VBSCRIPT - REGULAR EXPRESSIONS

What are Regular Expressions?

Regular Expressions is a sequence of characters that forms a pattern, which is mainly used for search and replace. The purpose of creating a pattern is to match specific strings, so that the developer can extract characters based on conditions and replace certain characters.

RegExp Object

RegExp object helps the developers to match the pattern of strings and the properties and methods help us to work with Regular Expressions easily. It is similar to RegExp in JavaScript

Properties

- **Pattern** - The Pattern method represents a string that is used to define the regular expression and it should be set before using the regular expression object.
- **IgnoreCase** - A Boolean property that represents if the regular expression should be tested against all possible matches in a string if true or false. If not specified explicitly, IgnoreCase value is set to False.
- **Global** - A Boolean property that represents if the regular expression should be tested against all possible matches in a string. If not specified explicitly, Global value is set to False.

Methods

- **Test**$search - string$ - The Test method takes a string as its argument and returns True if the regular expression can successfully be matched against the string, otherwise False is returned.
- **Replace**$search - string, replace - string$ - The Replace method takes 2 parameters. If the search is successful then it replaces that match with the replace-string, and the new string is returned. If there are no matches then the original search-string is returned.
- **Execute**$search - string$ - The Execute method works like Replace, except that it returns a Matches collection object, containing a Match object for each successful match. It doesn't modify the original string.

Matches Collection Object

The Matches collection object is returned as a result of the Execute method. This collection object can contain zero or more Match objects and the properties of this object are read-only.

- **Count** - The Count method represents the number of match objects in the collection.
- **Item** - The Item method enables the match objects to be accessed from matches collections object.

Match Object

The Match object is contained within the matches collection object. These objects represent the successful match after the search for a string.

- **FirstIndex** - It represents the position within the original string where the match occurred. This index are zero-based which means that the first position in a string is 0.
- **Length** - A value that represents the total length of the matched string.
- **Value** - A value that represents the matched value or text. It is also the default value when accessing the Match object.

All about Pattern Parameter

The pattern building is similar to PERL. Pattern building is the most important thing while working with Regular Expressions. In this section, we will deal with how to create a pattern based on various factors.

Position Matching

The significance of position matching is to ensure that we place the regular expressions at the correct places.

Symbol	Description
^	Matches only the beginning of a string.
$	Match only the end of a string.
\b	Matches any word boundary
\B	Matches any non-word boundary

Literals Matching

Any form of characters such as alphabet, number or special character or even decimal, hexadecimal can be treated as a Literal. Since few of the characters have already got a special meaning within the context of Regular Expression, we need to escape them using escape sequences.

Symbol	Description
Alphanumeric	Matches alphabetical and numerical characters only.
\n	Matches a new line.
\[	Matches [literal only
\]	Matches] literal only
\(	Matches (literal only
\)	Matches) literal only
\t	Matches horizontal tab
\v	Matches vertical tab
\\|	Matches | literal only
\{	Matches { literal only
\}	Matches } literal only
\\	Matches \ literal only
\?	Matches ? literal only
*	Matches * literal only
\+	Matches + literal only
\.	Matches . literal only
\b	Matches any word boundary

\B	Matches any non-word boundary
\f	Matches a form feed
\r	Matches carriage return
\xxx	Matches the ASCII character of an octal number xxx.
\xdd	Matches the ASCII character of an hexadecimal number dd.
\uxxxx	Matches the ASCII character of an UNICODE literal xxxx.

Character Classes Matching

The character classes are the Pattern formed by customized grouping and enclosed within [ ] braces. If we are expecting a character class that should not be in the list, then we should ignore that particular character class using the negative symobol, which is a cap ^.

Symbol	**Description**
[xyz]	Match any of the character class enclosed within the character set.
[^xyz]	Matches any of the character class that are NOT enclosed within the character set.
.	Matches any character class except \n
\w	Match any word character class. Equivalent to [a-zA-Z_0-9]
\W	Match any non-word character class. Equivalent to [^a-zA-Z_0-9]
\d	Match any digit class. Equivalent to [0-9].
\D	Match any non-digit character class. Equivalent to [^0-9].
\s	Match any space character class. Equivalent to [\t\r\n\v\f]
\S	Match any space character class. Equivalent to [^\t\r\n\v\f]

Repetition Matching

Repetition matching allows multiple searches within the regular expression. It also specifies the number of times an element is repeated in a Regular Expression.

Symbol	**Description**
*	Matches zero or more occurrences of the given regular Expression. Equivalent to {0,}.
+	Matches one or more occurrences of the given regular Expression. Equivalent to {1,}.
?	Matches zero or one occurrences of the given regular Expression. Equivalent to {0,1}.
{x}	Matches exactly x number of occurrences of the given regular expression.
{x,}	Match atleast x or more occurrences of the given regular expression.
{x,y}	Matches x to y number of occurences of the given regular expression.

Alternation & Grouping

Alternation and grouping helps developers to create more complex Regular Expressions in particularly handling intricate clauses within a Regular Expression which gives a great flexibility and control.

Symbol	Description
0	Grouping a clause to create a clause. "*xy*?*z*" matches "xyz" or "z".
|	Alternation combines one regular expression clause and then matches any of the individual clauses. "*ij*|23|*pq*" matches "ij" or "23" or "pq".

Building Regular Expressions

Below are few examples, which clearly explain on how to build a Regular Expression.

Regular Expression	Description
"^\s*.." and "..\s*$"	Represents that there can be any number of leading and trailing space characters in a single line.
"($\s?|#\s?)?"	Represents an optional $ or # sign followed by an optional space.
"(\d + (\.(\d\d?)?)?)"	Represents that at least one digit is present followed by an optional decimals and two digits after decimals.

Example

The below example checks whether or not the user entered an email id whose format should match such that there is an email id followed by '@' and then followed by domain name.

```
<!DOCTYPE html>
<html>
<body>
<script language="vbscript" type="text/vbscript">
  strid = "welcome.user@tutorialspoint.co.us"
  Set re = New RegExp
  With re
      .Pattern    = "^[\w-\.]{1,}\@([\da-zA-Z-]{1,}\.){1,}[\da-zA-Z-]{2,3}$"
      .IgnoreCase = False
      .Global     = False
  End With

  ' Test method returns TRUE if a match is found
  If re.Test( strid ) Then
      Document.write(strid & " is a valid e-mail address")
  Else
      Document.write(strid & " is NOT a valid e-mail address")
  End If

  Set re = Nothing
</script>
</body>
</html>
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js