A Drip of JavaScript

Filtering Arrays with Array#filter

Originally published in the A Drip of JavaScript newsletter.

Working with arrays is a daily task for most developers. And one of the most common tasks is taking an array and filtering it down to a subset of the elements it contains.

A straightforward way of doing the filtering manually might look like this:

```
var sidekicks = [
    { name: "Robin",     hero: "Batman"   },
    { name: "Supergirl", hero: "Superman" },
    { name: "Oracle",    hero: "Batman"   },
    { name: "Krypto",    hero: "Superman" }
];

var batKicks = [];

for (var i = 0; i < sidekicks.length ; i++) {
    if (sidekicks[i].hero === "Batman") {
        batKicks.push(sidekicks[i]);
    }
}

// Outputs: [
//    { name: "Robin",  hero: "Batman"   },
//    { name: "Oracle", hero: "Batman"   }
```

```
// ]
console.log(batKicks);
```

Fortunately, in JavaScript, arrays have the handy `filter` method which we can use to do the filtering for us instead of manually looping through the array ourselves.

```
var sidekicks = [
    { name: "Robin",     hero: "Batman"   },
    { name: "Supergirl", hero: "Superman" },
    { name: "Oracle",    hero: "Batman"   },
    { name: "Krypto",    hero: "Superman" }
];

var batKicks = sidekicks.filter(function (el) {
    return (el.hero === "Batman");
});

// Outputs: [
//    { name: "Robin",  hero: "Batman"   },
//    { name: "Oracle", hero: "Batman"   }
// ]
console.log(batKicks);

var superKicks = sidekicks.filter(function (el) {
    return (el.hero === "Superman");
});

// Outputs: [
//    { name: "Supergirl", hero: "Superman" },
//    { name: "Krypto",    hero: "Superman" }
// ]
console.log(superKicks);
```

The filter method accepts a callback function. In that callback function we examine each element of the array indidually, and return `true` if we want the element to pass the filter.

In addition to the individual array element, the callback also has access to the index of the current element and the full array, like so:

```
var filtered = sidekicks.filter(function (el, index, arr) {
    // Filtering here
});
```

This allows you to create more complex filters that depend on the element's relationship with other elements, or the array as a whole.

Because the filter method returns an array, it can be chained together with other array methods like `sort` and `map` (see issues #3 and #6.) For example:

```
var sortedBatKickNames = sidekicks.filter(function (el) {
    return (el.hero === "Batman");
}).map(function(el) {
    return el.name;
}).sort();

// Outputs: ["Oracle", "Robin"];
console.log(sortedBatKickNames);
```

Because the `filter` method is part of the ECMAScript 5 specification, it isn't available in older browsers like IE8. If you need to use it in older browsers, then you will need to use a compatibility shim or the equivalent filter method in a library like Underscore or Lo-Dash.

Hopefully this has given you some idea of how you can use `filter` in your everyday work.

Thanks for reading!

Josh Clanton
