ES7 and ES8 Features

Module 2: ES7/ES2016 Features

Azat Mardan @azat_co

Let's dive deeper into the ES7/ES2016 proposals and features.

`Array.prototype.includes`

`Array.prototype.includes` is a replacement for `Array.prototype.indexOf(...)>-1` which developers used to check for presence of a value in an array.

In ES6:

```
let arr = ['react', 'angular', 'vue']

// WRONG
if (arr.indexOf('react')) { // 0 -> evaluates to false, definitely as we expected
  console.log('Can use React') // this line would never be executed
}

// Correct
if (arr.indexOf('react') !== -1) {
  console.log('Can use React')
}
```

Or a little bit hacky bitwise NOT operator ~ to make the code more compact since ~ (bitwise NOTing) against any number is equivalent to `-(a + 1)`:

```
let arr = ['react', 'angular', 'vue']

// Correct
if (~arr.indexOf('react')) {
  console.log('Can use React')
}
```

The ES7 code with `includes` would be:

```
let arr = ['react', 'angular', 'vue']

// Correct
if (arr.includes('react')) {
  console.log('Can use React')
}
```

Developers can also use `includes` with strings:

```
let str = 'React Quickly'

// Correct
if (str.toLowerCase().includes('react')) {  // true
  console.log('Found "react"')
}
```

Interestingly, many JavaScript libraries already had `includes` or similarly working `contains` (but TC39 decided not to use name contains because of MooTools).

Some other (older) includes:

- jQuery: `$.inArray`
- Underscore.js: `_.contains`
- Lodash: `_.includes` (and in version 3 and earlier, `_.contains`, just like in Underscore)
- CoffeeScript: `in` operator (example)
- Dart: `list.contains` (example)

In addition to being more eloquent and actually giving developers the boolean value instead of position of a match, `includes` also works with `NaN` (not a number).

Finally, `includes` has `fromIndex` second optional parameter. This is good for optimization, because it allows to look for a match starting at a certain position.

More examples:

```
console.log([1, 2, 3].includes(2)) // === true)
console.log([1, 2, 3].includes(4)) // === false)

console.log([1, 2, NaN].includes(NaN)) // === true)

console.log([1, 2, -0].includes(+0)) // === true)
console.log([1, 2, +0].includes(-0)) // === true)

console.log(['a', 'b', 'c'].includes('a')) // === true)
console.log(['a', 'b', 'c'].includes('a', 1)) // === false)
```

Note: `indexOf()` is still useful when you need an exact index (position) of the match, not just true/false value whether it exists or not.

All in all, `includes` brings simplicity to any developer who has to check if a value is in an array/list… which is like almost all of us. Rejoice!

Exponentiation Operator

This operator is mostly for developers doing some math and is useful in case of 3D, Virtual Reality, SVG or data visualization.

So in ES6/ES2015, you can use `Math.pow` or create a small recursive arrow function:

```
calculateExponent = (base, exponent) => base*((--exponent>1)?calculateExponent(base, exponent):base)
console.log(calculateExponent(7,12) === Math.pow(7,12)) // true
console.log(calculateExponent(2,7) === Math.pow(2,7)) // true
```

Now in ES7/ES2016, math-oriented developers can use shorter syntax:

```
let a = 7 ** 12
let b = 2 ** 7
console.log(a === Math.pow(7,12)) // true
console.log(b === Math.pow(2,7)) // true
```

Developers also can use operation assignment:

```
let a = 7
a **= 12
let b = 2
b **= 7
console.log(a === Math.pow(7,12)) // true
console.log(b === Math.pow(2,7)) // true
```

Many new ES features are borrowed from other languages (CoffeeScript - love it, Ruby, etc.) As you can guess, exponential operator exists in other languages:

- Python: `x ** y`
- CoffeeScript: `x ** y`
- F#: `x ** y`
- Ruby: `x ** y`
- Perl: `x ** y`
- Lua, Basic, MATLAB: `x ^ y`

Not having an exponential operator in JavaScript was never a problem for me personally. :)

I never wrote anything exponential in my 15 years of writing JavaScript besides interviews and coding tutorials like this one... Was the lack of an exponential operator a big miss for you?

That's all for ES7 features. :-)