State of ES6 support

As mentioned in Chapter 1, ES6 was finalized in 2015. Since then, browsers have been rolling out support with each incremental update. At the time of writing (March 2016), the current version of Chrome (Chrome 49) has 91% support. The current version of Chrome Canary (a nightly build of Chrome) has 94% support.

While this is great for Chrome, not all browsers currently offer the same level of support. Firefox is pretty close at 91%, and Edge 14 is 86%. But, what about Internet Explorer 11? Unfortunately, IE 11 is at 16%, and it will never get higher than that.

If you want to find out what the level of support is for each individual browser, check out the Kangax Compatibility Table [1]. It is a great reference for which browsers or environments support what.

Looking at the table, you will see that most browsers are improving their support of ES6 features. Unfortunately, some are not, and even worse, some will never change. We can not always bank on users only using something like Chrome or Firefox. As developers, we need to make sure that we are supporting the widest range of browsers we need to for our projects. In order to do this, we need to take our ES6 code and transpile it down into ES5 JavaScript that will be readable by the browser.

What is transpiling?

Transpiling is the act of taking one language and compiling it to another language. It is also referred to as source-to-source compiling. Let's take this example:

```
const numbers = [1,2,3,4,5];
const max = Math.max(...numbers);
```

In the above code, we use the spread operator to expand out the `numbers` array into `Math.max`. We will look at the spread operator in more detail later, but for now let's focus on the fact that this code will not run in any browser that do not support ES6. However, taking this code and running it through a transpiler, for example Babel [2], will compile it down to something that looks like this:

```
"use strict";

var numbers = [1, 2, 3, 4, 5];
var max = Math.max.apply(Math, numbers);
```

This is code that all browsers understand today. The idea of transpiling code is not new; many languages compile from one source to another. There are many languages [3] that use JavaScript as their compile target. One of the most popular is CoffeeScript [4]. This is a language that came about as a way to extend and add new functionality to JavaScript, so that you can write something like:

```
multi = (mul,numbers...) ->
   numbers.map(n -> n * multi)
```

That would equate to compiled JavaScript that looks like this:

```
var multi,
  slice = [].slice;

multi = function() {
  var mul, numbers;
  mul = arguments[0], numbers = 2 <= arguments.length ? slice.call(arguments, 1) : [];
  return numbers.map(n(function() {
    return n * multi;
  }));
};
```

CoffeScript brought some elegant new features to JavaScript, and ultimately some of those features were the inspiration for features like arrow functions in ES6!

We have talked about the current state of support and how we can use transpilers, or source-to-source compilers to work with ES6 in browsers that do not support it fully yet. In the next chapter, we will look at how to actually perform the transpile step!

Additional Resources

- 1:https://kangax.github.io/compat-table/es6/
- 2: https://babeljs.io/
- 3: https://github.com/jashkenas/coffeescript/wiki/list-of-languages-that-compile-to-js
- 4: http://coffeescript.org/