A Drip of JavaScript

Using JavaScript's Array Methods on Strings

Originally published in the A Drip of JavaScript newsletter.

We've talked before about using `apply` and `call` to let methods from one object operate on a different object. But this time we'll take a detailed look at how that works with strings and array methods specifically.

Because in many ways strings behave as if they were arrays of characters, many of JavaScript's standard array methods can operate on strings as well. Many, however, is not all. For example:

```
var ontologist = "Anselm";
[].push.call(ontologist, " of Canterbury");

// Outputs: "Anselm"
console.log(ontologist);
```

As you can see, the `ontologist` string was not modified, even though the whole point of `push` is to modify the array. This is because in JavaScript, strings are immutable.

Because strings are immutable (cannot be modified) any method which attempts to change the string will fail. That immediately rules out methods like `push`, `pop`, `shift`, and `splice`.

However, other methods which treat the string as read-only are fair game. Consider these:

```
var ontologist = "Anselm";

var hasSomeA = [].some.call(ontologist, function(val) {
    return val === "A";
});

// Outputs: true
console.log(hasSomeA);

var everyCharIsE = [].every.call(ontologist, function(val) {
    return val === "E";
});

// Outputs: false
console.log(everyCharIsE);

var beforeM = [].filter.call(ontologist, function(val) {
    return val < "m";
});

// Outputs: ["A", "e", "l"]
console.log(beforeM);
```

It's great that all of these methods can operate on strings, but you may have noticed that `filter` returns an array rather than a string. If you think about it, this makes sense. Using `call` or `apply` doesn't change the function's logic, just what value it operates on.

Any array method which returns an array will continue to do so even if called on a string value. If you need your final result to be a string again, then you will need to convert it back to a string with `join`, like so:

```
var beforeM = [].filter.call(ontologist, function(val) {
    return val < "m";
}).join("");

// Outputs: "Ael"
console.log(beforeM);
```

Above, I mentioned that many array methods will fail to operate on strings because they are immutable. But there is a fairly simple way around this. We can first convert the string into an array and then convert it back again.

For example:

```
var reversed;

reversed = [].reverse.call(ontologist);

// Outputs: "Anselm"
console.log(reversed);

reversed = [].slice.call(ontologist).reverse().join("");

// Outputs: "mlesnA"
console.log(reversed);
```

Our first attempt at reversing the string fails because the `reverse` method attempts to mutate the value it is called upon. But in our second attempt we first use the `slice` method to convert the string into an array, and only then attempt to reverse it, using join to convert it into a new string after the mutatation. This basic approach means that you can use the entire range of array methods to manipulate strings.

As you can see, using array methods on strings can be a little quirky at times, but can also be very powerful. Hopefully this brief introduction has helped you see some places where it can help you in your day to day work.

Thanks for reading!

Josh Clanton

© 2015. All rights reserved.