Getting Literal With ES6 Template Strings

By Addy Osmani
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 (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, localisation 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.

String Substitution

One of their first real benefits is string substitution. Substitution allows us to take any valid JavaScript expression (including say, the addition of variables) and inside a Template Literal, the result will be output as part of the same string.

Template Strings can contain placeholders for string substitution using the `${ }` syntax, as demonstrated below:

```
// 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. Crazy!`);

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

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

// And another example
var thing = 'drugs';
console.log(`Say no to ${thing}. Although if you're talking to ${thing} you may a

// => Say no to drugs. Although if you're talking to drugs you may already be on
```

If you require backticks inside of your string, it can be escaped using the backslash character \ as follows:

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

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 `\` (blackslash) before each newline. For example:

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

Whilst this should work fine in most modern JavaScript engines, the behaviour 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`);
```

Tagged Templates

So far, we've looked at using Template Strings for string substitution and for creating multiline strings. Another powerful feature they bring is tagged templates. Tagged Templates transform a Template String by placing a function name before the template string. For example:

```
fn`Hello ${you}! You're looking ${adjective} today!`
```

The semantics of a tagged template string are very different from those of a normal one. In essence, they are a special type of function call: the above "desugars" into

```
fn(["Hello ", "! You're looking ", " today!"], you, adjective);
```

Note: Nicholas Zakas goes into more detail on the break-down of these arguments in the Template Strings section of his excellent book, Understanding ES6.

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; is a fun tag"
```

Our tagged template solution could thus be written as follows:

```
// HTML Escape helper utility
var util = (function () {
  // Thanks to Andrea Giammarchi
  var
    reEscape = /[&<>'"]/g,
    reUnescape = /&(?:amp|#38|lt|#60|gt|#62|apos|#39|quot|#34);/g,
    oEscape = {
      '&': '&amp;',
      '<': '&lt;',
      '>': '&gt;',
      "'": '&#39;',
      '"': '&quot;'
    },
    oUnescape = {
      '&amp;': '&',
      '&#38;': '&',
      '&lt;': '<',
      '&#60;': '<',
      '&gt;': '>',
      '&#62;': '>',
      '&apos;': "'",
      '&#39;': "'",
      '&quot;': '"',
      '&#34;': '"'
    },
    fnEscape = function (m) {
      return oEscape[m];
    },
    fnUnescape = function (m) {
      return oUnescape[m];
    },
    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];
    var substitutions = [].slice.call(arguments, 1);
    for (var i = 0; i < substitutions.length; ++i) {
        result += util.escape(substitutions[i]) + pieces[i + 1];
    }

    return result;
}

var username = "Domenic Denicola";
var tag = "& is a fun tag";
console.log(html`<b>${username} says</b>: "${tag}"`);
//=> <b>Domenic Denicola says</b>: "&amp; is a fun tag"
```

Other possible uses include auto-escaping, formatting, localisation and in general, more complex substitutions:

```
// Contextual auto-escaping
qsa`.${className}`;
safehtml`<a href="${url}?q=${query}" onclick="alert('${message}')" style="color:

// Localization and formatting
l10n`Hello ${name}; you are visitor number ${visitor}:n! You have ${money}:c in y

// Embedded HTML/XML
```

```
jsx`<a href="${url}">${text}</a>` // becomes React.DOM.a({ href: url }, text)

// DSLs for code execution
var childProcess = sh`ps ax | grep ${pid}`;
```

Summary

Template Strings are in Chrome 41 beta+, IE Tech Preview, Firefox 35+ and io.js. Practically speaking if you would like to use them in production today, they're supported in major ES6 Transpilers, including Traceur and 6to5. Check out our Template Strings sample over on the Chrome samples repo if you'd like to try them out. You may also be interested in ES6 Equivalents in ES5, which demonstrates how to achieve some of the sugaring Template Strings bring using ES5 today.

Template Strings bring many important capabilities to JavaScript. These include better ways to do string & expression interpolation, multiline strings and the ability to create your own DSLs.

One of the most significant features they bring are tagged templates - a critical feature for authoring such DSLs. They receive the parts of a Template String as arguments and you can then decide how to use the strings and substitutions to determine the final output of your string.

Further Reading

- http://www.2ality.com/2015/01/template-strings-html.html
- https://leanpub.com/understandinges6/read/#leanpub-auto-tagged-templates
- http://jaysoo.ca/2014/03/20/i18n-with-es6-template-strings/
- http://odetocode.com/blogs/scott/archive/2014/09/30/features-of-es6-part-8-tagged-templates.aspx