A Drip of JavaScript

The Difference Between Boolean Objects and Boolean Primitives in JavaScript

Originally published in the A Drip of JavaScript newsletter.

One of the unintuitive things about JavaScript is the fact that there are constructors for each of the primitive value types (boolean, string, etc), but what they construct isn't actually the same thing as the primitive.

Take booleans, for example. In most code, the primitive values are used, like so:

```
var primitiveTrue = true;
var primitiveFalse = false;
```

There is also the `Boolean` function, which can be used as an ordinary function which returns a boolean primitive:

```
var functionTrue = Boolean(true);
var functionFalse = Boolean(false);
```

But the `Boolean` function can also be used as a constructor with the `new` keyword:

```
var constructorTrue = new Boolean(true);
var constructorFalse = new Boolean(false);
```

The tricky thing here is that when `Boolean` is used as a constructor, it doesn't return a primitive. Instead it returns an object.

```
// Outputs: true
console.log(primitiveTrue);

// Outputs: true
console.log(functionTrue);

// Outputs: Boolean {}
console.log(constructorTrue);
```

It turns out that using the `Boolean` constructor can be quite dangerous. Why? Well, JavaScript is pretty aggressive about type coercion. If you try adding a string and a number, the number will be coerced into a string.

```
// Outputs: "22"
console.log("2" + 2);
```

Likewise, if you try to use an object in a context that expects a boolean value, the object will be coerced to `true`.

```
// Outputs: "Objects coerce to true."
if ({}) { console.log("Objects coerce to true."); }
```

And since the `Boolean` object is an object, it will also coerce to `true`, even if its internal value is `false`.

```
// Outputs: "My false Boolean object is truthy!"
if (constructorFalse) {
    console.log("My false Boolean object is truthy!");
} else {
    console.log("My false Boolean object is falsy!");
}
```

If you actually need to get at the internal value of a `Boolean` object, then you'll need to use the `valueOf` method.

```
// Outputs: "The value of my false Boolean object is falsy!"
if (constructorFalse.valueOf()) {
    console.log("The value of my false Boolean object is truthy!");
} else {
    console.log("The value of my false Boolean object is falsy!");
}
```

But because of the quirks surrounding `Boolean` objects, you're probably best off avoiding them altogether. In fact, linting tools like JSHint and JSLint will flag the `Boolean` constructor as a potential error in your code.

In the event that you need to explicitly coerce another type of value into `true` or `false`, you're better off using `Boolean` as an ordinary function, or using the not operator twice.

```
// Two approaches to coercing 0 into false
var byFunction = Boolean(0);
var byNotNot = !!0;
```

The double not above is pretty simple, though it can be confusing if you haven't seen it before. Using a single not operator coerces the value into a boolean primitive and then reverses it. (To `true` in this case). The second use of the not operator reverses the value again, so that it is flipped back to the correct boolean representation of the original value.

You're reasonably likely to see both of these approaches in JavaScript code, though in my experience the double not is more common.

Thanks for reading!

Josh Clanton

© 2015. All rights reserved.