(https://goo.gle/cds20-sessions) now!

Getting Literal With ES6 Template Strings

By Addy Osmani (/web/resources/contributors/addyosmani)
Eng Manager, Web Developer Relations

Strings in JavaScript have been historically limited, lacking the capabilities one might expect coming from languages like Python or Ruby. ES6 Template Strings (https://www.chromestatus.com/feature/4743002513735680) (available in Chrome 41+), fundamentally change that. They introduce a way to define strings with domain-specific languages (DSLs), bringing better:

- String interpolation
- Embedded expressions
- Multiline strings without hacks
- String formatting
- String tagging for safe HTML escaping, localization and more.

Rather than stuffing yet another feature into Strings as we know them today, Template Strings introduce a completely different way of solving these problems.

Syntax

Template Strings use back-ticks (``) rather than the single or double quotes we're used to with regular strings. A template string could thus be written as follows:

```
var greeting = `Yo World!`;
```

So far, Template Strings haven't given us anything more than normal strings do. Let's change that.

```
// Simple string substitution
var name = "Brendan";
console.log(`Yo, ${name}!`);

// => "Yo, Brendan!"
```

As all string substitutions in Template Strings are JavaScript expressions, we can substitute a lot more than variable names. For example, below we can use expression interpolation to embed for some readable inline math:

```
var a = 10;
var b = 10;
console.log(`JavaScript first appeared ${a+b} years ago. Wow!`);

//=> JavaScript first appeared 20 years ago. Wow!

console.log(`The number of JS MVC frameworks is ${2 * (a + b)} and not ${10 *
//=> The number of JS frameworks is 40 and not 200.
```

They are also very useful for functions inside expressions:

```
function fn() { return "I am a result. Rarr"; }
console.log(`foo ${fn()} bar`);
//=> foo I am a result. Rarr bar.
```

The `${}` works fine with any kind of expression, including member expressions and method calls:

```
var user = {name: 'Caitlin Potter'};
console.log(`Thanks for getting this into V8, ${user.name.toUpperCase()}.`);

// => "Thanks for getting this into V8, CAITLIN POTTER";
```

Multiline Strings

Multiline strings in JavaScript have required hacky workarounds for some time. Current solutions for them require that strings either exist on a single line or be split into multiline strings using a `\` (backslash) before each newline. For example:

```
var greeting = "Yo \
World";
```

Whilst this should work fine in most modern JavaScript engines, the behavior itself is still a bit of a hack. One can also use string concatenation to fake multiline support, but this equally leaves something to be desired:

```
var greeting = "Yo " +
"World";
```

Template Strings significantly simplify multiline strings. Simply include newlines where they are needed and BOOM. Here's an example:

Any whitespace inside of the backtick syntax will also be considered part of the string.

```
console.log(`string text line 1
string text line 2`);
```

cellent book, Understanding ES6 (https://leanpub.com/understandinges6/read#leanpub-auto-multiline-stri

Note how the (n + 1)th argument corresponds to the substitution that takes place between the nth and (n + 1)th entries in the string array. This can be useful for all sorts of things, but one of the most straightforward is automatic escaping of any interpolated variables.

For example, you could write a HTML-escaping function such that..

```
html`<p title="${title}">Hello ${you}!</p>`
```

returns a string with the appropriate variables substituted in, but with all HTML-unsafe characters replaced. Let's do that. Our HTML-escaping function will take two arguments: a username and a comment. Both may contain HTML unsafe characters (namely ', ", <, >, and &). For example, if the username is "Domenic Denicola" and the comment is "& is a fun tag", we should output:

```
<b>Domenic Denicola says:</b> "&amp;amp; is a fun tag"
```

```
    },
    replace = String.prototype.replace
  ;
  return (Object.freeze || Object)({
    escape: function escape(s) {
      return replace.call(s, reEscape, fnEscape);
    },
    unescape: function unescape(s) {
      return replace.call(s, reUnescape, fnUnescape);
    }
  });
}());

// Tagged template function
function html(pieces) {
    var result = pieces[0];
```

Check out our Template Strings sample (https://github.com/GoogleChrome/samples/tree/gh-pages/template-literals-es6) over on the Chrome samples repo if you'd like to try them out. You may also be interested in ES6 Equivalents in ES5 (https://github.com/addyosmani/es6-equivalents-in-es5#template-literals), which demonstrates how to achieve some of the sugaring Template Strings bring using ES5 today.