Creating Bound Functions with Function#bind

Originally published in the A Drip of JavaScript newsletter.

Continuing from some of our previous discussions of functions as first-class values, it's time to tackle `Function`'s `bind` method. As you might guess, the purpose of `bind` is to "bind" a function. But what does that mean, exactly?

```
var hero = {
    name: "Batman",
    signal: function () {
        console.log(this.name + " has been signaled.");
    }
};

// Outputs: "Batman has been signaled."
hero.signal();
```

Consider if we wanted to let another object signal Batman. We could do something like this:

```
var commissioner = {
    name: "Jim Gordon",
    signalBatman: function() {
        hero.signal();
    }
};
```

```
// Outputs: "Batman has been signaled."
commissioner.signalBatman();
```

But what if the `hero` variable gets redefined?

```
hero = {
    name: "Superman",
    signal: function () {
        console.log(this.name + " has been signaled.");
    }
};

// Outputs: "Superman has been signaled."
commissioner.signalBatman();
```

Commissioner Gordon is signaling Superman? That can't be right. Let's try using `bind` instead.

```
var hero = {
    name: "Batman",
    signal: function () {
        console.log(this.name + " has been signaled.");
    }
};

var commissioner = {
    name: "Jim Gordon",
    signalBatman: hero.signal.bind(hero)
};

hero = {
    name: "Superman",
    signal: function () {
        console.log(this.name + " has been signaled.");
    }
};
```

```
// Outputs: "Batman has been signaled."
commissioner.signalBatman();
```

As you can see, the `bind` method allows us to create a new function which is permanently bound to a given value of `this`. You can't even override its `this` value using `call` or `apply`. This can be quite handy when you need to pass around a function that needs a certain `this` value in order to function correctly.

For instance, consider good old `console.log`:

```
// Outputs: "logging"
console.log("console.logging");

var justLog = console.log;

// TypeError: Illegal invocation
justLog("just logging");
```

It turns out that `log` just won't work without `console`. But is there a way we can just pass around the function instead of the entire console object? With `bind` there is.

```
// Outputs: "logging"
console.log("console.logging");

var justLog = console.log.bind(console);

// Outputs: "just logging"
justLog("just logging");
```

Unfortunately, `bind` is only supported in Internet Explore 9 or higher. If you need this functionality in older browsers, you can use Underscore, Lo-Dash, or the ES5 shim library.

That's a brief introduction to creating bound functions with `bind`. Next time we'll look at using `bind` for partial application.

Thanks for reading!

Joshua Clanton
