A Drip of JavaScript

Creating Chainable Interfaces in JavaScript

Originally published in the A Drip of JavaScript newsletter.

When first learning jQuery, one of the things that most strikes developers is the ease of using its chainable interface to just keep stringing commands together.

```
$("#myDiv")
    .addClass("myClass")
    .css("color", "red")
    .append("some text");
```

This approach is very powerful. But until you understand how it works, it can seem rather mysterious and complicated. Fortunately, it's very straightforward to implement, so that's what we'll be looking at.

That "jQuery style" chainability is also known as a fluent interface. The fundamental thing that makes a fluent interface possible is for each method to return an object so that you can then call methods upon it.

That's a little abstract, so here is a concrete example:

```
function Book(name, author) {
    this.name = name;
    this.author = author;
}
```

```
Book.prototype.setName = function (name) {
    this.name = name;
    return this;
}

Book.prototype.setAuthor = function (author) {
    this.author = author;
    return this;
}

lotr = new Book("Lord of the Rings", "Tolkien");

// Outputs: {
//     name: "Lord of the Rings",
//     author: "Tolkien"
// }
console.log(lotr);

// Whoops! Details were slightly wrong.
// Let's fix that.
lotr.setAuthor("JRR Tolkien") // Returns `lotr`
    .setName("The Lord of the Rings"); // Returns `lotr`

// Outputs: {
//     name: "The Lord of the Rings",
//     author: "JRR Tolkien"
// }
console.log(lotr);
```

The trick here is in our prototype methods. We created `setName` and `setAuthor` which can be called on any `Book` object. But then we made sure that when the methods had done their work, they returned the object they were originally called on.

Since that object was a `Book`, we could immediately call any `Book` method upon it. And that's all that chaining requires.

Thanks for reading!

Joshua Clanton
