JavaScript String Methods Reference - Impressive Webs

By Louis Lazaris on July 2nd, 2012 | 24 Comments

When writing JavaScript, I often find myself Googling for info, or using Mozilla's reference to find the exact syntax and argument definitions for methods associated with string manipulation.

A lot of references I come across have far too much info, so this post will give examples and brief descriptions of some of the most common and useful string-related methods. I tried to put the most common ones near the top, for quick reference.

Of course, most experienced developers will be quite familiar with many of these, but I think this is a good list for beginners to understand the range of methods available that can help accomplish complex operations using simple syntax.

Convert to String

You can convert a number, Boolean, or an object to a string:

```
var myNumber = 24; // 24
var myString = myNumber.toString(); // "24"
```

As pointed out in the comments, you can also do it with `String()`:

```
var myNumber = 24; // 24
var myString = String(myNumber); // "24"
```

Nicholas Zakas, in Professional JavaScript for Web Developers, says: "If you're not sure that a value isn't `null` or `undefined`, you can use the `String()` casting function, which always returns a string regardless of the value type."

Split a String Into Multiple Substrings

To separate a string into an array of substrings, you can use the `split()` method:

```
var myString = "coming,apart,at,the,commas";
var substringArray = myString.split(","); // ["coming", "apart", "at", "the", "commas"]
var arrayLimited = myString.split(",", 3); // ["coming", "apart", "at"]
```

As shown in the last line, a second optional argument limits the array to the number of items specified by the argument.

Get the Length of a String

To find out how many characters long a string is, use the `length` property:

```
var myString = "You're quite a character.";
var stringLength = myString.length; // 25
```

Note: For any method that deals with individual characters, see Mathias' comment and his article on JavaScript encoding.

Locate a Substring within a String

There are two methods for doing this.

Using `indexOf()`:

```
var stringOne = "Johnny Waldo Harrison Waldo";
```

```
var wheresWaldo = stringOne.indexOf("Waldo"); // 7
```

The `indexOf()` method starts searching for the substring (the first argument passed in) from the beginning of the string, and returns the position of the start of the first occurrence of the substring.

Using `lastIndexOf()`:

```
var stringOne = "Johnny Waldo Harrison Waldo";
var wheresWaldo = stringOne.lastIndexOf("Waldo"); // 22
```

The `lastIndexOf()` method is exactly the same except it returns the starting position of the last occurrence of the passed in substring.

In both methods, if the substring is not found, the returned value is `-1`, and both allow an optional second argument indicating the position in the string where you want to start the search. So with a second argument of “5”, `indexOf()` starts searching at character 5, ignoring characters 0-4, while `lastIndexOf()` starts searching at character 5 and goes in reverse, ignoring characters 6 and above.

Replace a Substring

To replace part or all of a string with a new string, you can use `replace()`:

```
var slugger = "Josh Hamilton";
var betterSlugger = slugger.replace("h Hamilton", "e Bautista");
console.log(betterSlugger); // "Jose Bautista"
```

The first argument is the substring you want to replace, and the second argument is the new substring. This will only replace the first instance of the matched substring.

To replace all instances of the matched substring, use a regular expression with the global flag:

```
var myString = "She sells automotive shells on the automotive shore";
var newString = myString.replace(/automotive/g, "sea");
console.log(newString); // "She sells sea shells on the sea shore"
```

The second argument can include special replacement patterns or can be a function. More info on those options on MDN’s reference.

Find the Character at a Given Position

To find out which character is at a specified position, you can use `charAt()`:

```
var myString = "Birds of a Feather";
var whatsAtSeven = myString.charAt(7); // "f"
```

As is often the case in JavaScript, the first position in the string is referenced with “0”, instead of “1”.

Alternatively, you could use `charCodeAt()`, which gives you the character code, instead of the character itself:

```
var myString = "Birds of a Feather";
var whatsAtSeven = myString.charCodeAt(7); // "102"
var whatsAtEleven = myString.charCodeAt(11); // "70"
```

Notice that the character code for the uppercase “F” (position 11) is different from the character code for the lowercase “f” (position 7).

Concatenating Multiple Strings

For the most part, when you concatenate strings, you’ll use the addition operator (+). But you do also have the option to use the `concat()` method:

```
var stringOne = "Knibb High football ";
var stringTwo = stringOne.concat("rules."); // "Knibb High football rules"
```

You can also pass multiple strings into it, and all will be appended (in the order they appear) to the original string:

```
var stringOne = "Knibb ";
var stringTwo = "High ";
```

```
var stringThree = "football ";
var stringFour = "rules.";
var finalString = stringOne.concat(stringTwo, stringThree, stringFour);
console.log(finalString); // "Knibb High football rules."
```

Slice a String (Extract a Substring)

There are three different ways to create a new string value from part of another string:

Using `slice()`:

```
var stringOne = "abcdefghijklmnopqrstuvwxyz";
var stringTwo = stringOne.slice(5, 10); // "fghij"
```

Using `substring()`:

```
var stringOne = "abcdefghijklmnopqrstuvwxyz";
var stringTwo = stringOne.substring(5, 10); // "fghij"
```

For both `slice()` and `substring()`, the first argument is the character at which to begin the substring (again using zero-based numbering) and the second argument (which is optional) is the character in the string *after* you want the substring to end. So in the examples above, the arguments “5, 10” mean characters 5 through 9 are “sliced” to create the new string.

Using `substr()`

```
var stringOne = "abcdefghijklmnopqrstuvwxyz";
var stringTwo = stringOne.substr(5, 10); // "fghijklmno"
```

For `substr()`, once again the first argument represents the character that begins the new string and the second argument is optional. But this time, the second argument represents the total number of characters that should be included, starting with the character in the “5” position.

Convert a String to Uppercase or Lowercase

There are four methods that do case conversion. Two for converting a string to all uppercase:

```
var stringOne = "Speak up, I can't hear you.";
var stringTwo = stringOne.toLocaleUpperCase(); // "SPEAK UP, I CAN'T HEAR YOU"
var stringThree = stringOne.toUpperCase(); // "SPEAK UP, I CAN'T HEAR YOU"
```

And two for converting a string to lower case:

```
var stringOne = "YOU DON'T HAVE TO YELL";
var stringTwo = stringOne.toLocaleLowerCase(); // "you don't have to yell"
var stringThree = stringOne.toLowerCase(); // "you don't have to yell"
```

Generally, the results between the “locale” method and the non-locale method are the same, but according to MDN’s reference “for some locales, such as Turkish, whose case mappings do not follow the default case mappings in Unicode, there may be a different result.” Zakas’ book says “if you do not know the language in which the code will be running, it is safer to use the locale-specific methods.”

Pattern Matching

Matching a pattern within a string can be done using one of two methods, which basically work the same way.

The `match()` method is called on a string and is passed a regular expression:

```
var myString = "How much wood could a wood chuck chuck";
var myPattern = /.ood/;
var myResult = myString.match(myPattern); // ["wood"]
var patternLocation = myResult.index; // 9
var originalString = myResult.input // "How much wood could a wood chuck chuck"
```

And the `exec()` method is called on a `RegExp` object and is passed the string:

```
var myString = "How much wood could a wood chuck chuck";
var myPattern = /.huck/;
var myResult = myPattern.exec(myString); // ["chuck"]
var patternLocation = myResult.index; // 27
var originalString = myResult.input // "How much wood could a wood chuck chuck"
```

For both methods, only the first matched occurrence is returned. If no match is found it will return `null`.

You can also use the `search()` method, which accepts a regular expression as the only argument and returns the location of the first occurrence of the pattern:

```
var myString = "Assume";
var patternLocation = myString.search(/ume/); // 3
```

If no match is found, the method returns “-1”.

Compare Two Strings for Sort Order

You can compare two strings to see which one comes first alphabetically using `localeCompare`, with three possible return values:

```
var myString = "chicken";
var myStringTwo = "egg";
var whichCameFirst = myString.localeCompare(myStringTwo); // -1 (except Chrome, which returns -2)
whichCameFirst = myString.localeCompare("chicken"); // 0
whichCameFirst = myString.localeCompare("apple"); // 1 (Chrome returns 2)
```

As shown above, a negative number is returned if the string argument comes after the original string, a positive number is returned if the string argument comes before, and `0` is returned if the strings are equal.

Since a browser can return any negative number or any positive number for the before and after results (the spec doesn’t require anything specific), when checking the value, it’s best to use `if ( result < 0 )`, rather than `if ( result === -1 )`, the latter of which wouldn’t work in Chrome.

Further Reading

If you notice any errors, omissions, or incomplete info, please comment and I’ll update accordingly. Meanwhile, if you want more on string-related stuff in JavaScript, here are a few sources:

Photo credit: tin cans with string from Bigstock.