A Drip of JavaScript

The Problem with Testing for NaN in JavaScript

Originally published in the A Drip of JavaScript newsletter.

In JavaScript, the special value `NaN` (meaning "not a number") is used to represent the result of a mathematical calculation that cannot be represented as a meaningful number. For example:

```
var divisionByZod = 42 / "General Zod";

// Outputs: NaN
console.log(divisionByZod);
```

You are also likely run into `NaN` when using `parseInt` or `parseFloat` to extract a number from a string, as in this example:

```
var doomedParse = parseInt("Doomsday", 10);

// Outputs: NaN
console.log(doomedParse);
```

As a result, JavaScript developers often need to test a result variable to see whether it contains `NaN` or whether it is a meaningful value instead. There are several different ways of checking if a value is `NaN`, but unfortunately most of them are unreliable. Let's walk through some of them and see how they fail.

The logical thing to do is check whether the result is equal to `NaN`.

```
var divisionByZod = 42 / "General Zod";

var equalsNaN = (divisionByZod === NaN);

// Outputs: false
console.log(equalsNaN);
```

What's going on here? In JavaScript, `NaN` has the distinction of being the only value that is not equal to itself. That means we can't find out whether a value is `NaN` by checking equality to `NaN` because the answer will always be no.

What else could we try? How about the built in `isNaN` function?

```
var divisionByZod = 42 / "General Zod";

var valueIsNaN = isNaN(divisionByZod);

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

Awesome! This one seems to work. And in fact it does work under some circumstances. However, despite the name, the purpose of the `isNaN` function isn't to check whether a value is `NaN`. Instead, the purpose is to check whether a value cannot be coerced to a number. Because of this, it may return false positives. Consider the following:

```
var isJorElNaN = isNaN("Jor El");

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

Unless you are certain that you will be dealing only with numbers or `NaN`, the `isNaN` function isn't quite fit for this purpose.

However, it turns out that ECMAScript 6 creates a new function for the specific purpose of checking whether a value is `NaN`. Confusingly, this function is also called `isNaN`, though it is attached to the `Number` object rather than directly to the global object. It can be used like this:

```
var divisionByZod = 42 / "General Zod";

var valueIsNaN = Number.isNaN(divisionByZod);

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

Excellent! We've got a method that really works. Unfortunately, `Number.isNaN` is only available in newer versions of FireFox and Chrome, and isn't yet supported in IE or other major browsers. This may be a viable approach in the future, but for right now it can only be used in very narrow circumstances.

Maybe we can check `NaN`'s type instead?

```
// Outputs: "Number"
console.log(typeof NaN);
```

Yes, you're reading that correctly. In JavaScript, the value `NaN` is of type `number`. That's not going to be very helpful.

So what can we do to get something that works correctly, but is also available cross-browser? Remember how I mentioned that `NaN` is the only value in JavaScript that is not equal to itself? We can take advantage of that fact.

```
var divisionByZod = 42 / "General Zod";
```

```
// This can only be true if the value is NaN
var valueIsNaN = (divisionByZod !== divisionByZod);

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

Currently the best way to check whether a value is `NaN` is to check whether it is not equal to itself. If so, then you know that it is `NaN`.

Of course, using this technique isn't especially readable, so it is common to wrap it up into a utility method which communicates the intent more clearly. Both Underscore and Lo-Dash provide implementations.

I hope this has helped you get a better grasp on the quirks of `NaN`.

Thanks for reading!

Josh Clanton
