A Drip of JavaScript

Basic Inheritance with Object.create

Originally published in the [A Drip of JavaScript newsletter](#).

A few issues back we looked at [how to implement basic inheritance with constructors](#). In this issue, we'll look at how to do the same with the newer `Object.create`.

When using constructors for inheritance, we attach properties to the constructor's `prototype` property like so:

Here's a little refresher:

```
function SuperHuman (name, superPower) {
    this.name = name;
    this.superPower = superPower;
}

SuperHuman.prototype.usePower = function () {
    console.log(this.superPower + "!");
};

var banshee = new SuperHuman("Silver Banshee", "sonic wail");

// Outputs: "sonic wail!"
banshee.usePower();
```

The `SuperHuman` constructor contains our initialization logic, while `SuperHuman.prototype` contains the methods that are shared across all `SuperHuman` instances.

If we were to implement the same basic logic using `Object.create`, it would look a bit different:

```
var superHuman = {
    usePower: function () {
        console.log(this.superPower + "!");
    }
};

var banshee = Object.create(superHuman, {
    name: { value: "Silver Banshee" },
    superPower: { value: "sonic wail" }
});

// Outputs: "sonic wail!"
banshee.usePower();
```

In this case we first define the prototype object `superHuman`, and then we use `Object.create` to make a new object which inherits from `superHuman`. That second argument might look a little strange to you, but it's just a simple property descriptor object, like we use with `Object.defineProperty` to fine-tune an object's properties.

Now, what if we want to create a new type which inherits from `superHuman` while adding its own functionality? What would that look like?

```
var superHero = Object.create(superHuman, {
    allegiance: { value: "Good" },
    saveTheDay: {
        value: function () {
            console.log(this.name + " saved the day!");
        }
    }
});

var marvel = Object.create(superHero, {
    name: { value: "Captain Marvel" },
    superPower: { value: "magic" }
});

// Outputs: "Captain Marvel saved the day!"
marvel.saveTheDay();
```

So far so good. But does Captain Marvel have access to the `superHuman` prototype methods?

```
// Outputs: "magic!"
marvel.usePower();
```

Yes, she does!

Using `Object.create` makes setting up inheritance chains simple because any object can be used as a prototype. However, inheritance managed by `Object.create` can't be detected by `instanceof`. Instead you'll need to use the `isPrototypeOf` method, like so:

```
// Outputs: true
console.log(superHero.isPrototypeOf(marvel));

// Outputs: true
console.log(superHuman.isPrototypeOf(marvel));
```

Because both `superHero` and `superHuman` are part of `marvel`'s prototype chain, their `isPrototypeOf` calls each return true.

As with other JavaScript features we've reviewed, `Object.create` is a feature of ECMAScript 5 and is not available in older browsers like IE8.

That's our brief introduction to using `Object.create`. Thanks for reading!

Joshua Clanton
