A Drip of JavaScript

Creating Objects Without Prototypes

Originally published in the A Drip of JavaScript newsletter.

Last drip we talked about inheritance using prototypes and Object.create. But one point that sometimes surprises new JavaScript developers is that even ordinary "empty" objects are already part of an inheritance chain. Consider the following:

```
var empty = {};

// Outputs: Function Object()
console.log(empty.constructor);
```

Every time you create a new object via an object literal (the `{}`), behind the scenes JavaScript invokes the `Object` constructor to create the object, just as if you'd used `new Object()`. This is what allows your new object to inherit properties from `Object.prototype`.

But sometimes it would be convenient to create an object that doesn't inherit from a prototype at all. For instance, if you'd like to use an object as a hash/map of arbitrary keys to values.

```
var dictionary = {
    destructor: "A destructive element"
};

function getDefinition(word) {
```

```
    return dictionary[word];
}

// Outputs: "A destructive element"
console.log(getDefinition("destructor"));

// Outputs: function Object()
console.log(getDefinition("constructor"));
```

As you can see, our `getDefinition` function works perfectly for words that we have explicitly defined. However, it also returns inherited properties like `constructor`.

One way of solving this would be to introduce a `hasOwnProperty` check to make sure that the properties aren't inherited. But another way is just to ensure that our `dictionary` never inherits any properties to begin with.

Fortunately, `Object.create` makes that rather easy.

```
var dictionary = Object.create(null, {
    destructor: { value: "A destructive element" }
});

function getDefinition(word) {
    return dictionary[word];
}

// Outputs: "A destructive element"
console.log(getDefinition("destructor"));

// Outputs: undefined
console.log(getDefinition("constructor"));
```

As you can see, in this case `getDefinition` returned our intended result.

The trick here is in our first argument to `Object.create`. Normally this is where we would pass in the object that we want to serve as the prototype. But if we pass in

`null` instead, our new object doesn't inherit from a prototype at all.

Of course, `Object.create` isn't available in IE8 and older, so you should only use this technique in modern browsers.

I hope you found this week's drip informative!

Joshua Clanton
