JavaScript String Type

Summary: in this tutorial, you will learn about the JavaScript `String` type and how to manipulate strings effectively.

Introduction to JavaScript String type

The `String` type is object wrapper of the string primitive type and can be created by using the `String` constructor as follows:

```
let str = new String('JavaScript String Type');
```

The `String` type has a property named `length` that specifies the number of characters in the string.

```
console.log(str.length); // 22
```

In this example, the value of the `length` property is `22` that also is the number of characters in the string `'JavaScript String Type'`.

To get the primitive string value, you use one of the following methods of the String object: `valueOf()`, `toString()`, and `toLocaleString()`.

```
console.log(str.valueOf());
console.log(str.toString());
console.log(str.toLocaleString());
```

To access an individual character in a string, you use square bracket notation `[]` with a numeric index. The index of the first character is zero as shown in this example:

```
console.log(str[0]); // J
```

The square bracket notation was introduced in ES5. Prior to ES5, you use the `charAt()` method, which is more verbose:

```
console.log(str.charAt(0)); // J
```

When you call a method on a primitive string variable or a literal string, it is converted to an instance of the `String` type. For example:

```
'literal string'.toUpperCase();
```

This feature is known as primitive wrapper types in JavaScript.

String manipulation

The `String` type provides many useful methods for manipulating strings effectively. We will examine each of them in the following section.

1) Concatenating strings

To concatenate two or more strings you use the `concat()` method as follows:

```
let firstName = 'John';
let fullName = firstName.concat(' ','Doe');
console.log(fullName); // "John Doe"
console.log(firstName); // "John"
```

The `concat()` method concatenates two or more strings and returns the result string. Note that the `concat()` method does not modify the original string.

Besides the `concat()` method, you can also use the addition operator (`+`) for concatenating strings. In practice, the addition operator is used more often than the `concat()` method.

```
let firstName = 'John';
let fullName = firstName + ' ' + 'Doe';
console.log(fullName); // "John Doe"
```

2) Extracting substrings

To extract a substring from a string, you use the `substr()` method:

```
substr(startIndex,[length]);
```

The `substr()` method accepts two arguments.

The first argument `startIndex` is the location at which the characters are being extracted, while the second argument `length` specifies the number of characters to extract.

```
let str = "JavaScript String";

console.log(str.substr(0, 10)); // "JavaScript"
console.log(str.substr(11,6)); // "String"
```

If you omit the `length` argument, the `substr()` method extracts the characters to the end of the string.

Sometimes, you want to extract a substring from a string using starting and ending indexes. In this case, you use the `substring()` method:

```
substring(startIndex,endIndex)
```

See the following example:

```
let str = "JavaScript String";
console.log(str.substring(4, 10)); // "Script"
```

3) Locating substrings

To locate a substring in a string, you use the `indexOf()` method:

```
string.indexOf(substring,[fromIndex]);
```

The `indexOf()` method accepts two arguments: a substring to locate and the `fromIndex` at which the method starts searching forward in the string.

The `indexOf()` returns the index of the first occurrence of the substring in the string. If the substring is not found, the `indexOf()` method returns `-1`.

```
let str = "This is a string";
console.log(str.indexOf("is")); // 2
```

The following example uses the `fromIndex` argument:

```
console.log(str.indexOf('is', 3)); //5
```

To find the location of the last occurrence of a substring in a string, you use the `lastIndexOf()` method.

```
string.lastIndexOf(substring,[fromIndex])
```

Unlike the `indexOf()` method, the `lastindexOf()` method searches backward from the `fromIndex` argument.

```
console.log(str.lastIndexOf('is')); // 5
```

The `lastIndexOf()` method also returns `-1` if the substring is not found in the string as shown in this example:

```
console.log(str.lastIndexOf('are')); // -1
```

4) Removing whitespace characters

To remove all leading and trailing whitespace characters of a string, you use the `trim()` method.

```
let rawString = ' Hi  ';
let strippedString = rawString.trim();
console.log(strippedString); // "Hi"
```

> Note that the `trim()` method creates a copy of the original string with whitespace characters stripped, it doesn't change the original string.

ES6 introduced two new methods for removing whitespace characters from a string:

- `trimStart()` returns a string with whitespace stripped from the beginning of a string.
- `trimEnd()` returns a string with the whitespace stripped from the end of a string.

5) Changing cases

To change cases of a string, you use `toLowerCase()` and `toUpperCase()` methods:

```
let greeting = 'Hello';
console.log(greeting.toLowerCase()); // 'hello'
console.log(greeting.toUpperCase()); // 'HELLO';
```

In some languages, the rules for converting a string to lowercase and uppercase are very specific.

Therefore, it is safer to use the `toLocaleLowerCase()` and `toLocaleUpperCase()` methods, especially when you don't know which language the code will deal with.

6) Comparing strings

To compare two strings, you use the `localeCompare()` method:

```
first.localeCompare(second);
```

The `localeCompare()` returns one of three values: -1, 0, and 1.

- If the first string comes before the second string alphabetically, the method returns -1.

- If the first string comes after the second string alphabetically, the method returns 1.
- If two strings are equal, the method returns 0.

For example:

```
console.log('A'.localeCompare('B')); // -1
console.log('B'.localeCompare('B')); // 0
console.log('C'.localeCompare('B')); // 1
```

7) Matching patterns

The `match()` method allows you to match a string with a specified regular expression and get an array of results.

The `match()` method returns null if it does not find any match. Otherwise, it returns an array containing the entire match and any parentheses-capture matched results.

If the global flag (`g`) is not set, the element zero of the array contains the entire match. Here is an example:

```
let expression = '1 + 2 = 3';
let matches = expression.match(/\d+/);
console.log(matches[0]); // "1"
```

Output:

```
1
```

In this example, the pattern matches any number in the `expression` string.

In case the global flag (`g`) is set, the elements of the result array contain all matches as follows:

```
let expression = '1 + 2 = 3';
let matches = expression.match(/\d+/g);

for (const match of matches) {
```

```
  console.log(match);
}
```

Output:

```
1
2
3
```

In this example, the `matches` array contains all the matches including `1`, `2`, and `3` in the `expression` string.

If you only need to find out if a string matches a regular expression, you use the `search()` method instead.

Similar to the `match()` method, the `search()` method accepts a regular expression and returns the position of the string where the first match is found. In case no match is found, it returns -1.

```
let str = "This is a test of search()";
let pos = str.search(/is/);
console.log(pos); // 2
```

8) Replacing substrings

To replace a substring in a string, you use the `replace()` method.

```
string.replace(regularExpression, replaceText)
```

The first argument of the `replace()` method could be a regular expression or a string. If it is a regular expression, the `replace()` method will find the matches and replace them with the second argument (`replaceText`). In case it is a string, the `replace()` method will perform an exact search and carry the replacement.

The `replace()` method returns a copy of the original string after making the replacements.

The following example illustrates how to use a regular expression to replace `the` with `a`:

```
let str = "the baby kicks the ball";

// replace "the" with "a"
let newStr = str.replace(/the/g, "a");

console.log(newStr); // "a baby kicks a ball"
console.log(str); // "the baby kicks the ball"
```

The following example shows how to replace `kicks` with `holds` by using the first argument as a literal string.

```
newStr = str.replace('kicks', 'holds');
console.log(newStr); // "the baby holds the ball"
```

In this tutorial, you have learned about the JavaScript String type and how to manipulate strings effectively.