JANUARY 21, 2020 / #JAVASCRIPT

The Differences Between forEach() and map() that Every Developer Should Know

Ibrahima Ndaw

JavaScript has some handy methods which help us iterate through our arrays. The two most commonly used for iteration are `Array.prototype.map()` and `Array.prototype.forEach()`.

But I think that they remain a little bit unclear, especially for a beginner. Because they both do an iteration and output something. So, what is the difference?

In this article, we'll look at the following:

- Definitions
- The returning value
- Ability to chain other methods
- Mutability

Definitions

The `map` method receives a function as a parameter. Then it applies it on each element and returns an entirely new array populated with the results of calling the provided function.

This means that it returns a new array that contains an image of each element of the array. It will always return the same number of items.

```
const myAwesomeArray = [5, 4, 3, 2, 1]

myAwesomeArray.map(x => x * x)

// >>>>>>>>>>>>>>>>> Output: [25, 16, 9, 4, 1]
```

Like `map` , the `forEach()` method receives a function as an argument and executes it once for each array element. However, instead of returning a new array like `map` , it returns `undefined` .

```
const myAwesomeArray = [
  { id: 1, name: "john" },
  { id: 2, name: "Ali" },
  { id: 3, name: "Mass" },
]

myAwesomeArray.forEach(element => console.log(element.name))
// >>>>>>>>> Output : john
//                    Ali
//                    Mass
```

The first difference between `map()` and `forEach()` is the returning value. The `forEach()` method returns `undefined` and `map()` returns a new array with the transformed elements. Even if they do the same job, the returning value remains different.

```
const myAwesomeArray = [1, 2, 3, 4, 5]
myAwesomeArray.forEach(x => x * x)
//>>>>>>>>>>>>>return value: undefined

myAwesomeArray.map(x => x * x)
//>>>>>>>>>>>>>return value: [1, 4, 9, 16, 25]
```

2. Ability to chain other methods

The second difference between these array methods is the fact that `map()` is chainable. This means that you can attach `reduce()`, `sort()`, `filter()` and so on after performing a `map()` method on an array.

That's something you can't do with `forEach()` because, as you might guess, it returns `undefined`.

```
const myAwesomeArray = [1, 2, 3, 4, 5]
myAwesomeArray.forEach(x => x * x).reduce((total, value) => total + va
//>>>>>>>>>>>>> Uncaught TypeError: Cannot read property 'reduce' of u
myAwesomeArray.map(x => x * x).reduce((total, value) => total + value)
//>>>>>>>>>>>>>return value: 55
```

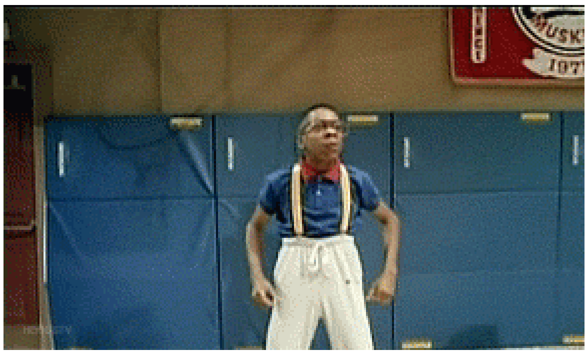
A mutable object is an object whose state can be modified after it is created. So, what about `forEach` and `map` regarding mutability?

Well, according to the MDN documentation:

`forEach()` does not mutate the array on which it is called. (However, `callback` may do so).

`map()` does not mutate the array on which it is called (although `callback`, if invoked, may do so).

JavaScript is weird.



Here, we see a very similar definition, and we all know that they both receive a `callback` as an argument. So, which one relies on immutability?

Well, in my opinion, this definition is not clear though. And to know which does not mutate the original array, we first have to check how these two methods work.

The `map()` method returns an entirely new array with transformed elements and the same amount of data. In the case of `forEach()`, Therefore, we see clearly that `map()` relies on immutability and `forEach()` is a mutator method.

4. Performance Speed

Regarding performance speed, they are a little bit different. But, does it matter? Well, it depends on various things like your computer, the amount of data you're dealing with, and so on.

You can check it out on your own with this example below or with jsPerf to see which is faster.

```
const myAwesomeArray = [1, 2, 3, 4, 5]

const startForEach = performance.now()
myAwesomeArray.forEach(x => (x + x) * 10000000000)
const endForEach = performance.now()
console.log(`Speed [forEach]: ${endForEach - startForEach} miliseconds

const startMap = performance.now()
myAwesomeArray.map(x => (x + x) * 10000000000)
const endMap = performance.now()
console.log(`Speed [map]: ${endMap - startMap} miliseconds`)
```

Final Thoughts

As always, the choice between `map()` and `forEach()` will depend on your use case. If you plan to change, alternate, or use the data, you should pick `map()`, because it returns a new array with the transformed data.

Hopefully, this post clears up the differences between these two methods. If there are more differences, please share them in the comment section, otherwise thanks for reading it.

Read more of my articles on my blog

Photo by Franck V. on Unsplash

Ibrahima Ndaw

JavaScript enthusiast, Full-stack developer & blogger

States Federal Tax Identification Number: 82-0779546)

Our mission: to help people learn to code for free. We accomplish this by creating thousands of videos, articles, and interactive coding lessons - all freely available to the public. We also have thousands of freeCodeCamp study groups around the world.

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

You can make a tax-deductible donation here.

Trending Guides

Python Parse JSON	HTML Video
HTML Cheat Sheet	CSS Overflow
ROW_NUMBER in SQL	What is REST?
HTML Center Image	VGA No Signal
JavaScript Replace	CSS Text Align
HTML HR Tag Example	HTML Label Tag
JavaScript UpperCase	CSS Button Style
Ordered List in HTML	For Loop in Java
Screenshot on Windows	Linux cp Command
Sort an Array in Java	Link CSS to HTML
Cast a Function in SQL	Declare an Array in Java
Java Logical Operators	Get Current Time in Python
Chmod Command in Linux	Change Div Background Color
Python Get Last Element	Get Variable Type in Python
Python Add to Dictionary	Logical Operators in Python

Our Nonprofit

About Alumni Network Open Source Shop Support Sponsors Academic Honesty