A Drip of JavaScript

Finding Array Elements with Array#indexOf

Originally published in the A Drip of JavaScript newsletter.

If you are a JavaScript developer you are probably quite familiar with the `indexOf` method available on strings. The basic functionality of `Array`'s `indexOf` is very similar. But we will also explore the complexities introduced due to arrays being composed of disparate types of values.

Suppose that you have an array which represents an ordered rank of elements and you need to find out just where a given element falls in the ranking. Without `indexOf` we would need to do something like this.

```
var power = [
    "Superman",
    "Wonder Woman",
    "Batman"
];

for (var i = 0; i < power.length && power[i] !== "Wonder Woman"; i++) {
    // No internal logic is necessary.
}

var rank = i + 1;

// Outputs: "Wonder Woman's rank is 2"
console.log("Wonder Woman's rank is " + rank);
```

While the `for` loop isn't too complicated, with `indexOf` we no longer have to keep track of the state of our counters and array elements. Instead we can simply do this:

```
var power = [
    "Superman",
    "Wonder Woman",
    "Batman"
];

var index = power.indexOf("Wonder Woman");

var rank = index + 1;

// Outputs: "Wonder Woman's rank is 2"
console.log("Wonder Woman's rank is " + rank);
```

The `indexOf` method will return the index of the first element which matches your search. Or if it does not find a match it will return `-1`.

It also gives you the ability to start your search at an index other than `0`. This can help speed things up if you already know you can rule out some portion at the beginning of the array. You can use it like this:

```
var power = [
    "Superman",
    "Wonder Woman",
    "Batman"
];

var index = power.indexOf("Wonder Woman", 1);

var rank = index + 1;

// Outputs: "Wonder Woman's rank is 2"
console.log("Wonder Woman's rank is " + rank);
```

While merely skipping one element is a trivial gain, if you are dealing with thousands of elements you may be able to obtain a significant boost in performance.

There is one final aspect to consider: equality. The `indexOf` method uses strict equality (`===`) to determine if an element matches your search. This works great for primitives like strings and numbers. But you may have difficulty using `indexOf` to search for a given object or array. For example:

```
var basilicas = [
    {
        name: "St. Peter's",
        city: "Rome"
    },
    {
        name: "St. John Lateran",
        city: "Rome"
    }
];

var lateranIndex = basilicas.indexOf({
    name: "St. John Lateran",
    city: "Rome"
});

// Outputs: -1
console.log(lateranIndex);
```

What's going on here? Why isn't `indexOf` finding the index of St. John Lateran? That's a consequence of how object equality works in JavaScript. Two objects can have exactly the same properties and values, but unless they are the exact same object instance they are not equal. If you need to match on an object or array with `indexOf` , you'll need to retain a reference to the original object instance, like so:

```
var peters = {
    name: "St. Peter's",
    city: "Rome"
```

```
};

var lateran = {
    name: "St. John Lateran",
    city: "Rome"
};

var basilicas = [peters, lateran];

var lateranIndex = basilicas.indexOf(lateran);

// Outputs: 1
console.log(lateranIndex);
```

If you need a more robust solution that allows you to match on complex objects without retaining a reference, look into combining the findIndex and isEqual methods from Lo-Dash.

It is worth pointing out that `Array`'s `indexOf` method is part of the ECMAScript 5 specification. As such it is not available in IE8 and below. If you need this functionality in older browsers, you can use ES5 Shim to backport it, or the equivalent `indexOf` method in the Underscore or Lo-Dash libraries.

Thanks for reading!

Joshua Clanton
