A Drip of JavaScript

Transforming Arrays with Array#map

Originally published in the A Drip of JavaScript newsletter.

One of the most common tasks that developers perform in any language is taking an array of values and transforming those values. Up until recently, doing that in JavaScript took a fair bit of boilerplate code. For instance, here is some code for darkening RGB colors:

```
var colors = [
    {r: 255, g: 255, b: 255 }, // White
    {r: 128, g: 128, b: 128 }, // Gray
    {r: 0,   g: 0,   b: 0   }  // Black
];

var newColors = [];

for (var i = 0; i < colors.length; i++) {
    transformed = {
        r: Math.round( colors[i].r / 2 ),
        g: Math.round( colors[i].g / 2 ),
        b: Math.round( colors[i].b / 2 )
    };

    newColors.push(transformed);
}

// Outputs:
```

```
// [
//    {r: 128, g: 128, b: 128 },
//    {r: 64,  g: 64,  b: 64  },
//    {r: 0,   g: 0,   b: 0   }
// ];
console.log(newColors);
```

As you can see, there's quite a bit going on in that code that isn't really about what we want to accomplish, but is keeping track of trivial things like the current index and moving the values into the new array. What if we didn't have to do all of that?

Fortunately in ECMAScript 5 (the latest version of JavaScript), we don't. Here is the same example rewritten to take advantage of the `map` method:

```
var newColors = colors.map(function(val) {
    return {
        r: Math.round( val.r / 2 ),
        g: Math.round( val.g / 2 ),
        b: Math.round( val.b / 2 )
    };
});
```

Much nicer isn't it? Invoking `map` returns a new array created by running a transformation function over each element of the original array.

Now the only thing you need to keep track of is the logic of the transformation itself.

Of course, `map` isn't limited to simple transformations like this. Your function can also make use of two additional parameters, the current index and the array itself. Consider the following example:

```
var starter = [1, 5, 5];

function multiplyByNext (val, index, arr) {
    var next = index + 1;
    // If at the end of array
    // use the first element
    if (next === arr.length) {
        next = 0;
    }

    return val * arr[next];
}

var transformed = starter.map(multiplyByNext);

// Outputs: [5, 25, 5]
console.log(transformed);
```

As you can see, the additional parameters make it easy to create transformation functions which use the array element's neighbors. This can be useful in implementing something like Conway's Game of Life.

Browser support for `map` is pretty good, but not universal. It isn't supported in IE 8 and below. You have a few options for dealing with this.

1. Don't use `map`.
2. Use something like es5-shim to make older IE's support `map`.
3. Use the `_.map` method in Underscore or Lodash for an equivalent utility function.

One of the most powerful techniques for avoiding programming bugs is to reduce the number of things that you are keeping track of manually. Array's `map` method is one more tool to help you do exactly that.

Thanks for reading! If you enjoyed this article, consider sending it to a friend.

Josh Clanton
