Javascript Regular Expressions

2015-01-15 about keywords

As a JavaScript developer, I always write regex to do some magic job, but it is too powerful to create an expected expression always, so this post will help to give some brief description, and give a tool later along with the post.

Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects. These patterns are used with the exec and test methods of RegExp, and with the match, replace, search, and split methods of String. This chapter describes JavaScript regular expressions.

**Writing a Regular Expression Pattern**

A regular expression pattern is composed of simple characters, such as /abc/, or a combination of simple and special characters, such as /ab\*c/ or /Chapter (\d+)\.\d\*/. The last example includes parentheses which are used as a memory device. The match made with this part of the pattern is remembered for later use, as described in Using Parenthesized Substring Matches.

**Using Special Characters**

When the search for a match requires something more than a direct match, such as finding one or more b's, or finding white space, the pattern includes special characters. For example, the pattern /ab\*c/ matches any character combination in which a single 'a' is followed by zero or more 'b's (\* means 0 or more occurrences of the preceding item) and then immediately followed by 'c'. In the string "cbbabbbbcdebc," the pattern matches the substring 'abbbbc'.

The following table provides a complete list and description of the special characters that can be used in regular expressions.

| **Character** | **Meaning** |
| --- | --- |
| [\](http://www.getzipweb.com/" \l "special-backslash) | Matches according to the following rules: A backslash that precedes a non-special character indicates that the next character is special and is not to be interpreted literally. For example, a 'b' without a preceding '\' generally matches lowercase 'b's wherever they occur. But a '\b' by itself doesn't match any character; it forms the special [word boundary character](http://www.getzipweb.com/#special-word-boundary). A backslash that precedes a special character indicates that the next character is not special and should be interpreted literally. For example, the pattern /a\*/ relies on the special character '\*' to match 0 or more a's. By contrast, the pattern /a\\*/ removes the specialness of the '\*' to enable matches with strings like 'a\*'. Do not forget to escape \ itself while using the RegExp("pattern") notation because \ is also an escape character in strings. |
| [^](http://www.getzipweb.com/" \l "special-caret) | Matches beginning of input. If the multiline flag is set to true, also matches immediately after a line break character. For example, /^A/ does not match the 'A' in "an A", but does match the 'A' in "An E". The '^' has a different meaning when it appears as the first character in a character set pattern. See [complemented character sets](http://www.getzipweb.com/#special-negated-character-set) for details and an example. |
| [$](http://www.getzipweb.com/" \l "special-dollar) | Matches end of input. If the multiline flag is set to true, also matches immediately before a line break character. For example, /t$/ does not match the 't' in "eater", but does match it in "eat". |
| [\*](http://www.getzipweb.com/" \l "special-asterisk) | Matches the preceding character 0 or more times. Equivalent to {0,}. For example, /bo\*/ matches 'boooo' in "A ghost booooed" and 'b' in "A bird warbled", but nothing in "A goat grunted". |
| [+](http://www.getzipweb.com/" \l "special-plus) | Matches the preceding character 1 or more times. Equivalent to {1,}. For example, /a+/ matches the 'a' in "candy" and all the a's in "caaaaaaandy". |
| [?](http://www.getzipweb.com/" \l "special-questionmark) | Matches the preceding character 0 or 1 time. Equivalent to {0,1}. For example, /e?le?/ matches the 'el' in "angel" and the 'le' in "angle" and also the 'l' in "oslo". If used immediately after any of the quantifiers \*, +, ?, or {}, makes the quantifier non-greedy (matching the fewest possible characters), as opposed to the default, which is greedy (matching as many characters as possible). For example, applying /\d+/ to "123abc" matches "123". But applying /\d+?/ to that same string matches only the "1". Also used in lookahead assertions, as described in the x(?=y) and x(?!y) entries of this table. |
| [.](http://www.getzipweb.com/" \l "special-dot) | (The decimal point) matches any single character except the newline character. For example, /.n/ matches 'an' and 'on' in "nay, an apple is on the tree", but not 'nay'. |
| [(x)](http://www.getzipweb.com/" \l "special-capturing-parentheses) | Matches 'x' and remembers the match, as the following example shows. The parentheses are called *capturing parentheses*. The '(foo)' and '(bar)' in the pattern /(foo) (bar) \1 \2/ match and remember the first two words in the string "foo bar foo bar". The \1 and \2 in the pattern match the string's last two words. Note that \1, \2, \n are used in the matching part of the regex. In the replacement part of a regex the syntax $1, $2, $n must be used, e.g.: 'bar foo'.replace( /(...) (...)/, '$2 $1' ). |
| [(?:x)](http://www.getzipweb.com/" \l "special-non-capturing-parentheses) | Matches 'x' but does not remember the match. The parentheses are called *non-capturing parentheses*, and let you define subexpressions for regular expression operators to work with. Consider the sample expression /(?:foo){1,2}/. Without the non-capturing parentheses, the {1,2} characters would apply only to the last 'o' in 'foo'. With the non-capturing parentheses, the {1,2} applies to the entire word 'foo'. |
| [x(?=y)](http://www.getzipweb.com/" \l "special-lookahead) | Matches 'x' only if 'x' is followed by 'y'. This is called a lookahead. For example, /Jack(?=Sprat)/ matches 'Jack' only if it is followed by 'Sprat'. /Jack(?=Sprat|Frost)/ matches 'Jack' only if it is followed by 'Sprat' or 'Frost'. However, neither 'Sprat' nor 'Frost' is part of the match results. |
| [x(?!y)](http://www.getzipweb.com/" \l "special-negated-look-ahead) | Matches 'x' only if 'x' is not followed by 'y'. This is called a negated lookahead. For example, /\d+(?!\.)/ matches a number only if it is not followed by a decimal point. The regular expression /\d+(?!\.)/.exec("3.141") matches '141' but not '3.141'. |
| [x|y](http://www.getzipweb.com/" \l "special-or) | Matches either 'x' or 'y'. For example, /green|red/ matches 'green' in "green apple" and 'red' in "red apple." |
| [{n}](http://www.getzipweb.com/" \l "special-quantifier) | Matches exactly n occurrences of the preceding character. N must be a positive integer. For example, /a{2}/ doesn't match the 'a' in "candy," but it does match all of the a's in "caandy," and the first two a's in "caaandy." |
| [{n,m}](http://www.getzipweb.com/" \l "special-quantifier-range) | Where n and m are positive integers. Matches at least n and at most m occurrences of the preceding character. When m is zero, it can be omitted. For example, /a{1,3}/ matches nothing in "cndy", the 'a' in "candy," the first two a's in "caandy," and the first three a's in "caaaaaaandy" Notice that when matching "caaaaaaandy", the match is "aaa", even though the original string had more a's in it. |
| [[xyz]](http://www.getzipweb.com/" \l "special-character-set) | Character set. This pattern type matches any one of the characters in the brackets, including [escape sequences](https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Values,_variables,_and_literals#Unicode_escape_sequences). Special characters like the dot(.) and asterisk (\*) are not special inside a character set, so they don't need to be escaped. You can specify a range of characters by using a hyphen, as the following examples illustrate. The pattern [a-d], which performs the same match as [abcd], matches the 'b' in "brisket" and the 'c' in "city". The patterns /[a-z.]+/ and /[\w.]+/ match the entire string "test.i.ng". |
| [[^xyz]](http://www.getzipweb.com/" \l "special-negated-character-set) | A negated or complemented character set. That is, it matches anything that is not enclosed in the brackets. You can specify a range of characters by using a hyphen. Everything that works in the normal character set also works here. For example, [^abc] is the same as [^a-c]. They initially match 'r' in "brisket" and 'h' in "chop." |
| [[\b]](http://www.getzipweb.com/" \l "special-backspace) | Matches a backspace (U+0008). You need to use square brackets if you want to match a literal backspace character. (Not to be confused with \b.) |
| [\b](http://www.getzipweb.com/" \l "special-word-boundary) | Matches a word boundary. A word boundary matches the position where a word character is not followed or preceeded by another word-character. Note that a matched word boundary is not included in the match. In other words, the length of a matched word boundary is zero. (Not to be confused with [\b].) Examples: /\bm/ matches the 'm' in "moon" ; /oo\b/ does not match the 'oo' in "moon", because 'oo' is followed by 'n' which is a word character; /oon\b/ matches the 'oon' in "moon", because 'oon' is the end of the string, thus not followed by a word character; /\w\b\w/ will never match anything, because a word character can never be followed by both a non-word and a word character. |
| [\B](http://www.getzipweb.com/" \l "special-non-word-boundary) | Matches a non-word boundary. This matches a position where the previous and next character are of the same type: Either both must be words, or both must be non-words. The beginning and end of a string are considered non-words. For example, /\B../ matches 'oo' in "noonday", and /y\B./ matches 'ye' in "possibly yesterday." |
| [\c](http://www.getzipweb.com/" \l "special-control)*[X](http://www.getzipweb.com/" \l "special-control)* | Where *X* is a character ranging from A to Z. Matches a control character in a string. For example, /\cM/ matches control-M (U+000D) in a string. |
| [\d](http://www.getzipweb.com/" \l "special-digit) | Matches a digit character. Equivalent to [0-9]. For example, /\d/ or /[0-9]/ matches '2' in "B2 is the suite number." |
| [\D](http://www.getzipweb.com/" \l "special-non-digit) | Matches any non-digit character. Equivalent to [^0-9]. For example, /\D/ or /[^0-9]/ matches 'B' in "B2 is the suite number." |
| [\f](http://www.getzipweb.com/" \l "special-form-feed) | Matches a form feed (U+000C). |
| [\n](http://www.getzipweb.com/" \l "special-line-feed) | Matches a line feed (U+000A). |
| [\r](http://www.getzipweb.com/" \l "special-carriage-return) | Matches a carriage return (U+000D). |
| [\s](http://www.getzipweb.com/" \l "special-white-space) | Matches a single white space character, including space, tab, form feed, line feed. Equivalent to [ \f\n\r\t\v​\u00a0\u1680​\u180e\u2000​\u2001\u2002​\u2003\u2004​\u2005\u2006​\u2007\u2008​\u2009\u200a​\u2028\u2029​​\u202f\u205f​\u3000]. For example, /\s\w\*/ matches ' bar' in "foo bar." |
| [\S](http://www.getzipweb.com/" \l "special-non-white-space) | Matches a single character other than white space. Equivalent to [^ \f\n\r\t\v​\u00a0\u1680​\u180e\u2000​\u2001\u2002​\u2003\u2004​\u2005\u2006​\u2007\u2008​\u2009\u200a​\u2028\u2029​\u202f\u205f​\u3000]. For example, /\S\w\*/ matches 'foo' in "foo bar." |
| [\t](http://www.getzipweb.com/" \l "special-tab) | Matches a tab (U+0009). |
| [\v](http://www.getzipweb.com/" \l "special-vertical-tab) | Matches a vertical tab (U+000B). |
| [\w](http://www.getzipweb.com/" \l "special-word) | Matches any alphanumeric character including the underscore. Equivalent to [A-Za-z0-9\_]. For example, /\w/ matches 'a' in "apple," '5' in "$5.28," and '3' in "3D." |
| [\W](http://www.getzipweb.com/" \l "special-non-word) | Matches any non-word character. Equivalent to [^A-Za-z0-9\_]. For example, /\W/ or /[^A-Za-z0-9\_]/ matches '%' in "50%." |
| [\](http://www.getzipweb.com/" \l "special-backreference)*[n](http://www.getzipweb.com/" \l "special-backreference)* | Where *n* is a positive integer, a back reference to the last substring matching the *n* parenthetical in the regular expression (counting left parentheses). For example, /apple(,)\sorange\1/ matches 'apple, orange,' in "apple, orange, cherry, peach." |
| [\0](http://www.getzipweb.com/" \l "special-null) | Matches a NULL (U+0000) character. Do not follow this with another digit, because \0 is an octal [escape sequence](https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Values,_variables,_and_literals#Unicode_escape_sequences). |
| [\xhh](http://www.getzipweb.com/" \l "special-hex-escape) | Matches the character with the code hh (two hexadecimal digits) |
| [\uhhhh](http://www.getzipweb.com/" \l "special-unicode-escape) | Matches the character with the code hhhh (four hexadecimal digits). |

Advanced Searching With Flags

Regular expressions have four optional flags that allow for global and case insensitive searching. These flags can be used separately or together in any order, and are included as part of the regular expression.

| Flag | Description |
| --- | --- |
| g | Global search. |
| i | Case-insensitive search. |
| m | Multi-line search. |
| y | Perform a "sticky" search that matches starting at the current position in the target string. |