Day 1: RegExp

by AvimanyuSingh

Regular expressions (RegExp) are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects.

Creating a regular expression

Regular expressions can be constructed in two ways:

- Using a regular expression literal, which consists of a pattern enclosed between slashes:

```
var re = /ab+c/;
```

- Using the constructor function of the RegExp object:

```
var re = new RegExp("ab+c");
```

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*/*.

Syntax:

```
/pattern/flags
new RegExp(pattern[, flags])
```

flags

If specified, flags can have any combination of the following values:

- *g*: global match.
- *i*: ignore case.
- *m*: multiline. Treats beginning (^) and end ($) characters as working over multiple lines.
- *u*: unicode. Treat pattern as a sequence of unicode code points.
- *y*: sticky. Matches only from the index indicated by the lastIndex property of this regular expression in the target string.

Special characters in regular expressions

- *Character Classes*
- *Character Sets*
- *Alteration*
- *Boundaries*
- *Grouping and back references*
- *Quantifiers*
- *Assertions*

Character Classes

. (dot): Matches any single character except line terminators.

\d: Matches a digit character in the basic Latin alphabet. Equivalent to [0-9].

\D: Matches any character that is not a digit in the basic Latin alphabet. Equivalent to [^0-9].

\w: Matches any alphanumeric character from the basic Latin alphabet, including the underscore. Equivalent to [A-Za-z0-9_].

\W: Matches any character that is not a word character from the basic Latin alphabet. Equivalent to [^A-Za-z0-9_].

\s: Matches a single white space character, including space, tab, form feed, line feed and other Unicode spaces.

\S: Matches a single character other than white space.

Character Sets

[abcd]: Matches any one character from the set {a, b, c, d}. Equivalent to [a-d].

[^abcd]: Matches anything other than the enclosed characters. Equivalent to [^a-d].

Alteration

a|b: matches either a or b.

Boundaries

^: Matches beginning of input. If the multiline flag is set to true, also matches immediately after a line break character.

$: Matches end of input. If the multiline flag is set to true, also matches immediately before a line break character.

\b: Matches a zero-width word boundary, such as between a letter and a space.

\B: Matches a zero-width non-word boundary, such as between two letters or between two spaces.

Grouping and back references

(a): Matches a and remembers the match. These are called capturing groups.

(?:a): Matches a but does not remember the match. These are called non-capturing groups.

\n: Here n is a positive integer. A back reference to the last substring matching the n parenthetical in the regular expression.

Quantifiers

*a**: Matches the preceding item a, 0 or more times.

a+: Matches the preceding item a, 1 or more times.

a?: Matches the preceding item a, 0 or 1 time.

a{n}: Here n is a positive integer. Matches exactly n occurrences of the preceding item a.

a{n, }: Here n is a positive integer. Matches at least n occurrences of the preceding item a.

a{n, m}: Here n and m are positive integers. Matches at least n and at most m occurrences of the preceding item a.

Assertions

a(?=b): Matches a only if a is followed by b.

a(?!b): Matches a only if a is not followed by b.

Working with regular expressions

Regular expressions are used with the RegExp methods:

- *test*
- *exec*

and with the String methods:

- *match*
- *search*
- *split*
- *replace*

test

The test() method executes a search for a match between a regular expression and a specified string. Returns true or false.

```
// Test whether "learn" is contained at the very beginning of a string

var re = /^learn/;
var str1 = "learn regular expressions";
var str2 = "write regular expressions";

console.log(re.test(str1));
console.log(re.test(str2));
```

The code output is:

```
true
false
```

exec

The exec() method executes a search for a match in a specified string. Returns a result array or null.

```
// Match "quick brown" followed by "jumps", ignoring characters in between
// Remember "brown" and "jumps"
// Ignore case

var re = /quick\s(brown).+?(jumps)/ig;
var str = "The Quick Brown Fox Jumps Over The Lazy Dog.";
var res = re.exec(str);

console.log(res);
console.log();

// The result object contains following information:
// 1. [0] is the full string of characters matched
// 2. [1], ...[n] is the parenthesized substring matches, if any. The number of possible parenthesized substrings i
s unlimited.
// 3. index     is the 0-based index of the match in the string.
// 4. input is the original string.

console.log("string of characters matched = " + res[0]);
console.log("first parenthesized substring match = " + res[1]);
console.log("second parenthesized substring match = " + res[2]);
console.log("index of the match = " + res.index);
console.log("original string = " + res.input);
```

The code output is:

```
[ 'Quick Brown Fox Jumps',
  'Brown',
  'Jumps',
  index: 4,
  input: 'The Quick Brown Fox Jumps Over The Lazy Dog.' ]

string of characters matched = Quick Brown Fox Jumps
first parenthesized substring match = Brown
second parenthesized substring match = Jumps
index of the match = 4
original string = The Quick Brown Fox Jumps Over The Lazy Dog.
```

match

The match() method retrieves the matches when matching a string against a regular expression.

```
// Find 'Chapter' followed by 1 or more numeric characters followed by a decimal point and numeric character 0 or m
ore times.
// Ignore case

var re = /see (chapter \d+(\.\d)*)/i;
var str = 'For more information on regular expressions, see Chapter 3.4.5.1';

console.log(str.match(re));
```

The code output is:

```
[ 'see Chapter 3.4.5.1',
  'Chapter 3.4.5.1',
  '.1',
  index: 45,
  input: 'For more information on regular expressions, see Chapter 3.4.5.1' ]
```

search

The search() method executes a search for a match between a regular expression and this String object. If successful, search() returns the index of the first match of the regular expression inside the string. Otherwise, it returns -1.

```
// Test whether "learn" is contained in a string

var re = /learn/;
var str1 = "Today, we will learn regular expressions";
var str2 = "Tomorrow, we will write regular expressions and learn some complex expressions";
var str3 = "We are done now";

console.log(str1.search(re));
console.log(str2.search(re));
console.log(str3.search(re));
```

The code output is:

```
15
48
-1
```

split

The split() method splits a String object into an array of strings by separating the string into substrings. Separator specifies the character(s) to use for separating the string. The separator is treated as a string or a regular expression. If separator is omitted, the array returned contains one element consisting of the entire string. If separator is an empty string, str is converted to an array of characters.

```
// Split the first and last names, which are separated by space.

var str = "Julia Roberts";
var res = str.split(" ");

console.log("First name is = " + res[0]);
console.log("Last name is = " + res[1]);
```

The code output is:

```
First name is = Julia
Last name is = Roberts
```

replace

The replace() method returns a new string with some or all matches of a pattern replaced by a replacement. The pattern can be a string or a RegExp, and the replacement can be a string or a function to be called for each match.

```
// Replace "RegExp" by "Regular Expression"

var re = /RegExp/;
var str = "Learn about RegExp.";
var rpl = "Regular Expression";

console.log(str.replace(re, rpl));
```

The code output is:

```
Learn about Regular Expression.
```