written by Thomas McDevitt on 20 April 2016 in design, development process

mutating javascript arrays

There are plenty of functions that can extract, modify, add, or do almost anything with arrays in JavaScript. An important consideration is whether or not the original array gets mutated when a certain action is run on it. Regardless of whether or not it's fine to have the original array change down the line, it's important to be aware of what effect various actions will have on it.

what's the difference?

In a nutshell...

- An accessor method leaves the original array(s) unchanged and can return either a modified array or another detail.
- On the other hand, a mutator method makes its changes to the original array and often returns something else.

Let's check out some quick examples.

```
var arr1 = ['apple', 'banana']; var arr2 = ['mango', 'grape', 'durian']; var arrJoined =
arr1.concat(arr2);
```

`Array.prototype.concat` is an *accessor* method. Here, `arr1` and `arr2` are not changed during the concatenation, the return value for which is a new array with all five elements that gets assigned to the `arrJoined` variable. It's possible to run this without the assignment but the new array will simply be lost.

Also note that it's possible to simply assign the result of `concat()` to one of the previous variables; just keep in mind that it's not mutating the original array — it's creating a new array and then discarding its reference to the old one.

```
var arr = [5, 10, 15]; arr.push(20); // arr is now equal to [5, 10, 15, 20]
```

`Array.prototype.push` is a *mutator* method and the original array now has an extra element. Caching the result of `push()` is unnecessary, but would return the new length of the array (in this case, it's 4) should that be useful.

let's be creative...

By understanding the effect and return value of the different prototypic methods, it's possible to combine them to some interesting effects. Here are some examples.

Turning mutators into accessors: `Array.prototype.slice` and `Array.prototype.concat` are accessor methods. The former returns a portion of an array starting from a provided index whereas the latter will join an array to another. Conveniently, both methods can run without an argument to simply make a copy of the original array. This has some interesting uses.

- We can make a new copy of an array without having to loop through all the properties of the old one and push them to a new one.
- We can turn some mutators directly into accessors (once again, be weary of the return value when deciding how to use it this way).

For instance, imagine we have an array of transactions ordered by date but we want to find the three largest transactions without losing the time-relative data.

```
var transactions = [50, 21, 33, 178, 12, 19]; var threeHighest = transactions .slice()
// accessor - create a copy .sort(function(t1, t2) { return t2 > t1; }) // mutator -
sort the transactions on the copy .slice(0, 3); // accessor - get the first three results
console.log(threeHighest); // [178, 50, 33]
```

Using this method, despite requiring a mutator to arrange the transactions in order, we've still retained the original data whilst storing the new information in another variable!

Turning accessors into mutators: `Array.prototype.push` is a mutator that adds an element to an array. Should we wish to add another array to the original one *whilst also* mutating the original array, we could loop through the second array and keep running `push()`. Or...

```
var arr = ['sheep', 'giraffe', 'llama']; var arr2 = ['dog', 'cat', 'fish']; arr.push.apply(arr,
arr2); // returns 6 - length of new array console.log(arr); // array with all 6 elements
```

This way, not only have we added the new elements by making creative use of Function.prototype.apply but other references to the original array are also now up to date. Give it a try!

any other tips?

There are obviously heaps of other innovative ways to use the various array methods, the above being just a few examples. If you've come up with any particularly nifty ways to handle arrays then do let us know about them. Stay creative! :)

> Why did the programmer quit his job? ...because he didn't get arrays.

If you enjoyed the read, drop us a comment below or share the article, follow us on Twitter or subscribe to our #MetaBeers newsletter. Before you go, grab a PDF of the article, and let us know if it's time we worked together.