Maximum size of an Array in Javascript

Asked 13 years, 7 months ago Modified 3 years, 9 months ago

Viewed 195k times

139

Context: I'm building a little site that reads an rss feed, and updates/checks the feed in the background. I have one array to store data to display, and another which stores ID's of records that have been shown.

Question: How many items can an array hold in Javascript before things start getting slow, or sluggish. I'm not sorting the array, but am using jQuery's inArray function to do a comparison.

The website will be left running, and updating and its unlikely that the browser will be restarted / refreshed that often.

If I should think about clearing some records from the array, what is the best way to remove some records after a limit, like 100 items.

javascript arrays

Share

Improve this question

edited Jun 4, 2016 at 4:01

bren

4,314 ● 3 ● 29 ● 44

Follow

asked May 27, 2011 at 16:14

addedlovely
3,184 ● 3 ● 24 ● 37

6 You will probably run into more problems with the browser leaking memory from toolbars than from the JS code. :) Firefox 4 I point my finger at you. – epascarello May 27, 2011 at 16:18

2 How often are you checking the array (ex 2s interval)? What constitutes sluggish (ex >500ms)? What order of magnitude is your array (ex thousands, millions, billions)? – zzzzBov May 27, 2011 at 16:20

2 do benchmark testing with jsperf.com – VirtualTroll May 27, 2011 at 16:27

I'll be checking and updating the array every minute. And yes sluggish would be a performance hit that starts effecting that load and check, and other animations on the page, hard to define sorry! – addedlovely May 29, 2011 at 6:24

@Amine thanks for the link, looks like that website will be my new best friend :) – addedlovely May 29, 2011 at 6:26

7 Answers

Sorted by: Highest score (default)

198

The maximum length until "it gets sluggish" is totally dependent on your target machine and your actual code, so you'll need to test on that (those) platform(s) to see what is acceptable.

However, the maximum length of an array according to the ECMA-262 5th Edition specification is bound by an unsigned 32-bit integer due to the *ToUint32* abstract operation, so the longest possible array could have 2^{32}-1 = 4,294,967,295 = 4.29 billion elements.

Share Improve this answer Follow

edited Oct 8, 2012 at 0:25

answered May 27, 2011 at 16:20

maerics
156k ● 47 ● 276 ● 299

15 @Barkermn01: the ECMA-262 5th Edition specification uses the abstract operation *ToUint32* for checking the length of an array on any operation that modifies its length, so I think the underlying architecture of the machine (or web browser) is irrelevant. – maerics May 27, 2011 at 16:31

1 hrm nice just read that one awsome 64Bit browser are flaming pointless then, – Barkermn01 May 27, 2011 at 16:41

4 @Barkermn01, 64bit browsers still ahve a lot of other improvements. Remember that being a javascript interpreter isn't the only thing a browser does. – Razor Storm May 27, 2011 at 17:04

2 Wowzer wouldn't of expected it to be that high. OK nice I think I'll be fine! – addedlovely May 29, 2011 at 6:25

1 Actually an array can have at most 4294967295 (2^31-1) elements. See stackoverflow.com/a/12766547/396458 – NullUserException Oct 7, 2012 at 6:34

32

No need to trim the array, simply address it as a circular buffer (index % maxlen). This will ensure it never goes over the limit (implementing a circular buffer means that once you get to the end you wrap around to the beginning again - not possible to overrun the end of the array).

For example:

```
var container = new Array ();
var maxlen = 100;
var index = 0;

// 'store' 1538 items (only the last 'maxlen' items ar
for (var i=0; i<1538; i++) {
   container [index++ % maxlen] = "storing" + i;
}

// get element at index 11 (you want the 11th item in
eleventh = container [(index + 11) % maxlen];

// get element at index 11 (you want the 11th item in
thirtyfifth = container [(index + 35) % maxlen];

// print out all 100 elements that we have left in the
// that it doesn't matter if we address past 100 - cir
// so we'll simply get back to the beginning if we do
for (i=0; i<200; i++) {
   document.write (container[(index + i) % maxlen] + "
}
```

Share Improve this answer

Follow

answered Jul 13, 2012 at 8:55

Lelanthran
1,529 ● 12 ● 19

5 Clever idea, but by doing this you'll potential overwrite data, confusing indexes, and possibly resulting in strange

behavior. – john k Oct 23, 2014 at 2:31

12 The idea is to implement a ring-buffer, so yes - you are intentionally "forgetting" old data (that's what a ring buffer is used for) and that was what the questioner asked for. – Lelanthran Oct 24, 2014 at 6:26

2 I was just bored-clicking around SO and found this response. love the technique with overwriting indexes as needed. – Kyle Hotchkiss Feb 6, 2016 at 2:56

18

Like @maerics said, your target machine and browser will determine performance.

But for some real world numbers, on my 2017 enterprise Chromebook, running the operation:

```
console.time();
Array(x).fill(0).filter(x => x < 6).length
console.timeEnd();
```

- `x=5e4` takes 16ms, good enough for 60fps
- `x=4e6` takes 250ms, which is noticeable but not a big deal
- `x=3e7` takes 1300ms, which is pretty bad
- `x=4e7` takes 11000ms and allocates an extra 2.5GB of memory

So around 30 million elements is a hard upper limit, because the javascript VM falls off a cliff at 40 million elements and will probably crash the process.

EDIT: In the code above, I'm actually filling the array with elements and looping over them, simulating the minimum of what an app might want to do with an array. If you just run `Array(2**32-1)` you're creating a sparse array that's closer to an empty JavaScript object with a length, like `{length: 4294967295}`. If you actually tried to use all those 4 billion elements, you'll definitely crash the javascript process.

Share Improve this answer Follow

edited Mar 2, 2021 at 20:55

answered Nov 7, 2019 at 7:23

Carl Walsh
6,871 ● 3 ● 50 ● 55

8

You could try something like this to test and trim the length:

http://jsfiddle.net/orolo/wJDXL/

```
var longArray = [1, 2, 3, 4, 5, 6, 7, 8];

if (longArray.length >= 6) {
  longArray.length = 3;
}

alert(longArray); //1, 2, 3
```

Run code snippet   Expand snippet

Share   Improve this answer

Follow

edited Mar 9, 2020 at 7:39

Mohammad Faisal

2,392 ● 1 ● 18 ● 27

answered May 27, 2011 at 16:21

orolo

3,951 ● 2 ● 31 ● 30

2   Ended up using slice as I needed to trim from the start of the array, thanks though. – addedlovely May 29, 2011 at 7:41

3

I have built a performance framework that manipulates and graphs millions of datasets, and even then, the javascript calculation latency was on order of tens of milliseconds. Unless you're worried about going over the array size limit, I don't think you have much to worry about.

Share   Improve this answer

Follow

answered May 27, 2011 at 17:07

Razor Storm

12.3k ● 20 ● 95 ● 151

-1

It will be very browser dependant. 100 items doesn't sound like a large number - I expect you could go a lot higher than that. Thousands shouldn't be a problem. What may be a problem is the total memory consumption.

Share Improve this answer

Follow

answered May 27, 2011 at 16:23

rjmunro

28k ● 21 ● 112 ● 134

-1

I have shamelessly pulled some pretty big datasets in memory, and altough it did get sluggish it took maybe 15 Mo of data upwards with pretty intense calculations on the dataset. I doubt you will run into problems with memory unless you have intense calculations on the data and many many rows. Profiling and benchmarking with different mock resultsets will be your best bet to evaluate performance.

Share Improve this answer

Follow

answered May 27, 2011 at 16:24

Stephane Gosselin

9,138 ● 5 ● 44 ● 67