn.microsoft.com/en-us/library/ms974570.aspx Microsoft beefs up VBScript with regular expression – Intro (1999)
- http://msdn.microsoft.com/en-us/library/f97kw5ka(VS.85).aspx
  - Vbscript pattern

Links (2)

- http://msdn.microsoft.com/en-us/library/az24scfc.aspx
  - Regular expression language elements - .Net