A Drip of JavaScript

Constructors in JavaScript

Originally published in the A Drip of JavaScript newsletter.

Despite the fact that they are very powerful, constructors are one of the most underused features of JavaScript. (Probably because they have a very influential detractor.) But if you want to really know JavaScript, you'll need to learn how they work.

What is a constructor? It's an ordinary function that is used with the `new` operator to produce a specialized type of object.

```javascript
// `Color` is a constructor
var red = new Color(255, 0, 0);
```

In this example, red is a new "Color" object. But how does that work?

```javascript
function Color(r, g, b) {
    this.r = r;
    this.g = g;
    this.b = b;
}

var red = new Color(255, 0, 0);
```

As you can see, the Color constructor is merely taking the arguments given to it and attaching them to the `this` object. That's because when the constructor is invoked

by `new`, the constructor's `this` is set to the object that `new` will return.

Which means that the code above is roughly equivalent to:

```
var red = {
    r: 255,
    g: 0,
    b: 0
};
```

So why would we use a constructor? There are a couple of important reasons.

First, using a constructor means that all of these objects will be created with the same basic structure, and we are less likely to accidentally make a mistake than if we were creating a whole bunch of generic objects by hand. (Especially if we make the constructor throw errors on invalid input.)

Second, using a constructor means that the object is explicitly marked as an instance of `Color`.

```
var red = new Color(255, 0, 0);

var blue = { r: 255, g: 0, b: 0 };

// Outputs: true
console.log(red instanceof Color);

// Outputs: false
console.log(blue instanceof Color);
```

That gives us a means to ensure that we are receiving the correct type of data to process.

Third, using a constructor means that we can easily assign specialized methods to the constructor's prototype, and they will instantly be available to all objects created

by the constructor.

```
function Color(r, g, b) {
    this.r = r;
    this.g = g;
    this.b = b;
}

Color.prototype.getAverage = function () {
    var total = this.r + this.g + this.b;
    var avg = total / 3;
    return parseInt(avg, 10);
};

var red = new Color(255, 0, 0);
var white = new Color(255, 255, 255);

// Outputs: 85
console.log(red.getAverage());

// Outputs: 255
console.log(white.getAverage());
```

It is important to **always** use the new keyword when invoking the constructor. Otherwise the constructor may clobber the `this` which was accidentally passed to it. In most cases that is the global object (`window` in the browser or `global` in Node.)

Because of this danger, it is customary to capitalize a constructor's name so that others know to invoke it with `new`.

An upcoming article will deal with how to further mitigate the danger inherent in working with constructors.

Josh Clanton
