HACKERNOON

3 JavaScript Performance Mistakes You Should Stop Doing

ishay
October 6th 2018
dishay

What if I told you everything you knew was a lie, what will happen if you learn some of the key features our beloved ECMAScript have published over the recent years, are actually dangerous performance traps, sugar coated in a slick looking one line callback functional code?
This story starts a few years ago, back in the naive days of ES5...

I still remember this day vividly, ES5 was released, and great new array functions were introduced to our dear JavaScript. Among them were forEach, reduce, map, filter—they made us feel the language is growing, getting more functional, writing code became more fun and smooth, and the result was easier to read and understand.

About the same time, a new environment grew—Node.js, it gave us the ability to have a smooth transition from front-end to back-end while truly redefining full stack development.

Nowadays, Node.js, using the latest ECMAScript over V8, is trying to be considered as part of the major league server-side development languages, and as such, it needs to prove worthy in performance. Yes, there are so many parameters to be taken into account, and yes, there is no silver bullet language which is superior to all. But, is writing JavaScript using the out-of-the-box features provided like the mentioned above array function helping or harming your application performance?

Moreover, client-side javascript is claiming to be a reasonable solution for more than just presentation\view, as end-users computers grow stronger, and networks faster—but can we rely on this when our application requires blazing fast performance and might be a very large and complex one?

To test these questions, I tried comparing a few scenarios and drilled down to understand the results I got. I executed the following tests on Node.js v10.11.0 and in the Chrome browser, both on macOS.

1. Looping Over an Array

The first scenario which came to mind was summing an array of 10k items, this is a valid real-life solution I stumbled upon while trying to fetch a long table of items from the database and enhance it with the total sum, without having an additional query to the DB.

I compared the summing of random 10k items using for, for-of, while, forEach, and reduce. Running the tests 10,000 times returned the following results:

```
 For Loop, average loop time: ~10 microseconds
For-Of, average loop time: ~110 microseconds
ForEach, average loop time: ~77 microseconds
While, average loop time: ~11 microseconds
Reduce, average loop time: ~113 microseconds
```

While googling how to sum an array, reduce was the best-offered solution but it's the slowest. My go-to forEach wasn't much better. Even the newest for-of (ES6) provides inferior performance. It turns out, the good old for loop (and also while) provides the best performance by far —10x better!

How can the newest and recommended solution make JavaScript so much slower? The cause of this pain comes from two main reasons, reduce and forEach requires a call back function to be executed which is called recursively and bloats the stack, and additional operation and verification which are made over the executed code (described here).

2. Duplicating an Array

While this sounds like a less interesting scenario, this is the pillar of immutable functions, which doesn't modify the input when generating an output.

Performance testing findings here again show the same interesting trend—when duplicating 10k arrays of 10k random items, it is faster to use the old school solutions. Again the trendiest ES6 spread operation `[...arr]` and Array from `Array.from(arr)` plus the ES5 map `arr.map(x => x)` are inferior to the veteran slice `arr.slice()` and concatenate `[].concat(arr)`.

```
Duplicate using Slice, average: ~367 microseconds
Duplicate using Map, average: ~469 microseconds
Duplicate using Spread, average: ~512 microseconds
Duplicate using Conct, average: ~366 microseconds
Duplicate using Array From, average: ~1,436 microseconds
Duplicate manually, average: ~412 microseconds
```

3. Iterating Objects

Another frequent scenario is iterating over objects, this is mainly necessary when we try to traverse JSON's and objects, and while not looking for a specific key value. Again there are the veteran solutions like the for-in `for(let key in obj)`, or the later `Object.keys(obj)` (presented in es6) and `Object.entries(obj)` (from ES8) which returns both key and value.

Performance analysis of 10k objects iterations, each of which contains 1,000 random keys and values, using the above methods, reveals the following.

```
Object iterate For-In, average: ~240 microseconds
Object iterate Keys For Each, average: ~294 microseconds
Object iterate Entries For-Of, average: ~535 microseconds
```

The cause is the creating of the enumerable array of values in the two later solutions, instead of traversing the object directly without the keys array. But the bottom line result is still causing concerns.

Bottom Line

My conclusion is clear—if blazing fast performance is key for your application, or if your servers require to handle some load—using the coolest, more readable, cleaner options will blow a major punch to your application performance—which can get up to 10 times slower!

Next time, before blindly adopting the slickest new trends, make sure they also align with your requirements—for a small application, writing fast and a more readable code is perfect—but for stressed servers and huge client-side applications, this might not be the best practice.

Javascript # Performance # Coding # Programming # Arrays

Continue the discussion

More by kadishay

JavaScript V8 Engine Explained

Yotam Kadishay
Jan 23

Javascript

Salary Negotiation Done Right

Yotam Kadishay
Nov 05

Negotiation

Hackernoon Newsletter curates great stories by real tech professionals

Get solid gold sent to your inbox. Every week!

Email

First Name

Last Name

Sign Up

If you are ok with us sending you updates via email, please tick the box. Unsubscribe whenever you want.

Terms of Service

I agree to leave Hackernoon.com and submit this information, which will be collected and used according to Upscribe's privacy policy.

More Related Stories

The 7 Pro Tips To Get Productive With Angular CLI & Schematics

Tomas Trajan
Jan 15

Angular

11 Tips to Improve AngularJS Performance

Alex Kras
Apr 28

Angularjs

1 AMAZING Secret of Every INSANELY Successful Team

Ravi Shankar Rajan
Jun 25

Leadership

0–100 in Django: Starting an app the right way

Jeremy Spencer
Sep 04

Python

ArraySort in typescript

Neeraj Dana
Mar 11

Typescript

Brand-as-Author
Sitewide Billboard

Contact Us
Privacy
Terms