A Drip of JavaScript

An Introduction to Writing Your Own JavaScript Compatibility Shims

Originally published in the A Drip of JavaScript newsletter.

We've talked in several past issues about the need to use compatibility shims to get newer features of JavaScript ported back into older browsers. Today we're going to talk about how those shims work and how you can write your own.

We are going to create a shim for the previously discussed Array#map, which is not available in IE8 and below. To recap, the `map` method returns a new array which is created by running a transformation function over each element of the original array, like so:

```
var evens = [2, 4, 6, 8, 10];

var odds = evens.map(function(val) { return val - 1; });

// Outputs: [1, 3, 5, 7, 9]
console.log(odds);
```

Perhaps the most important part of writing a compatibility shim is using feature detection to ensure that we don't accidentally overwrite the `map` method in a modern browser.

```
// Only use the shim if map isn't defined
if (!Array.prototype.map) {
    // Create and attach shim here
}
```

Once we know that we are in a `map` less browser, we can actually create the shim itself.

```
// Only use the shim if map isn't defined
if (!Array.prototype.map) {
    // Map will accept a single function as an argument
    Array.prototype.map = function(fn) {
        var i,
            newVal,
            mappedArr = [];

        for (i = 0; i < this.length; i++) {
            // Run the callback function over each element
            // to get the updated value
            newVal = fn(this[i]);

            // Push the new value onto the new array
            mappedArr.push(newVal);
        }

        // Return our new array
        return mappedArr;
    };
}
```

Now if you load that code up in IE8 and below, you should have a functional (if incomplete) `map` method where it was completely missing before.

Of course, it's worth pointing out that this particular implementation of `map` isn't production-worthy. For one thing, it doesn't take into account the fact that `map` will pass in the index and array as arguments to the callback function. It also fails to

account for the fact that `map` itself will accept a second argument to set its `this` value. For details, see our previous discussion of map, as well as the detailed documentation on MDN.

And that leads me to the second most important part of writing a compatibility shim. You need to make sure that you get the functionality right. A lot of the time it is really simple to code for the most common use cases, but much more difficult to ensure that you've covered all of the uncommon ones. If you'd like to see how much of a difference that makes, take a look at ES5-shim's implementation of map. It is significantly more complex.

Because of this complexity, it is generally best to rely on an established library with lots of unit tests to do the heavy lifting for you. However, if your goal is to learn about a new feature of JavaScript, building your own implementation will teach you more of the ins and outs than anything else.

Thanks for reading!

Josh Clanton
