A Drip of JavaScript

Using ECMAScript 6 Maps

Originally published in the A Drip of JavaScript newsletter.

In last week's drip I discussed using `Object.create(null)` in order to simplify creating key-value maps in JavaScript. But even with that approach, objects aren't a substitute for a full-fledged map data type. For example:

```
var heroesNemesis = Object.create(null);

var greenLantern = { name: "Green Lantern" };
var sinestro = { name: "Sinestro" };

heroesNemesis[greenLantern] = sinestro;

// Outputs: { name: "Sinestro" }
console.log(heroesNemesis[greenLantern]);

var wonderWoman = { name: "Wonder Woman" };

// Outputs: { name: "Sinestro" }
console.log(heroesNemesis[wonderWoman]);
```

Why is our object reporting Wonder Woman's nemesis as Sinestro? In JavaScript, an object's keys can only be strings. If you try to supply something else, the JavaScript engine will try to coerce it into a string with `toString`.

And the default `toString` implementation for an object returns `"[object Object]"` regardless of what properties the object possesses. So unless we decide to start monkeying around with `Object.prototype` (generally a bad idea) our "map" object can't distinguish between different objects as keys.

Fortunately, ECMAScript 6 (the next version of JavaScript) includes a new `Map` data type which allows us to use any valid JavaScript value as a key.

```
var heroesNemesis = new Map();

var greenLantern = { name: "Green Lantern" };
var sinestro = { name: "Sinestro" };

heroesNemesis.set(greenLantern, sinestro);

// Outputs: { name: "Sinestro" }
console.log(heroesNemesis.get(greenLantern));

var wonderWoman = { name: "Wonder Woman" };

// Outputs: undefined
console.log(heroesNemesis.get(wonderWoman));
```

As you can see, `Map` gives us the result we'd expect. To use `Map` our main interfaces are `set` which accepts a key and associated value to store, and `get` which accepts a key and returns the associated value.

When checking to see whether a given key matches, `Map` uses the "same value" algorithm. So even if two objects look identical, only the original can be used to retrieve it's associated value. For example:

```
var clonedLantern = { name: "Green Lantern" };

// Outputs: undefined
console.log(heroesNemesis.get(clonedLantern));
```

And you can determine how many elements have been stored in a `Map` by checking its `size` property.

```
// Outputs: 1
console.log(heroesNemesis.size);
```

This basic set of functionality for `Map` is supported in the latest versions of Chrome and FireFox, as well as IE11. In future drips we'll look into more advanced features of `Map`.

Thanks for reading!

Joshua Clanton
