A Drip of JavaScript

Testing Array Contents with Array#every

Originally published in the A Drip of JavaScript newsletter.

We've talked before about how common it is to test the contents of an array to figure out what to do next, and how Array's `some` method will let us test whether some elements meet our criteria. But sometimes we need to make sure that **every** element in an array meets certain criteria.

Here is an example of the pattern at work:

```
var heroes = [
    { name: "Superman",     universe: "DC"     },
    { name: "Batman",       universe: "DC"     },
    { name: "Spider-Man",   universe: "Marvel" },
    { name: "Wonder Woman", universe: "DC"     }
];

// Default to true
var areAllDC = true;

for (var i = 0; i < heroes.length && areAllDC; i++) {
    if (heroes[i].universe !== "DC") {
        areAllDC = false;
    }
}
```

```
// Outputs: false
console.log(areAllDC);
```

We iterate over the array with a for loop, but with an extra check on `areAllDC` to halt iteration as soon as we've determined that there is a non-matching element.

So, that's the "classical" looping approach. Fortunately, JavaScript also gives us a very nice built-in function to do this work for us. And we can do so without having to constantly write loops and keep track of extraneous variables and code.

The `every` function is available on all arrays, and we can use it like this:

```
function isDC(element) {
    return (element.universe === "DC");
}

// Outputs: false
console.log(heroes.every(isDC));

var villains = [
    { name: "Brainiac", universe: "DC" },
    { name: "Sinestro", universe: "DC" },
    { name: "Darkseid", universe: "DC" },
    { name: "Joker",    universe: "DC" }
];

// Outputs: true
console.log(villains.every(isDC));
```

The callback will continue to be executed on each element until it returns a falsy value (`false` , `undefined` , etc.) or it reaches the end of the array. If it reaches the end without returning a falsy value, then `every` will return `true` .

The callback function also has access to two other parameters: the current index, and the array as a whole. You can use them to evaluate the current element in the context of the entire array.

For example:

```
function isSameUniverse(el, index, arr) {
    // The first element doesn't have a predecessor,
    // so don't evaluate it.
    if (index === 0) {
        return true;
    } else {
        // Do the universe properties of
        // current and previous elements match?
        return (el.universe === arr[index - 1].universe);
    }
}

// Outputs: false
console.log(heroes.every(isSameUniverse));

// Outputs: true
console.log(villains.every(isSameUniverse));
```

Because `every` is part of the ES5 specification, it isn't available in IE8 and below. But if you want to use it in older browsers, you can use Underscore, Lo-Dash, or an ES5 shim to make it available.

Thanks for reading!

Josh Clanton
