Scott McAllister

@stmcallister

<https://stmcallister/github.io>

Performance Analysis Cage Match! Map, Filter, Reduce vs Loops!

Functions built in to the Array object, like Map, Filter, and Reduce make our code more declarative and readable. But how do these functions perform against for and forEach loops? Come watch these methodologies go head-to-head as we see just how fast each of these functions compare to similar loops!

*Interested in functional programming*

*Declaratively rather than iteratively.*

**Array.map()**

Creates a new array from the results.

**Array.filter()**

*Look at each elements and checks if it passes the test.*

*Makes it more readable? How’s the performance?*

**Array.reduce()**

*It creates a new object, not an array, reduces to a single value*

*Aggregator value in the 1st parameter.*

MDN documentation

**Array.find()**

Returns the value of the first element that satisfies the provided test

Compatible across all Browser with specific versions

**Ramda**

* “A practical functional library for JavaScript programmers”
* Ramda emphasizes a purer functional style. Immutability and side-effect free functions are at the heart of its design philosophy.
* Ramda functions are automatically curried.
* The parameters to Ramda functions are arranged…

**Lodash**

* Lodash makes JS easier by taking the hassle out of working with arrays, numbers, objects, strings. Lodash’s modular methods are great for: ….

**Performance Analysis JS**

* <http://github.com/dg92/Performance-Analysis-JS>
* Performs functions on the array
* Uses console.time() to track time per operation
* Compares functions with for loop

*Uses the time-part of the console API*

**Console.time()**

* Each timer has a unique name
* Up to 10,000 timers running on a given page
* Outputs time elapsed in milliseconds

**Test Object**

{

Id: 1,

Upvotes: 440,

downvotes: 469,

commentCount: 548

}

*Console.time cannot be assigned to a variable*

console.time(‘for loop’)

(for loop here)

Console.timeEnd(‘for loop’)

**Filter**

**Map**

**Reduce**

**Map**

**Comparison of JS native, for loop, for each, Lodash**

*forEach is slower. Use a regular for loop. Less readable, better performance.*

*Chrome in V8 engine, if hot code, it will compile as if there’s no function call at all.*

for of

for(… of …)

*iterate an array instead of an object*

*arrow functions were slower before, now they are optimized.*

*Arrow functions is a little bit faster than regular function*

**Results**

**Node**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
| JS Function | 555.480 |  |  |
| For loop |  |  |  |

**Chrome**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
| JS Function | 823. |  |  |
| For loop |  |  |  |

**Firefox**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
| JS Function | 206.8 |  |  |
| For loop |  |  |  |

**Edge**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
| JS Function | 550.777 |  |  |
| For loop |  |  |  |