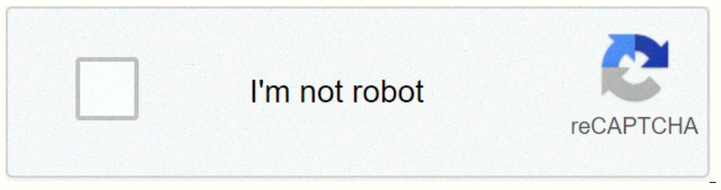

Continue

Map vs foreach performance

Photo by Nick Fewings on UnsplashSome of the most loved functions in JavaScript might be map and forEach. They both started to exist since ECMAScript 5, ores5 in short.In this post, I am going to talk about the main difference between each and show you some examples of their usages.Basically, looping over an object in JavaScript counts on whether or not the object is an iterable. Arrays are iterable by default.map and forEach are included in the Array.prototype, so we don’t need to think about iterable. If you want to study further, I recommend you check out what an iterable object is in JavaScript!map and forEach are helper methods in array to loop over an array easily. We used to loop over an array, like below, without any helper functions.The for loop has been with us since the very beginning of the JavaScript era. It takes 3 expressions: the initial value, condition, and final expression.This is a classic way of looping an array. Since ECMAScript 5, new functions have appeared to make us happier.mapmap does exactly the same thing as what the for loop does, except that map creates a new array with the result of calling a provided function on every element in the calling array.It takes two parameters: a callback function that will be invoked later when map or forEach is called, and the context variable called thisArg that a callback function will use when it’s invoked.The callback function can be used as below.The result of map is not equal to the original array.You can also pass the object to the map as thisArg.The object obj became the thisArg for map. But the arrow callback function can’t get obj as its thisArg.This is because arrow functions work differently from normal functions. Visit this article to see what’s different between arrow functions and normal functions.forEachforEach is another looping function for an array but there’s a difference between map and forEach in use. There are two parameters that map and forEach can take — a callback function and thisArg which they use as their this.Then, what’s different?map returns a new array of its original array. forEach, however, does not. But both of them ensure the immutability of the original object.~~ Edit ~~forEach doesn’t ensure the immutability of an array if you change values inside an array. This method only ensures immutability when you don’t touch any values inside.The example above is from Kenny Martin Rguez. Thank you! 🙇When to Use map() and forEach()?Since the main difference between them is whether or not there is a return value, you would want to use map to make a new array and use forEach just to map over the array.This is a simple example.In React, for example, map is used very commonly to make elements because map creates and returns a new array after manipulating data of the original array.On the other hand, forEach is useful when you want to do something with the data without creating a new array. By the way, the example could be refactored using filter.To summarize, I recommend you use map when creating a new array that you will use, and use forEach when you don’t need to make a new array, but rather, there’s something you want to do with the data.Some posts mentioned map is faster than forEach. So, I was curious if it’s for real. I found this comparison result:Left result jsPerf and right result jsPerfThe code looks very similar but the results are the opposite. Some tests said forEach is faster and some said map is faster.Maybe you are telling yourself that map/forEach is faster than the other, and you might be right. I’m not sure, honestly. I think readability is much more important than the speed between map and forEach when it comes to modern web development.But one thing’s for sure — both of them are slower than the built-in feature of JavaScript, for loop.map and forEach are handy functions for looping over an iterable object and might beautify your code and give you more readability.But one really important keynote to keep in mind is to understand what each does and when to use each.map would be good when you want to make a new array that does not affect the original array, and forEach would be nicer when you just want to map over an array. Want to learn more Advanced JavaScript? Check out: JavaScript — Understanding the Weird PartsIf you’ve worked with JavaScript for a little while, you’ve probably come across two seemingly similar Array methods: Array.prototype.map() and Array.prototype.forEach().So, what’s the difference?Map & ForEach DefinedLet’s first take a look at the definitions on MDN:forEach() — executes a provided function once for each array element.map() — creates a new array with the results of calling a provided function on every element in the calling array.What exactly does this mean?Well, the forEach() method doesn’t actually return anything (undefined). It simply calls a provided function on each element in your array. This callback is allowed to mutate the calling array.Meanwhile, the map() method will also call a provided function on every element in the array. The difference is that map() utilizes return values and actually returns a new Array of the same size.Code ExamplesConsider the below array. If we wanted to double each element in the array, we could use either map or forEach.ForEach:Note that you would never return from a forEach function as the return values are simply discarded:Result:Map:Result:Speed ConsiderationsjsPerf is a great website for testing the speed of different JavasScript methods and functions.Here are the results of my forEach() vs map() test:As you can see, on my machine forEach() was more than 70% slower than map(). Your browser is probably different. You can check out the full test results here:Functional ConsiderationsIt’s important to also understand that using map() may be preferable if you favor functional programming.This is because forEach() affects and changes our original Array, whereas map() returns an entirely new Array — thus leaving the original array unchanged.Which is better?That depends on what you’re trying to accomplish.forEach() may be preferable when you’re not trying to change the data in your array, but instead want to just do something with it — like saving it to a database or logging it out:And map() might be preferable when changing or altering data. Not only is it faster but it returns a new Array. This means we can do cool things like chaining on other methods (map(), filter(), reduce(), etc.)What we’re doing above is first mapping over arr and multiplying every element in the array times 2. After this, we filter through the array and only save the elements that are greater than 5. This leaves us with a final arr2 of [6,8,10].If you want to learn more about chaining map, reduce, and filter, check out my article: JavaScript — Learn to Chain Map, Filter, and Reduce.Key TakeawaysJust about anything you can do with forEach() you can do with map(), and vise versa.map() allocates memory and stores return values. forEach() throws away return values and always returns undefined.forEach() will allow a callback function to mutate the current array. map() will instead return a new array.Thanks for reading, and hopefully this was helpful! If you’re ready to finally learn Web Development, check out The Ultimate Guide to Learning Full Stack Web Development in 6 months.I publish 4 articles on web development each week. Please consider entering your email here if you’d like to be added to my once-weekly email list, or follow me on Twitter.If this post was helpful, please click the clap 👏button below a few times to show your support! ⬇⬇ CP Apr 23, 2020 ・Updated on May 5, 2020 ・1 min read When iterating through an array, when should we use for, forEach, or map? Here's a quick summary of each. for() Performance: Built-in function. Faster than the other iteration methods due to less overhead (e.g. no callbacks to initialize). Use it for large arrays. Short-circuiting: can use the break statement to stop the iteration. Beware of ES6 syntax--e.g. for (const key in object) for iterating objects, or for (const element of array) for iterating arrays--essentially this syntax turns the for() loop into a forEach loop--meaning, no short-circuiting available anymore. (Note: one is in and the other uses of) forEach() Takes a callback function: arr.forEach(callback) It has 3 params: value, index, and the original array. See example: > [1, 2].forEach((value, index, arr) => console.log(value, index, arr)) 1 0 [1, 2] 2 1 [1, 2] undefined The undefined is the returned value of the forEach() call. forEach ALWAYS iterate through the entire array. Not good for large arrays. It does NOT wait for asynchronous tasks to complete. map() Returns an array of the return values of the callback function. Everything else behaves the same as a forEach() call. The keys of a map() is ordered, it follows the order of insertion for objects. (While Object.keys() does not guarantee the order.) It does NOT wait for asynchronous tasks to complete. (Worth repeating). Because it returns an array, you can use map() with asynchronous calls like this: await Promise.all(array.map(...)) Latest run results: Run details: (Test run date: 11 hours ago) User agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:86.0) Gecko/20100101 Firefox/86.0 Browser/OS: Firefox 86 on Windows View result in a separate tab Test name Executions per second foreach 170375.2 Ops/sec for 624012.6 Ops/sec map 60410.1 Ops/sec I almost never use for loops in JavaScript and many other languages anymore. Instead, when iterating over collections I tend to use the map operator when it’s available. This applies to arrays which have a .map method as well as Observables which have a map operator and things such as lodash’s map function which allows for iteration over a broader set of collections including unordered collections such as Objects.Note: in this article I’ll occaionally use map as a shorthand for many iteration operators. My article is primarily geared towards JavaScript, but I think the concepts do apply more universally.I actually can’t think of the last time I found myself needing to use a for loop for anything. I can usually replace it with another operator. I also typically teach newer developers about iteration operators such as map and eschew for and other such looping constructs sometimes entirely even though they were / are a bread-and-butter of programming. for loops are one of the first things that many developers learn.So why use map?One of the most difficult questions I get when teaching is why we use constructs such as .map over for loops. I don’t think that I have a great answer. Even searching online for “for loop vs. map” and the like, I can’t find anything really concrete. It is interesting to me that all the top results I find for this search are related to JavaScript even though this could certainly apply to other programming languages that have a map and similar constructs as well. It also seems to be pretty contentious to some people.I decided to write this article because I’m not fully satisfied with what I found out there talking about constructs such as map vs. for loops. I’m hoping this provides more clarity to some people with the question — especially developers who are more familiar with for than with map.I won’t try to say that one is strictly better than the other. I personally prefer using collection iteration operators such as map over language looping constructs, and I will continue to teach this paradigm. I’ll discuss the reasons for my preference, and try to list the concrete benefits of using map. I think that most people I work with — once they understand the iteration syntax — prefer it over loops. However, this is more preference and working in a standard than a clear-cut victory for one or the other. I don’t think such a thing will ever exist. I also have to admit that I have a personal preference for the map syntax which does have some influence over my choice.For the sake of simplicity, I’ll stick to working with Arrays for the rest of this article even though the concepts I talk about can be applied to many other collection data structures more generally.What are Arrays?I always keep my wife in mind when I write these articles. She is not a programmer, but I think she has an ancillary understanding of programming concepts what with being forced to live with me. Thus, I’ll start with the basics even though most developers and even many laypeople will already understand this.An array is an ordered list — in fact this construct is called a List in some programming languages. Arrays contain data in a set order. For example, [1, 2, 3] is a list of three numbers. You could also have [2, 3, 1], or ['andrew', 'james', 'crites'] or any other list of data that you could think of. You can think of it as a column (or possibly a row) in a spreadsheet.We need to manipulate the data stored in arrays all the time. A very common way to do this is to take one array and generate a new array with new data. For example, if we take the array of names above, maybe we want to capitalize all of them and then display them back to a user, properly capitalized. This involves taking each item or element of the array and running a capitalization function against it. It will become ['Andrew', 'James', 'Crites'].We could write a simple program that runs the capitalization function on each element of the array…Arrays are typically 0-indexed. That means that the first element (also called the zeroth element) is accessed by 0 rather than 1. We can also access array elements by an index stored in a variable, likeArrays are indexed by the whole numbers (counting numbers starting with 0), in order.How the for loop worksArrays can be of arbitrary length. All of the array examples above have three items in there. They have a length of 3. However, arrays can be of essentially any size from 0 (empty) to millions of items or more. For example, if we have an array of a person’s names such as ['andrew', 'james', 'crites'] it’s certainly plausible that they would have two, four, or some other number of names like ['santiago', 'juan', 'mendez', 'menendez'] or ['john', 'smith'].Our program from above won’t work right for these other two arrays since it won’t capitalize 'menendez', and some undefined behavior might happen when we try to access the missing element, names[2] of the John Smith array.We also have to repeat ourselves quite a lot in our program by writing capitalize over and over again. This will be difficult to update later if our requirements change and we want to use a different function instead — maybe uppercase or something. It’s also more error prone since we might mistype as capitaliz once we get bored of typing it over-and-over again.All of these issues can be solved by constructs that move through an array regardless of its length. The for loop is one of the most basic of these and is often taught to developers very early on. What the for loop does is run the code inside of the loop over and over until some condition is met (or more accurately only while some condition is met).This is usually written as for (x = 0; x < names.length; x++).Now our program will work on an array of any length, and we have the added benefit of only having our capitalize function called in one spot. If we needed to update our logic to do something different or more complicated, we would only have to update it one time.If you were to read the suggested program in colloquial English, I might suggest:For x starting at 0…As long as x is less than the length of the names array…Capitalize the xth element of the names array…Then add one to xThe for loop above does ad-hoc iteration. We are moving through each element of the array, but we have to define the condition ourselves. We could have put x = 0 which would make our program run forever — that would be a big problem. We could also change how we update x, perhaps with x = x + 2.I posit that this is not true iteration because we can write for loops in a way that it does not move through every element of the array in order.True Iteration with mapSome languages allow you to perform true iteration using the for keyword, such as for (element: array). This will make element each element of the array in order inside of the for loop block. I won’t discuss this iteration construct in this article — it’s more like forEach I mention later. Languages may also call map something else or have some other way of writing it.Enter the map construct. This function takes another function, runs that function on every element of the array, and returns a new array:We could also simply write this as names.map(capitalize).It may look weird to people who are unfamiliar, but that function we give to map is run on each element. We can unravel it and it will be identical to our first implementation:In the same way that the code inside of our for loop is called as long as the condition is true, the code inside of map() is called one time for each element in the array.This does the same thing as our for loop, but the big difference is that the conditions for iteration are handled for us. We’re guaranteed to get every element of the array in the right order. We don’t have to manage our own index variable or condition for that variable.Map also provides the benefit of immutability. When we run map, it creates a new array and leaves our original array alone (assuming we don’t update the original array inside our function). This has benefits of not updating our data unexpectedly and allowing us to use our original array later on for some other purpose if we need it:We can write our for loop in a way that does not update our original array too, but this requires proper management of a new array:As with our earlier for loop this has the potential to be error-prone in some way. We also have to initialize capitalizedNames in the proper place and to an empty array rather than simply initializing it as the result that we want.Benefits of map over for loopsThis is a list of the benefits of using map as I see them:True iteration — you know that your code is going to run on each element of the array in the right order.Immutability — if written properly, your original array will be unaffected. This has potential benefits in cases where the original array is still needed elsewhere. for loops can of course also be written so as not to update the original array, but it requires more code and updating our new array as part of our loop operation. map keeps this cleaner since you only have to work in one scope to still maintain immutability:Scope isolation and Reusability — often the function inside map only needs to work within its own scope, i.e. against the elements inside the array themselves. It doesn’t require any variables outside of its scope. A for loop may require this (such as declaring capitalizeNames outside the loop and also using it in the loop). This reduces the chances of making a mistake by updating something inappropriately in the wrong scope. This also makes it easier to extract the operation we use in map for clarity / isolation / reuse.Cleaner code — I think less code is strictly better as long as the original meaning isn’t lost. When doing identical things, map can almost always be written with less code than for. It can be cleanly written on one line sometimes whereas for requires at least two — generally three with braces included. Finally, the array element you are working with for map automatically gets a name. You would have to assign this to a variable when using for if you wanted it. name is clearer than names[x]. Scope isolation and a reduction in the amount of variables you need alongside reduced size all make code objectively cleaner. It’s arguable whether the meaning of map vs. for is lost on some people, but I think the next point helps:Specificity — this applies to iteration operators generally rather than just map. Each has a specific purpose. map calls a function for each element of the array and returns a new array. reduce calls a function for each element of the array and keeps the results in an accumulator whose ultimate value is returned. filter takes an array and returns a new array whose values have met some condition against the original array. Each operator has a specific purpose that is easier to reason about. for loops can be used more generally, and you have to inspect the code to determine how it’s being used in a specific case.Composability — it is much easier to compose iteration operations together. For example, you could do map(map(map)) (not that you should need to), or more realistically reduce(filter(map)) . If you wanted to do something similar for for loops, you would have to manage a separate array and iteration process for all three operations and their results.I think the composability benefit merits an example. Let’s say we have an array of strings that we want to convert to numbers, and then and we want to get the sum of all the even numbers in the array. We can simply do:If we run this on ['1', '2', '3', '4', '5'] we will get 6, i.e. 2 + 4.With for loops, we have to do this:This is more code, it’s more difficult to reason about, there is more nesting, and it’s more error-prone — particularly in the spot where we have to manage the index of evenNumbers since we’re creating a new array of a different size. That’s something filter could do for us. We’re also managing quite a few variables here, and it could be pretty easy to get tripped up. We also use arbitrary names for indices such as x. Certainly we could write index here, but with the iteration operators we don’t even need the index at all.It’s also easier to add more composition using map and other iteration methods in the middle if we need to.Benefits (?) of the for loopThe for loop has potential benefits, but I don’t think they are as concrete as those offered by map or other iteration operators.Performance — maybe… I ran some unscientific performance tests of for vs. map on 10,000,000-item arrays in Chrome and Safari. map performed strictly worse in Chrome, but better in Safari. However, overall everything completed in 1–3 seconds which I think is still relatively fast for such a massive array. It’s plausible that you will work with such large arrays if you are doing things like processing video contents in-browser. Under these specific circumstances, if you need the benefit of around half a second of performance per-10,000,000 elements in Chrome you might be better off using a for loop for now. However, on other platforms / environments or other circumstances, map might still be faster, and in fact it may be faster in the future. Consider the other benefits of map and do your own testing before choosing for loops purely for performance reasons. Also keep in mind that while a for loop may run faster it might use more memory too.Familiarity — newer developers might not even know what map is, and it can be difficult to grasp at first. I think this is still a non-benefit, though, since it’s important for developers to always learn new concepts and see the benefits and drawbacks of each. I would also recommend that map be taught alongside and perhaps even instead of for nowadays. Perhaps there is some benefit to developers who use for being able to jump into a new code base without unfamiliar constructs, but I don’t think map is that difficult to understand… especially not compared to a lot of the other constructs or libraries you would expect a new developer to have to learn when transitioning platforms and languages. The benefits of familiarity in programming are not to be understated, but I think the solution here is to start teaching map earlier instead. In fact, some programming languages such as Rust don’t even have traditional for loops.Ease of understanding… but probably not — I think that the for loop is so ubiquitous and taught so early on that programmers have a grasp on it for most of their programming lives. This creates a possible illusion that it’s easier to understand. In fact, I think that map specifically may be easier to understand than a for loop in general. for loops for a specific purpose might be easier to understand, but since they can work so generally their meaning is not always clear. I think that map’s specific purpose makes it clearer. I do wonder if we taught map first and then talked about for loops to someone who was brand new to programming whether they would find one or the other easier. Regardless, I don’t think that map is a difficult concept to grasp — not more than for anyway. It’s just that for gets hammered home so much. Also, once you understand map it’s easier to understand all of the other iteration operators available. All-in-all, this one is subjective.Note that the ability to break out of for loops or continue through them isn’t a benefit since it’s obviated by other iteration operators. continue is trivially replaced by return if it’s needed. Operations where you would typically need break such as finding if some or all elements of an array match some condition or finding a specific array element all have iteration replacement methods in JavaScript and other languages, e.g. some, every, find, and perhaps fill… and many more. You might consider also using a tool belt like lodash if you need to do more interesting things or work with other types of collections rather than rewrite logic someone else has already done better.Non-benefits of mapI often hear the fact that map is “functional programming” as a reason why it’s beneficial. First of all, I can’t say I strictly know what functional programming means. I think it has a meaning similar “object-oriented programming” only its buzzword trap is worse. It seems like if you’re using map or the like in your code you’re doing functional programming in the same way that if you use objects in your code you’re doing object-oriented programming.Rather than falling back on the functional buzzword as a crutch, I’d rather look at the concrete benefits of map and other iteration functions I extol above.Another comment I see and agree with is that map is easier to read / looks better / is cooler than using a for loop. This is entirely subjective, though. I think that some people hate map vehemently and find it more difficult to read too. However, I’m sure you can find someone out there who says the same thing about for loops. I do in cases where they end up with some deep nesting. Maybe after accepting the other benefits of map it starts to look cooler. This ties in with the “cleaner code” benefit I listed above, but code cleanliness is more objective than simply liking how syntax looks or thinking it’s easier to read.What about forEach?In JavaScript there is an array operator forEach which does the same thing as map except it doesn’t return a new array. I think that forEach is almost like a transition between using for loops and other operators. For example, you could write the even number sum code using forEach instead, and it might look like this:This is sort of a mixture between the iteration operators and for loops. You get some of the benefits including true iteration and some scope isolation and reduction in code size, but you still don’t get any composability or specificity.I think that forEach is not needed (except maybe to help with debugging), and, in cases where you find yourself using it, you probably want map, reduce, some, every, find, or filter. In cases where you purely want to update an array element in-place, the benefits of map over forEach weaken… pretty much the only benefits you would get in that case are immutability and composability which you might not need. However, I would probably choose map in that case out of personal preference / dislike of forEach alongside the fact that forEach has no benefit in that case either.The performance of forEach vs. map is even less clear than of for vs. map, so I can’t say that performance is a benefit for either.In ConclusionI hope that this helps answer the question about “why use map over a for loop” to some degree. I think that iteration operators such as map have concrete benefits that the for loop simply doesn’t have. I think that the benefits of the for loop relegate it to specific use-cases related to performance. I ask any infuriated for loop fanatic to think: if you had learned about and were using map consistently first, would you still like for loops better once you learned about them? Another way to phrase the question is: can you think of concrete benefits to the for loop besides familiarity? Imagine that any new developer you work with knows map as well … or maybe even instead of for. Can you list other benefits?Performance is often cited as a key reason to choose for, but I think some of the arguments for this are spurious. A browser is not the best platform for working with very large data sets. Sometimes you might have to, but I think that the collections you’ll be working with in a browser are typically relatively small. I’ve seen the argument that for will save milli or even micro-seconds. You can add up all of the total amount of time saved for a huge total performance benefit for users. However, the issue here is that there is user down-time to consider. Users do a lot during idle time. The 1 millisecond you gain from using for pales in comparison to the 30 seconds a user will take to read the paragraph — and you can’t get any of that time back. I agree that in some cases there may be obvious performance benefits, but in cases where the benefits are very small you have to weigh the other benefits of map that I listed. Is 1 millisecond per 30 seconds worth sacrificing code size, composability, specificity, reusability, and simpler immutability and iteration? The 1/1000 seconds vs. 30 seconds may be very dramatic, but I don’t think it’s totally unrealistic either. There are other factors that may impact performance a lot more that are worth a look before you drill down to map vs. for.There is also the argument for consistency among code-bases… if you’re using for in one very tight performance spot you might as well use for everywhere so that you’re consistent. However, I think this goes against the specificity argument. You might need for in one spot for performance reasons, but the implication of using it everywhere is that it serves the same purpose everywhere. In fact, if we think about it I think we’ll all agree that for and map are not pure alternatives to each other… the only thing they share is that they can iterate over collections. Your decision about whether to use map or for to iterate should require more thought than consistency of using the same keyword.Finally, I’m not trying to discourage anyone from using for. Ultimately if you own a code-base and that’s what you’re more comfortable with, even if it’s for reasons you can’t enumerate, I say go for it. I only ask that you take the time to learn and understand map and see its potential benefits.