Why doesn't JavaScript support multithreading?

Asked 16 years, 3 months ago Modified 3 years, 1 month ago

Viewed 202k times

342

Is it a deliberate design decision or a problem with our current day browsers which will be rectified in the coming versions?

javascript multithreading browser

Share

Improve this question

Follow

edited Sep 5, 2008 at 0:17

community wiki
7 revs, 5 users 100%
Niyaz

3 See also the answers to the JavaScript and Threads question for information about web workers/worker threads. – Sam Hasler Sep 2, 2008 at 16:37

151 Hello, fellow Googler. You might notice that everything here appears to be quite dated (note this question was asked over 5 years ago.) Since it was asked, web browsers have gained some capabilities that, as far as I can tell, are more or less multithreading. Take a look at Web Workers:

msdn.microsoft.com/en-us/hh549259.aspx – ArtOfWarfare Nov 7, 2013 at 13:37

3 Multithread.js wraps Web Workers and allows for easy multithreading in JS. Works on all new browsers, including iOS Safari. :) – kwh Mar 8, 2014 at 19:03

1 Possible duplicate of JavaScript and Threads – Matt May 23, 2018 at 13:27

you can use (server worker) - A service worker is a script that your browser runs in the background, separate from a web page, opening the door to features that don't need a web page or user interaction. – Omar bakhsh Jul 19, 2020 at 1:04

16 Answers

Sorted by: Highest score (default)

234

JavaScript does not support multi-threading because the JavaScript interpreter in the browser is a single thread (AFAIK). Even Google Chrome will not let a single web page's JavaScript run concurrently because this would cause massive concurrency issues in existing web pages. All Chrome does is separate multiple components (different tabs, plug-ins, etcetera) into separate processes, but I can't imagine a single page having more than one JavaScript thread.

You can however use, as was suggested, `setTimeout` to allow some sort of scheduling and "fake" concurrency. This causes the browser to regain control of the rendering thread, and start the JavaScript code supplied to `setTimeout` after the given number of milliseconds. This is very useful if you want to allow the viewport (what you see) to refresh while performing operations on it. Just looping through e.g. coordinates and updating an element accordingly will just let you see the start and end positions, and nothing in between.

We use an abstraction library in JavaScript that allows us to create processes and threads which are all managed by the same JavaScript interpreter. This allows us to run actions in the following manner:

- Process A, Thread 1
- Process A, Thread 2
- Process B, Thread 1
- Process A, Thread 3
- Process A, Thread 4
- Process B, Thread 2
- Pause Process A
- Process B, Thread 3
- Process B, Thread 4
- Process B, Thread 5
- Start Process A
- Process A, Thread 5

This allows some form of scheduling and fakes parallelism, starting and stopping of threads, etcetera, but it will not be true multi-threading. I don’t think it will ever be implemented in the language itself, since true multi-threading is only useful if the browser can run a single page multi-threaded (or even more than one core), and the difficulties there are way larger than the extra possibilities.

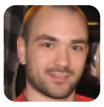
For the future of JavaScript, check this out: https://developer.mozilla.org/presentations/xtech2006/javascript/

Share Improve this answer Follow

edited Aug 1, 2018 at 22:46

user4437749
74 ● 6

answered Sep 2, 2008 at 16:29


Kamiel Wanrooij
12.4k ● 6 ● 38 ● 43

---

101 I think *never implemented* is too narrow a vision. I guarantee web apps will eventually be able to be truly multithreaded (it's only logical, as web apps become more dominant, and hardware becomes more parallel), and as I see it, since JavaScript is the de-facto language of web development, it will eventually have to support multithreading or be replaced by something that does. – devios1 Jun 2, 2011 at 3:02

6 Never is probably a bit of a too bold statement :) but I still think the advantages of true multithreaded javascript are not feasible in the foreseeable future ;) – Kamiel Wanrooij Jun 10, 2011 at 14:40

5 Although I would argue that web workers are more concurrent through a process-model than a thread-model. Web workers use message passing as the means of communication, which is an elegant solution to the 'usual'

concurrency issues in multithreaded applications. I am not sure if they can actually operate on the same objects as the main page concurrently. They cannot access the DOM as far as I know. Most of this is semantics though, web workers look promising for all intents and purposes.
– Kamiel Wanrooij Jun 29, 2012 at 13:28

1 *difficulties there are way larger than the extra possibilities* I'm not sure if you think of all the extra possibilities. I'm especially thinking of cases where webgl is used like in games or graphical visualizations. E.g. consider the new 3D version of Google Maps. In cities with many 3D models my PC needs ~2 min to load everything when many houses have to be rendered. At certain angles neither my graphics card nor my network are working to capacity. But 1 of 8 processors is at 100%. Multithreading is also a big concern in terms of fps as this example shows: youtube.com/watch?v=sJ2p982cZFc – Scindix Jan 11, 2017 at 16:46

34

JavaScript multi-threading (with some limitations) is here. Google implemented workers for Gears, and workers are being included with HTML5. Most browsers have already added support for this feature.

Thread-safety of data is guaranteed because all data communicated to/from the worker is serialized/copied.

For more info, read:

http://www.whatwg.org/specs/web-workers/current-work/

http://ejohn.org/blog/web-workers/

Share Improve this answer answered Apr 30, 2010 at 18:37

Follow

community wiki
Neil

12 But isn't it more of multi-process approach rather than multi-threaded? Threads are known to work within a single heap. – beefeather Aug 18, 2016 at 13:04

2 @beefeather, that's true. It is more of a process approach. – Neil Sep 8, 2016 at 21:44

1 WebWorkers do not have shared memory. To communicate with each other's they send messages. Messages are copies of memory. This is slower as shared memory. – neoexpert Jul 5, 2020 at 11:06

1 @neoexpert That's why you a SharedBufferArray in javascript as they read from the same memory chunk. – user3112634 Feb 5, 2023 at 13:22

25

Traditionally, JS was intended for short, quick-running pieces of code. If you had major calculations going on, you did it on a server - the idea of a JS+HTML *app* that ran in your browser for long periods of time doing non-trivial things was absurd.

Of course, now we have that. But, it'll take a bit for browsers to catch up - most of them have been designed around a single-threaded model, and changing that is not easy. Google Gears side-steps a lot of potential problems by requiring that background execution is isolated - no changing the DOM (since that's not thread-safe), no accessing objects created by the main thread (ditto). While restrictive, this will likely be the most practical design for the near future, both because it simplifies the design of the browser, and because it reduces the risk involved in allowing inexperienced JS coders mess around with threads...

@marcio:

> Why is that a reason not to implement multi-threading in Javascript? Programmers can do whatever they want with the tools they have.

So then, let's not give them tools that are so easy to **misuse** that every other website i open ends up crashing my browser. A naive implementation of this would bring you straight into the territory that caused MS so many headaches during IE7 development: add-on authors played fast and loose with the threading model, resulting in hidden bugs that became evident when object lifecycles changed on the primary thread. BAD. If you're writing multi-threaded ActiveX add-ons for IE, i guess it comes with the territory; doesn't mean it needs to go any further than that.

Share Improve this answer Follow

edited May 23, 2017 at 12:26

Community Bot
1 ● 1

answered Sep 2, 2008 at 16:19

Shog9
159k ● 36 ● 235 ● 240

7 "it reduces the risk involved > in allowing inexperienced JS coders > mess around with threads" Why is that a reason not to implement multi-threading in Javascript? Programmers can do whatever they want with the tools they have. If its good or bad, it's their problem. With the Google Chrome process model it can't even affect other applications. :)
– Marcio Aguiar Sep 2, 2008 at 17:08

5 @Shog9 - "Let's not give [programmers] tools that are so easy to misuse that every other website I open ends up crashing my browser." - What? By that same logic, no language should have multithreading, because if they offered that every other program you tried opening would crash. Except it doesn't work that way. Threading exists in most languages and most novice programmers don't touch it, and most of those that do don't put it into production, and those apps which are never become popular or widely used.
– ArtOfWarfare Nov 7, 2013 at 13:21

13

Do you mean why doesn't the language support multithreading or why don't JavaScript engines in browsers support multithreading?

The answer to the first question is that JavaScript in the browser is meant to be run in a sandbox and in a machine/OS-independent way, to add multithreading support would complicate the language and tie the language too closely to the OS.

Share Improve this answer answered Sep 2, 2008 at 16:09

Follow

matt b
140k ● 66 ● 284 ● 350

13

I don't know the rationale for this decision, but I know that you can simulate some of the benefits of multi-threaded programming using setTimeout. You can give the illusion of multiple processes doing things at the same time, though in reality, everything happens in one thread.

Just have your function do a little bit of work, and then call something like:

```
setTimeout(function () {
    ... do the rest of the work...
}, 0);
```

And any other things that need doing (like UI updates, animated images, etc) will happen when they get a chance.

Share Improve this answer

Follow

answered Sep 2, 2008 at 16:16

pkaeding
37.6k ● 31 ● 106 ● 142

Most of the cases I would want to use a `loop` inside the `setTimeout` but apparently that doesn't work. Have you done anything like that or do you have a hack? an example would be for array of 1000 element, I expect to use two for loops inside two `setTimeout` calls such that the first loops through and print element `0..499`, the second loops through and print element `500..999`. – benjaminz Nov 1, 2016 at 13:34

Usually the technique is to save the state and continue. For example, say you want to print 0 to 1000, you might print 0 to 499 and then do the setTimeout trick with the argument 500. The code inside would know to take the argument (500) and start the loop from there. – Eyal Feb 1, 2017 at 8:50

12

Node.js 10.5+ supports **worker threads** as experimental feature (you can use it with **--experimental-worker** flag enabled): https://nodejs.org/api/worker_threads.html

So, the rule is:

- if you need to do **I/O bound ops**, then use the internal mechanism (aka callback/promise/async-await)
- if you need to do **CPU bound ops**, then use worker threads.

Worker threads are intended to be long-living threads, meaning you spawn a background thread and then you communicate with it via message passing.

Otherwise, if you need to execute a heavy CPU load with an anonymous function, then you can go with **https://github.com/wilk/microjob**, a tiny library built around worker threads.

Share Improve this answer Follow

answered Sep 6, 2018 at 13:52

community wiki
Wilk

6

Just as matt b said, the question is not very clear. Assuming that you are asking about multithreading support in the language: because it isn't needed for 99.999% of the applications running in the browser currently. If you really need it, there are workarounds (like using window.setTimeout).

In general multithreading is very, very, very, very, very, very hard (did I say that it is hard?) to get right, unless you put in extra restrictions (like using only immutable data).

Share Improve this answer Follow

answered Sep 2, 2008 at 16:24
Grey Panther
13.1k ● 7 ● 49 ● 67

3

Intel has been doing some open-source research on multithreading in Javascript, it was showcased recently on GDC 2012. Here is the link for the video. The research group used OpenCL which primarily focuses on Intel Chip sets and Windows OS. The project is code-named RiverTrail and the code is available on GitHub

Some more useful links:

Building a Computing Highway for Web Applications

Share Improve this answer Follow

answered Apr 1, 2012 at 20:24

community wiki
Rohit Reddy Korrapolu

3

Currently some browsers do support multithreading. So, if you need that you could use specific libraries. For example, view the next materials:

- https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers (support background threads);
- https://keithwhor.github.io/multithread.js/ (the library).

Share Improve this answer Follow

answered Dec 7, 2015 at 12:38

community wiki
sergzach

2

Javascript is a single-threaded language. This means it has one call stack and one memory heap. As expected, it executes code in order and must finish executing a piece code before moving onto the next. It's synchronous, but at times that can be harmful. For example, if a function takes a while to execute or has to wait on something, it freezes everything up in the meanwhile.

Share Improve this answer Follow

answered Jan 21, 2020 at 15:02

community wiki
Omar bakhsh

This doesn't answer the question of *why* Javascript is the way it is, just explains single threading. – JonHerbert Apr 27, 2023 at 8:07

1

It's the implementations that doesn't support multi-threading. Currently Google Gears is providing a way to use some form of concurrency by executing external processes but that's about it.

The new browser Google is supposed to release today (Google Chrome) executes some code in parallel by separating it in process.

The core language, of course can have the same support as, say Java, but support for something like Erlang's concurrency is nowhere near the horizon.

Share Improve this answer Follow

answered Sep 2, 2008 at 16:14

Greg Roberts
798 ● 9 ● 14

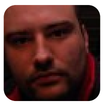
As far as I have heared Google Chrome will have multithreaded javascript, so it is a "current implementations" problem.

0

Share Improve this answer

Follow

answered Sep 2, 2008 at 16:05



BlaM

28.8k ● 33 ● 93 ● 106

0

Without proper language support for thread syncronization, it doesn't even make sense for new implementations to try. Existing complex JS apps (e.g. anything using ExtJS) would most likely crash unexpectedly, but without a `synchronized` keyword or something similar, it would also be very hard or even impossible to write new programs that behave correctly.

Share Improve this answer

Follow

answered Mar 8, 2010 at 8:54

community wiki

Erich Kitzmueller

0

Actually multi-threading is not related with the language itself. Here is a Multithreaded Javascript Engine for .NET. It works pretty well with multi-threading scenarios. It integrates with C# runtime so all the synchronization logic is similar to C#. You can start/await/wait tasks, and you can start threads. You can even put locks. Following sample demonstrates parallel loop using Javascript syntax in .NET runtime.

https://github.com/koculu/topaz

```
var engine = new TopazEngine();
engine.AddType(typeof(Console), "Console");
topazEngine.AddType(typeof(Parallel), "Parallel");
engine.ExecuteScript(@"
var sharedVariable = 0
function f1(i) {
    sharedVariable = i
}
Parallel.For(0, 100000 , f1)
Console.WriteLine(`Final value: {sharedVariable}`);
");
```

Besides that, Microsoft is working on Napa.js, a multi-threading supported Node.js clone.

https://github.com/microsoft/napajs

Share Improve this answer Follow

answered Oct 31, 2021 at 6:09

community wiki
Ahmed Yasin Koculu

-3

Multi-threading with javascript is clearly possible using webworkers bring by HTML5.

Main difference between webworkers and a standard multi-threading environment is memory resources are not shared with the main thread, a reference to an object is not visible from one thread to another. Threads communicate by exchanging messages, it is therefore possible to implement a synchronzization and concurrent method call algorithm following an event-driven design pattern.

Many frameworks exists allowing to structure programmation between threads, among them OODK-JS, an OOP js framework supporting concurrent programming https://github.com/GOMServices/oodk-js-oop-for-js

Share Improve this answer Follow

edited Sep 11, 2016 at 13:56

community wiki
2 revs
OBDM

6 Sharing memory is the exact definition of a Thread opposed to a separate process (eg fork() vs exec()). Threads can share objects, processes must use IPC. Web Workers are not multithreading. – felixfbecker Nov 14, 2016 at 15:42

-4

However you can use eval function to bring concurrency TO SOME EXTENT

```
/* content of the threads to be run */
var threads = [
        [
            "document.write('Foo <br/>');",
            "document.write('Foo <br/>');",
            "document.write('Foo <br/>');",
            "document.write('Foo <br/>');",
            "document.write('Foo <br/>');",
            "document.write('Foo <br/>');",
            "document.write('Foo <br/>');",
            "document.write('Foo <br/>');",
            "document.write('Foo <br/>');",
            "document.write('Foo <br/>');"
        ],
        [
            "document.write('Bar <br/>');",
            "document.write('Bar <br/>');",
            "document.write('Bar <br/>');",
            "document.write('Bar <br/>');",
            "document.write('Bar <br/>');",
            "document.write('Bar <br/>');",
            "document.write('Bar <br/>');",
            "document.write('Bar <br/>');",
            "document.write('Bar <br/>');"
        ]
    ];

window.onload = function() {
    var lines = 0, quantum = 3, max = 0;

    /* get the longer thread length */
    for(var i=0; i<threads.length; i++) {
    if(max < threads[i].length) {
        max = threads[i].length;
    }
    }

    /* execute them */
    while(lines < max) {
    for(var i=0; i<threads.length; i++) {
        for(var j = lines; j < threads[i].length &
j++) {
            eval(threads[i][j]);
        }
    }
    lines += quantum;
    }
}
```

Share Improve this answer   edited Feb 11, 2017 at 6:58

Follow

community wiki
2 revs
Madhusoodan P