JavaScript Regex Replace

Summary: in this tutorial, you'll learn how to use the string replace() method to return a new string with some or all matches of a regular expression replaced by a replacement string.

Introduction to the JavaScript replace() method

The `String.prototype.replace()` method works with both strings and regular expressions. This tutorial focuses solely on regular expressions.

The following shows the syntax of the `replace()` method:

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

In this syntax:

- The `regexp` is a regular expression to match.
- The `newSubstr` is a string to replace the matches. If the `newSubstr` is empty, the `replace()` method removes the matches.

The `replace()` returns a new string with the matches replaced by the `newSubstr`. Note that the `replace()` method doesn't change the original string but returns a new string.

By default, the `replace()` method replaces the first match if the `regexp` doesn't use the global flag (`g`). To replace all matches, you use the global flag (`g`) in the `regexp`.

JavaScript regex replace() method examples

Let's take some examples of using the `replace()` method.

1) A simple the JavaScript regex replace() method example

The following example uses the `replace()` method to replace the first match of the `JS` string with the `JavaScript` string:

```
const s = 'JS and js';
const re = /js/i;

const newS = s.replace(re, 'JavaScript');
console.log(newS);
```

Output:

```
JavaScript and js
```

The `/js/i` matches both `JS` and `js` in the `'JS and js'` string. However, the `replace()` method replaces only the first match (`JS`).

To replace all matches, you use the global flag (`g`) in the regular expression.

2) Using the JavaScript regex replace() method with the global flag

The following example uses the `replace()` method with a regular expression containing a global flag (`g`) to replace all matches:

```
const s = 'JS and js';
const re = /js/gi;

const newS = s.replace(re, 'JavaScript');
console.log(newS);
```

Output:

```
JavaScript and JavaScript
```

3) Using the JavaScript regex replace() method with capturing groups

When a regular expression contains the capturing groups, you can reference these groups in the `newSubstr` using the `$N` syntax where `N` is the grouping number. For example, $1 and $2 reference first and second capturing groups.

The following example illustrates how to use the `replace()` method with capturing groups to swap the first and last names in a person name:

```
let re = /(\w+)\s(\w+)/;
let name = 'Jane Doe';
let lastFirst = name.replace(re, '$2, $1');

console.log(lastFirst);
```

Output:

```
Doe, Jane
```

How it works.

The regular expression `/(\w+)\s(\w+)/` matches one or more word characters, a space, and then one or more word characters. In other words, it matches any string that has a word, space, and another word.

The regular expression contains two capturing groups. The first capturing group captures the first word and the second one captures the second word after the space.

In the `newSubstr`, we use `$1` to reference the first capturing group and `$2` to reference the second one. To swap the first name and last name, we place the second match (`$2`) first and then the first match (`$1`).

JavaScript regex replace() method with replacer function

The second argument of the `replace()` method can be a function like this:

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

The `replace()` method calls the `replacerFunction` after it finds the first match. The `replacerFunction` is used to create a substring to replace the match.

If the `regexp` uses the global flag (`g`), the `replace()` method will call the `replacerFunction` after every match.

The `replacerFunction` has the following arguments:

- `match` specifies the matched substring.
- `p1` , `p2` , ... the values of the capturing groups in the regexp.
- `offset` is an integer that specifies the offset of the matched substring within the input string.
- `string` is the input string.
- `groups` is an object whose are are the named capturing group and values are matched values.

Let's take an example of using the `replace()` method with a replacer function.

Suppose you have a string like this:

```
backgroundColor
```

And you want to transform it into something like:

```
background-color
```

To do that you can use the `replace()` method with a replacer function.

First, construct a regular expression that matches a capital letter:

```
/[A-Z]/g
```

Second, define a replacer function:

```
function replacer(match, offset) {
  return (offset > 0 ? '-' : '') + match.toLowerCase();
}
```

The `replacer()` function adds a hyphen if the matched letter is not at the beginning of the string and concatenates the hyphen with the matched letter converted to lowercase.

Third, use the `replace()` method to replace the match with the substring returned from the `replacer()` function:

```
function addHyphen(prop) {
  return prop.replace(/[A-Z]/g, replacer);
}
```

The following shows the complete code:

```
function replacer(match, offset) {
  return (offset > 0 ? '-' : '') + match.toLowerCase();
}

function addHyphen(prop) {
  return prop.replace(/[A-Z]/g, replacer);
}

const prop = 'backgroundColor';
console.log(addHyphen(prop));
```

Output:

```
background-color
```

To make the code more concise, you can use arrow functions with the replacer function as a callback function like this:

```
const addHyphen = (prop) =>
  prop.replace(
    /[A-Z]/g,
    (match, offset) => (offset > 0 ? '-' : '') + match.toLowerCase()
  );

const prop = 'backgroundColor';
console.log(addHyphen(prop));
```

Summary

- Use the `replace()` method to find matches against a regular expression and replace the matches with a new substring.