MDN Web Docs
moz://a

String.prototype.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. If `pattern` is a string, only the first occurrence will be replaced.

The original string is left unchanged.

JavaScript Demo: String.replace()

```
const p = 'The quick brown fox jumps over the lazy dog. If the dog reacted, was

console.log(p.replace('dog', 'monkey'));
// expected output: "The quick brown fox jumps over the lazy monkey. If the dog


const regex = /Dog/i;
console.log(p.replace(regex, 'ferret'));
// expected output: "The quick brown fox jumps over the lazy ferret. If the dog

```

Run ›

Reset

Syntax

```
replace(regexp, newSubstr)
replace(regexp, replacerFunction)

replace(substr, newSubstr)
replace(substr, replacerFunction)
```

Parameters

Parameters

`regexp` (pattern)

A `RegExp` object or literal. The match or matches are replaced with `newSubstr` or the value returned by the specified `replacerFunction`.

`substr`

A `String` that is to be replaced by `newSubstr`. It is treated as a literal string and is *not* interpreted as a regular expression. Only the first occurrence will be replaced.

`newSubstr` (replacement)

The `String` that replaces the substring specified by the specified `regexp` or `substr` parameter. A number of special replacement patterns are supported; see the "Specifying a string as a parameter" section below.

If `newSubstr` is an empty string, then the substring given by `substr`, or the matches for `regexp`, are removed (rather then being replaced).

`replacerFunction` (replacement)

A function to be invoked to create the new substring to be used to replace the matches to the given `regexp` or `substr`. The arguments supplied to this function are described in the "Specifying a function as a parameter" section below.

Return value

A new string, with some or all matches of a pattern replaced by a replacement.

Description

This method does not change the calling `String` object. It returns a new string.

To perform a global search and replace, include the `g` switch in the regular expression.

Specifying a string as a parameter

The replacement string can include the following special replacement patterns:

Pattern	Inserts
`$$`	Inserts a `"$"`.
`$&`	Inserts the matched substring.
`$` `	Inserts the portion of the string that precedes the matched substring.

Pattern	Inserts
`$'`	Inserts the portion of the string that follows the matched substring.

`$n`	Where `n` is a positive integer less than 100, inserts the `n` th parenthesized submatch string, provided the first argument was a `RegExp` object. Note that this is `1` -indexed. If a group `n` is not present (e.g., if group is 3), it will be replaced as a literal (e.g., `$3`).
`$<Name>`	Where `Name` is a capturing group name. If the group is not in the match, or not in the regular expression, or if a string was passed as the first argument to `replace` instead of a regular expression, this resolves to a literal (e.g., `$<Name>`). Only available in browser versions supporting named capturing groups.

Specifying a function as a parameter

You can specify a function as the second parameter. In this case, the function will be invoked after the match has been performed. The function's result (return value) will be used as the replacement string. (**Note:** The above-mentioned special replacement patterns do *not* apply in this case.)

Note that the function will be invoked multiple times for each full match to be replaced if the regular expression in the first parameter is global.

The arguments to the function are as follows:

Possible name	Supplied value
`match`	The matched substring. (Corresponds to `$&` above.)
`p1, p2, ...`	The *n*th string found by a parenthesized capture group (including named capturing groups), provided the first argument to `replace()` was a `RegExp` object. (Corresponds to `$1` , `$2` , etc. above.) For example, if `/(\a+)(\b+)/` , was given, `p1` is the match for `\a+` , and `p2` for `\b+` .
`offset`	The offset of the matched substring within the whole string being examined. (For example, if the whole string was `'abcd'` , and the matched substring was `'bc'` , then this argument will be `1` .)

Possible name	Supplied value
`string`	The whole string being examined.

`groups`	In browser versions supporting named capturing groups, will be an object whose keys are the used group names, and whose values are the matched portions (`undefined` if not matched).

(The exact number of arguments depends on whether the first argument is a `RegExp` object—and, if so, how many parenthesized submatches it specifies.)

The following example will set `newString` to `'abc - 12345 - #$*%'`:

```
function replacer(match, p1, p2, p3, offset, string) {
  // p1 is nondigits, p2 digits, and p3 non-alphanumerics
  return [p1, p2, p3].join(' - ');
}
let newString = 'abc12345#$*%'.replace(/([^\d]*)(\d*)([^\w]*)/, replacer);
console.log(newString);  // abc - 12345 - #$*%
```

Examples

Defining the regular expression in replace()

In the following example, the regular expression is defined in `replace()` and includes the ignore case flag.

```
let str = 'Twas the night before Xmas...';
let newstr = str.replace(/xmas/i, 'Christmas');
console.log(newstr);  // Twas the night before Christmas...
```

This logs `'Twas the night before Christmas...'`.

> **Note:** See this guide for more explanations about regular expressions.

Using global and ignore with replace()

Global replace can only be done with a regular expression. In the following example, the regular expression includes the global and ignore case flags which permits `replace()` to

regular expression includes the global and ignore case flags which permits `replace()` to replace each occurrence of `'apples'` in the string with `'oranges'`.

```
let re = /apples/gi;
let str = 'Apples are round, and apples are juicy.';

let newstr = str.replace(re, 'oranges');
console.log(newstr);  // oranges are round, and oranges are juicy.
```

This logs `'oranges are round, and oranges are juicy'`.

Switching words in a string

The following script switches the words in the string. For the replacement text, the script uses capturing groups and the `$1` and `$2` replacement patterns.

```
let re = /(\w+)\s(\w+)/;
let str = 'John Smith';
let newstr = str.replace(re, '$2, $1');
console.log(newstr);  // Smith, John
```

This logs `'Smith, John'`.

Using an inline function that modifies the matched characters

In this example, all occurrences of capital letters in the string are converted to lower case, and a hyphen is inserted just before the match location. The important thing here is that additional operations are needed on the matched item before it is given back as a replacement.

The replacement function accepts the matched snippet as its parameter, and uses it to transform the case and concatenate the hyphen before returning.

```
function styleHyphenFormat(propertyName) {
  function upperToHyphenLower(match, offset, string) {
    return (offset > 0 ? '-' : '') + match.toLowerCase();
  }
  return propertyName.replace(/[A-Z]/g, upperToHyphenLower);
}
```

Given `styleHyphenFormat('borderTop')`, this returns `'border-top'`.

Because we want to further transform the *result* of the match before the final substitution is made, we must use a function. This forces the evaluation of the match prior to the `toLowerCase()` method. If we had tried to do this using the match without a function, the `toLowerCase()` would have no effect.

```
let newString = propertyName.replace(/[A-Z]/g, '-' + '$&'.toLowerCase();
```

This is because `'$&'.toLowerCase()` would first be evaluated as a string literal (resulting in the same `'$&'`) before using the characters as a pattern.

Replacing a Fahrenheit degree with its Celsius equivalent

The following example replaces a Fahrenheit degree with its equivalent Celsius degree. The Fahrenheit degree should be a number ending with `"F"`. The function returns the Celsius number ending with `"C"`. For example, if the input number is `"212F"`, the function returns `"100C"`. If the number is `"0F"`, the function returns `"-17.77777777777778C"`.

The regular expression `test` checks for any number that ends with `F`. The number of Fahrenheit degree is accessible to the function through its second parameter, `p1`. The function sets the Celsius number based on the Fahrenheit degree passed in a string to the `f2c()` function. `f2c()` then returns the Celsius number. This function approximates Perl's `s///e` flag.

```
function f2c(x) {
  function convert(str, p1, offset, s) {
    return ((p1 - 32) * 5/9) + 'C';
  }
  let s = String(x);
  let test = /(-?\d+(?:\.\d*)?)F\b/g;
  return s.replace(test, convert);
}
```

Specifications

Specification
ECMAScript Language Specification (ECMAScript) # sec-string.prototype.replace

Browser compatibility

Report problems with this compatibility data on GitHub

replace

Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

Full support

Partial support

★ See implementation notes.

See also

- `String.prototype.replaceAll()`
- `String.prototype.match()`
- `RegExp.prototype.exec()`
- `RegExp.prototype.test()`

Last modified: Jul 20, 2021, by MDN contributors