Specific Programming Languages | Server-Side JavaScript | Node.js | +5

What are the disadvantages of using Node.js?

Answer | Request | Follow 487 | Comments 2+ | Share 8 | Downvote | •••

26 Answers

Quildreen Motta, Designer of the Purr & Siren programming languages
45.7k Views

Ah, so this question is fairly old, but I'll throw in an analysis of Node today (0.10~0.11), and contrast with the alternatives for building applications in the same niche (heavily I/O bound applications). Most of the points will consider the usage of Node.js with JavaScript, with some considerations for alternative languages.

Node is a neat platform, it has an attractive ecosystem, and it's got one of the only package managers that actually work. It also rides on the popularity of the JavaScript language, which gives it somewhat of a hype status. However, there are a few points where it falls short, even within the niche that it was designed for.

Bad concurrency primitives
Node's core is entirely built on top of the premise of asynchronous I/O. Being a single-threaded platform focused on I/O bound applications on top of a language that does not guarantee purity and has no support for co-routines, this is an understandable choice. It is much better than heavy-weight threads with shared state, as you have in Java and similar mainstream languages. But it's a poor model of concurrency to expose to the platform, none the less.

JavaScript doesn't really have any good primitive for concurrency (although generators will allow cooperative concurrency, and can be used experimentally in Node today), and the core, along most libraries in the ecosystem, just straight up expect you to use continuation-passing style, which leads to non-compositional code, codebases that are difficult to reason about, and lots and lots of duplication and call-site specific glueing.

In JavaScript land the alternative with most acceptance is Promises/A+, but Promises/A+ are Considered Harmful, specially due to how they automatically catch *synchronous* errors thrown in a function (but nothing else), and the automatic flattening of “thenables”, which leads to behaviour that is difficult to reason about, and impossibility of expressing some abstractions. Truly monadic specifications for promises were proposed, but ultimately rejected by the community and TC-39[1]. With generators, people started experimenting with Communicating Sequential Processes. A move to CSP would be pretty interesting, but they're fairly experimental right now.

Single-Threaded
Node being single-threaded means that one does not need to care about the problems of synchronising between threads, or shared mutable state (given JS's semantics). However, it also means that unlike preemptive concurrency, the programmers themselves have to decide how to deal with concurrency, with the default being *no concurrency at all!* It's easy to write some part of the code that might take a long time to finish in some edge case, and then lock the whole system up. Not a good thing to happen.

Green threads (like in Haskell), or lightweight processes with message passing for communication (like in Erlang) would be a better idea.

Lack of maturity
Most of the core libraries have reached the status of stable, and you can trust them to usually do the right thing. But the ecosystem itself is still fairly immature. It's also difficult to assess the quality of a particular module (core or otherwise) because of the lack of features for ensuring the quality of code from JavaScript itself — for example, in Haskell you can rely on the type system, QuickCheck and SmallCheck tests, and the language semantics; JS has nothing like that (there are some ports of QuickCheck, though).

The way npm is structured also makes it quite difficult to find trustable packages, and the ease of publishing your own package, alongside the Unix-philosophy that runs through part of the community (small modules that do only one thing, and one thing well), make it harder to spot packages that are reliable and proven.

Reliance on stringly-typed[2] programming

Related Questions

- What are some successful systems which uses node.js?
- How does Linkedin use Node.js?
- Why should I use Node.js?
- Due to Node.js event-loop nature under what circumstances would you say never to use Node.js?
- Does Quora use node.js?
- What companies in Ireland are using Node.js?
- What are the disadvantages of developing a web application using Node.js instead of Python?
- How is Instagram using Node.js?
- What are the advantages and disadvantages of using MongoDB/Mongoose and CouchDB/cradle with Node.js?
- Does Google plan to use Node.js?

More Related Questions

Question Stats

487 Followers
235,179 Views
Last Asked Aug 8
Edits

There's a rampant cult of “strings are easy” in the community, which leads to many bugs due to unnecessary reliance on strings, rather than proper data structures. Some of the XSS bugs found in Express are due to this mentality. Since it's a cultural thing, and JavaScript itself doesn't do much to make the alternatives be perceived as easier, it ends up making it even harder to trust thirdy-party libraries.

Hard to make things fault-tolerant
This was somewhat alleviated with the introduction of Domain, but they're still too low level, and don't give you all of the tools necessary to make systems reliable and fault-tolerant, as Erlang/OTP does. JavaScript's semantics don't help here as well, there are no mechanisms to handle or recover from errors that are thrown asynchronously, for example, and processes in Node are not lightweight like Erlang's.

JavaScript's semantics and culture
Even if you end up using an alternative and more principled programming language, like PureScript, most of the code you'll be running on top of will have JavaScript's problems all over (and this includes Node's core). This ranges from the implicit data structures conversions that the specification defines under the hood for all operators in the language (besides ===), to the culture of heavily ad-hoc overloading variadic methods depending on the value's type tag.

These are about all the things I can think off the top of my head, but there are some other fairly specific problems in the tooling around Node, some problems in npm (like the very existence of *peerDependencies)*, which are a direct result of the semantics of JS. As these are not directly related to every possible Node application, and you might never run into them, I decided to leave them off.

[1]: Promise.cast and Promise.resolve
[2]: Stringly Typed

Written Sep 20, 2014 · View Upvotes · Answer requested by Miguel Paraz

Upvote | 130 Downvote Comments 2+

Zef Hemel, programming language designer and implementer
77.9k Views

I like Javascript and node.js, but there's one disadvantage: its asynchronous programming model.

People told me: "asynchronous programming is the way to the future, it's the *right* way to achieve semi-concurrency". And to be honest, I believed them. I got used to using callbacks, three-level deep anonymous function callbacks. I got used to it. I was happy.

Until I used Python for a server-side project and remembered how much simpler the synchronous programming style is. How cleaner your code looks. I suppose I had the Stockhold syndrome for a while there.

But seriously, while asynchronous programming may be a good solution to the concurrency problem, it comes at a cost: a life filled with callbacks.

Updated Jul 28, 2012 · View Upvotes

Upvote | 320 Downvote Comments 24+

Max Seiden, Making Data Interactive @ Platfora
21.9k Views

I'm currently the lead architect and developer for (what's becoming) a decently large Nodejs application, and can confidently say that a lack of inherent code organization is a huge disadvantage.[1] It gets especially exacerbated when the development team as a whole isn't familiar with asynchronous programing or standard design patterns. There are just too many ways for code to get unruly and unmaintainable.

There are definitely modules that ease those pains though, such as: the cross-browser WebSocket for realtime apps., Express, and Mongoose. The issue I have with them is that they haven't had enough "competition" from other modules, and thus haven't matured to the point of being "generic" solutions. On a personal note, I still prefer Connect over Express, simply because it provides just enough functionality to make HTTP in Node manageable, but not so much that it forces you into one paradigm or another. I'd also suggest using ws instead of the cross-browser WebSocket for realtime apps. for the same reason.

A quick case study: LinkedIn uses Nodejs to power the JSON API for their mobile site/app. It does not however, power their backend. The Node services most likely draw from a caching layer, or use RPC/REST to query necessary data from the backend services. The advantage there is that they get really high throughput and a straightforward solution to horizontal scaling, without huge memory or CPU usage. (JVM)

[EDITS]

[1] - Mocha, is actually a great, flexible testing framework. I highly recommend it.

Updated Apr 19, 2013 · View Upvotes

Upvote | 69 Downvote Comments 4+

Jeremy Ashkenas
86.8k Views

First, a caveat: I think Node.js is wonderful, and has an excellent chance of becoming the gold standard for web applications over the coming year ... but you asked about the disadvantages, so:

- Unstable API: The Node API has a habit of changing in backwards-incompatible ways from release to release, and frequent changes to your codebase are required to keep things running on the latest version. That said, things are *supposed* to be more stable since 0.2.0 has been released.
- Lack of a standard library: JavaScript is a language with a beautiful core, but an absolutely anemic standard library -- things that you would take for granted as part of other server-side language installs will simply not be there.
- Lack of libraries in general: Need to find a mature database interface? An ORM? An image processing library? An XML parser? An S3 client? -- Since JavaScript hasn't yet enjoyed years of popularity on the server side, things like these are either brand-new and relatively untested, or still-in-the-works.
- It's not battle-tested yet: I think it's still a relatively open question about what the pain points are for large Node.js applications -- and until we have some in production for months and years, we're not going to know where the trouble might lie. There isn't anything obvious to be careful of, but it's uncharted territory, with respect to performance, security, and maintainability.

That's about all I can think of -- note that I didn't say the JavaScript was a disadvantage, or the evented model, or V8... I, for one, can't wait to have strong enough library support to justify using it for green field projects, and we seem to be approaching that day quite rapidly.

Written Aug 29, 2010 · View Upvotes

Upvote | 251 Downvote Comments 9+

Oleg Podsechin, I've been doing ServerJS for a few years now and have a few fairly large proj...
11.2k Views

I would say that Node's biggest problem at the moment is the amount of duplication of effort taking place when it comes to the ecosystem of packages & the amount of effort spent finding the best package to use for any given task that stems from this. Fortunately this is the area that Isaac Zimmitti Schlueter and other core team members will be focusing on next.

The issues associated with Node's asynchronous programming style mentioned in the other answers are less of a concern for two reasons:

- For certain types of problems where one deals primarily with events and maintains little state the async model is an ideal fit
- When you do need to maintain a lot of state and prefer a synchronous programming style, one can use the node-fibers (https://github.com/laverdet/node...) coroutine library or my common-node (https://github.com/olegp/common-...) package which provides a higher level synchronous abstraction layer

Edit based on experience of using Node for two years after the original answer:

At StartHQ we found that a major issue with Node that isn't easy to fix is stability. A misbehaving or poorly implemented package you depend on can easily crash your server, so in practice one needs to run all node processes with an additional watchdog process, like node-supervisor.

Updated Dec 12, 2013 · View Upvotes

Upvote | 24 Downvote Comment

Michael Chen, Exp Web Engineer
8.8k Views

I have used Node.js extensively and designed several node services running in production.

1. Lack of libraries support for database drivers, memcached and etc.
For example, lack of support for memcached compression. You can find libraries that implement the basic functions. I need the query timeout function for a database query but I can't find it in the mysql client that we use.
2. Event driven model will confuse a lot of programmers who are new to JavaScript. The callback chain can get very long which makes it harder to maintain. In web programming, you often need to grab some data to determine the next step, which means a long chain of callbacks.
3. Node.js is most likely not suited for general web development until a better web optimized framework come out. It's easy to write a simple web page using Node.js but you will need to implement a lot of the basic functions in other more mature web frameworks.
4. Hard memory limit. I ran into the memory limit problem very early on and it took a while to resolve that.
5. Node API is changing very rapidly which might cause incompatibility and bugs when you upgrade. One of our production service ran out of memory after upgrading to the latest node version. It took a few weeks to narrow down it's the node version we are using that caused the problem because it happened randomly on some servers after several days. The memory leak issue did not surface until testing the system extensively under load for a long period of time.

Updated Aug 31, 2012 · View Upvotes

Upvote | 12 Downvote Comments 3

Ohad Kravchick, Doing node.js for a year now, mostly using CoffeeScript. Loving it.
7.4k Views

Node.js is a single-threaded event-based system. As long as you are using any nonblocking I/O calls frequently, requests can be serviced in a timely fashion. However, if you start hogging the CPU even for a few milliseconds for a given request, all other requests cannot be serviced. Even if you have a single URL that's computationally expensive (think about your back office site), your entire service degrades. Node.js doesn't give a good workaround for such situations and you might have to break that URL into its own service, causing quite an administrative headache; in cases like this you wish it would just spawn a new thread for such a request.

While on the surface it sounds quite difficult to have node.js spend quite a lot of CPU cycles before calling any nonblocking I/O, such situations can arise, for example, when parsing or validating a decent amount of data naively, converting between formats naively (audio, video, but even large XML to JSON conversion), rendering really big and deep templates, and of course, actually doing math.

Written May 5, 2014 · View Upvotes

Upvote | 9 Downvote Comment 1

Rich Sadowsky, Using node.js to build Jibo's server-side infrastructure
4.8k Views

I realize this is an old question. Many of the answers are old too. My "answer" here is more a comment on all the answers that say "node.js is single threaded" as a disadvantage. A couple of points. As some say, this can be an advantage in some ways by keeping you away from deadlock bugs. But more important, the node.js ecosystem does not force you to program in a single thread when you are writing libraries and modules. You can use native C++ code to handle your multi-threaded needs. This is especially relevant when writing an embedded system or a super high performance server app.

The better way to state this attribute of node.js is that the primary event loop is single threaded. If you've done native code or V8 programming you already know that it is possible to create other event loops or do threads in your native code. The magic is that these functions jump back into the single event loop when they emit an event. Frankly this is so incredibly powerful that I think many people are missing some of the capabilities of it as an entire system. Node.js is more than just the JavaScript event loop deep down inside. It has a very powerful engine from Google's browser world at it's core.

If you look at the source code to any of node's standard modules, you'll see they exploit threading quite a bit in modules like fs (file system). I love it that they included a provision

to use this in a relatively well documented way (if you can handle systems level documentation).

Coming back around to the original question: you could say that having to write native addons in C++ is a disadvantage. I personally don't see it that way. C++ is the one constant throughout my 3 decades as a professional developer. It is powerful. It is dangerous. You have to know what you're doing. It can be hell to debug. There's a reason the V8 engine and so many other systems are written it it.

Written Oct 4, 2015 · View Upvotes

Upvote | 12 Downvote Comment

Myroslav Opyr, CTO of Quintagroup
6.1k Views

One of the issues with JavaScript programming in Node.js is its event-driven (callback) approach as solution to implied scalability. Developer has to think in terms of asynchronous execution model that is more complex then linear blocking I/O programming.

I'm "twisted" on Python thus can provide of example that work better for average developer. Eventlet (http://eventlet.net/) library provides nonblocking I/O in Coroutines that implies threading-like Blocking development paradigm without threading-related problems (i.e. asynchronous).

If Eventlet-like library exist for Javascript and Node.js would work in Coroutines style, it will be adopted by average developers sooner IMHO.

Written Sep 2, 2010 · View Upvotes

Upvote | 7 Downvote Comments 1+

Raghavendra Kidiyoor, Believe in no technology religion
6.6k Views

Here's a view from Alex Payne, co-author of the book on Scala. http://al3x.net/2010/07/27/node.... He also presents studies which conclude that event-based model maybe a bad idea for high concurrent servers (http://www.usenix.org/events/hot...) IMO, we should use the model that fits the problem domain and not sway one way or other.

Written Sep 1, 2010 · View Upvotes

Upvote | 32 Downvote Comments 2+

Alec Menke, Wait... This isn't Java?
2.2k Views

While answering this, I'm going to try to make sure all the "disadvantages" I list out are both core to Node.js *--not complaints about JavaScript--* and aren't rewrites of previous issues mentioned on this thread. I really recommend you read Quildreen Motta's answer. It's a great answer, and I won't end up going over any of what he discussed here.

The distinct issues I have with Node.js fall into two categories **library usage** and **the v8 engine** which it's built on.

You must download everything.

Since Node.js, by nature, is extremely lightweight, you end up downloading 200 files worth of external libraries, anytime you want to get started on even a small project. Everything downloads extremely quick, and npm is beautiful, but it just kills me that I have 400 different copies of express scattered throughout my computer.

Libraries can be very difficult to get started with.

I know I said, I wouldn't complain about JavaScript, but please bare with me, as this includes more facets than just JS.

Since a lot of the Node.js libraries you work with will contain a host of functions inherited from other libraries and C++ binaries, and since JavaScript itself isn't statically typed, it's very hard to know what you're working with. If you can't find excellent documentation online, for a given library, you just can't use it.

Tough to share code with the front-end.

Yes, with Node.js you can now share front-end and back-end code, but because Node.js relies on AMD this can be somewhat annoying to do. It isn't a huge deal, but unless you use some sort of pre-compiler, such as browserify, you end up having to come up with some clever hacks to make the transition seamless.

As we know, being clever has a cost.

Stuck, slowly, progressing with the rest of the web.

Mostly, this is awesome. V8 is truly amazing, but now we're stuck progressing with Chrome and the rest of the web. We get ECMAScript 6 at the same time as everyone else, even though, with Node.js we never need to worry about other browsers supporting these features.

I NEED IMPORT STATEMENTS AND DESTRUCTURING NOW!

Difficult to extend.

In my experience, it has been really tough to extend Node.js using C++. The documentation on their site isn't clear.. I need to spend more time diving into Google's documentation before I know for sure that this is a legitimate concern.

Written Mar 18 · View Upvotes

Upvote | 5 Downvote Comments 1+

Marco Pegoraro, I am a frontend developer. I do JS. To me knowledge is a product worth to share.
3.9k Views

I am part of a team that decided to introduce NodeJS as part of the company stack.

I think that the NPM ecosystem improves code reusability due to the micro-modules approach, testing is really easy with javascript (grunt-mocha-chai-sinon) and code organisation is fine thanks to CommonJS.

Not to mention the bright sides of async approach without bothering with multithreading!

The only really "disadvantage" that I see is **a lack in people knowledge around enterprise Javascript** programming.

Many many people still think of Javascript as a "jQuery plugin language" and they don't really study it, but this is not NodeJS problem, is people.

We are addressing this "disadvantage" by running internal courses in the company and I see that people who didn't knew the real power of async js they really enjoy a lot to learn it.

They turn to be really active in the learning process, thankful to the company for providing new knowledge and, eventually, better resources than before (before === java).

Written Sep 9, 2014 · View Upvotes

Upvote | 7 Downvote Comment

Alexander Gugel
9.3k Views

- Callback Hell: Everything is asynchronous by default. This means you are likely to end up using tons of nested callbacks. Nevertheless, there are a number of solutions to this problem, e.g. caolan/async or Understanding promises in node.js. This problem is specific to JavaScript, not Node.JS.

- Single-threaded: Node.js is single-threaded. You can take advantage of multiple CPUs, but in general everything is designed to use the Event-Loop in order to achieve extraordinary performance. This can also be an advantage, since e.g write conflicts on files aren't that relevant.

- JavaScript: There is a reason why is it called Node.**JS**. JavaScript has been designed in 10 days and that's partly obvious. I really like JS, but there are some obvious drawbacks (but every programming language sucks, just read YourLanguageSucks - Theory.org Wiki). Prototypal instantiation wasn't originally planned to be integrated. Prototypes are much more elegant, but since Netscape wanted to jump onto the Java-train, they wanted to introduce the *new* keyword.

- Event Loop: The Event Loop is the core of Node.js and it's a genius idea. But: Don't use Node.js for blocking, CPU-intensive tasks. Node.js is not suited for stuff like that. Node.js is suited for I/O stuff (like web servers).

TL;DR
Don't use Node.js for CPU-intensive tasks.
Node.js rocks for servers.

JavaScript partly sucks. Use CoffeeScript in order to solve this problem.

Node.js certainly has some disadvantages, but it is currently one of the best tools out there in order to create asynchronous, non-blocking apps. It's great.

Written Jun 28, 2014 · View Upvotes

Upvote | 29 Downvote Comments 2

Carlos Matias La Borde, active Node.js user
7.8k Views

I generally love Node. But one thing that I find really, really annoying is how it handles modules. Specifically how many modules seem to be nested rather than stored under some root level directory and accessed as dependencies from there.

I've had a couple projects break because the path was too long. **The path was too long.** What? As in it would look like:

blah/node_modules/blah.0/node_modules/blah/node_modules/blah/node_modules/blah

Or something like that... On and on and on. It broke my git on Windows. I tried long path, but eventually I had to delete plugins just because their path was too long. That's just stupid. It's also ugly and unnecessary.

It would be so easy to fix, too. With an index of modules, or module directories, or even just picking a shorter path name as a crude solution (e.g. nm instead of node_modules).

EDIT: The just released Node 5 has NPM 3.3 which supposedly now supports a fix for this. Kudos Team Node!

Updated Nov 16, 2015 · View Upvotes · Answer requested by Igor Silva

Upvote | 36 Downvote Comments 7+

Michael J. Ryan
5.6k Views

Okay, just realizing how old this question and some of the answers are... I'll state a few disadvantages.

The event loop. While the event loop is one of its' largest advantages, not understanding how this works is one of its' biggest disadvantages as well. Do *NOT* do long running, synchronous, cpu bound logic in Node.js. Spin it off to a worker queue, another thread/process (use a pool), or any number of other ways to remove the heavy lifting from your main event loop.

Not understanding functional programming techniques. Many people come into node.js and JavaScript in general and try to bring an OOP mindset with them. You *can* use OOP style hierarchy with JS, but it's not considered a best practice. Highly decoupled, well tested modules with plain objects, and functional style tends to work better for larger projects.

npm - another best/worst advantage situation. Get your own caching npm setup for internal use, especially if you are setting up a CI/CD process. This is critical. Some say to check in your dependencies (node_modules) into source control, I disagree... especially if your deployment and development platforms don't match and you have native modules. Checking in binary dependencies doesn't give you much when you are developing on windows or mac, and deploying to linux. Having an npm module/version hickup when you need to deploy a bug fix is painful.

I absolutely love the freedom and flexibility that node offers, and sometimes you are re-inventing the wheel. I feel at this point, the biggest issues come down to technique and experience in the platform. It's become popular faster than the community can inform those new to the environment.

Written May 7, 2014 · View Upvotes

Upvote | 16 Downvote Comment

Ravi Raj, FOSS enthusiast,realtime system architect,building scalable,fault-tolerant di...
930 Views

I'm a newbie on Node.js. I am sharing my doubts here.

1. Single threaded, No multi threading support, can not utilize multiple core without hack (please ignore here context switching over threads cost .. it's negotiable)

For big heavy compute tasks like image encoding, Node.js can fire up child processes or send messages to additional worker processes. In this design, you'd have one thread managing the flow of events and N processes doing heavy compute tasks and chewing up the other 15 CPUs.

For scaling throughput on a web service, you should run multiple Node.js servers on one box, one per core and split request traffic between them. This provides excellent CPU-affinity and will scale throughput nearly linearly with core count.

2. NO scalable (You have to scale OS & hardware)

3. Code will be messy after few months, tons of callbacks

4. Node is based on event loop , if by mistake we did memory leaks or anything else(CPU intensive tasks) in JS, Event loop is started blocking, then we have only one option either shift load to another box or restart server.js

5. Dealing with mysql type db is painful

6. Need to understand javascript basics (need good conceptual understanding like In java script, everything is object except undefined & null.

http://www.addyosmani.com/resour...

7. Production Setup is not easy if you want robust,redundant & reliable application

http://stackoverflow.com/questio...

Written Jan 21 · View Upvotes

Upvote | 2 Downvote Comment 1

David Bonnie, Hi-tech enthusiast.
3.3k Views

I think the node community is not as helpful as other solutions, like ruby on rails. I site in three node-related channels, and virtually nobody speaks. It takes days to get an answer and researching. It's new, and it is becoming easier. There are mature solutions at this point, but it takes some time to find them.

You might think you found something, spend time implementing it, then realize it isn't supported anymore or something else is better, or just is broken.

I'm a front-end programming and if you've done any ajax with javascript, then the event-based model isn't too difficult to get your head around.

I like node for two reasons, it is fun -- and it will impress employers in a job interview. I would not want to be responsble for an developing an enterprise application under these conditions though.

Written Sep 23, 2012 · View Upvotes

Upvote | 4 Downvote Comments 1+

Nikhil Singh
3.8k Views

A drawback of nodejs that I encountered was the hard coded memory limits. See http://stackoverflow.com/questio... overcome this issue, it is recommended that you use a cluster of your nodejs app. But this too is not very satisfactory.
More reading: http://code.google.com/p/v8/issu...

Another drawback is getting used to async programming model. If you know what will be the output of the following code, then you are good or else get used to it.

```
for (var i = 0; i < 4; i++) {
   setTimeout(function () {
      console.log(i);
   }, 1000);
}
```

Written Jun 21, 2012 · View Upvotes

Upvote | 5 Downvote Comments 2

Michal Boška
1.5k Views

Basically it boils down to these things:

- **Javascript** as an inconsistent, not type-safe language that can get messy really easily. If any serious logic or complex (business) requirements are in play, you want to stay away from javascript and use something more predictable, where you can easily reason about the code. That's what the "script" part in Java"script" means.
- **Too fragile event loop model** that cannot scale beyond single thread in a single process. Do anything more CPU-complex inside an event loop and you degrade the performance of your entire application really fast. The problem is also that you cannot get past this limitation really well, because there's no way you can spawn a new "worker thread" inside a Node process.
- **Too wild-west libraries ecosystem** where everyone thinks writing 1000+th npm library that does the same as the other 1000 is cool. That's when you as a library user start to get confused which library should you use in production system that should be stable. 0.0.1 or 0.23.5 or 0.50.3, or 1.0.0 (thank god a library with major version 1) *alpha*? And also this "micro-module" thing many praise as good - you totally lose control about what code are you running (do you really have time to audit every single string-left-pad library)? I mentioned left-pad on purpose, because it so happens that a guy simply removed his library because he just felt like it and nearly half of the node.js "internet" broke down - everyone depended on his library. That's just too much of a risk.
- **Async programming model** while being good for some use cases is not always the best thing you can use, not always really needed and you can reason about synchronous code more easily.

And some other reasons i can't think of right now :o) I've done some node.js programming, the project is now in production, but... i decided not to use node.js when i can avoid it (which is pretty much always, because even for asynchronous IO apps there are much better alternatives)

Written May 22 · View Upvotes

Upvote | 2 Downvote Comment

Marcin Krol
2.7k Views

1. Callback spaghetti.

Colleagues tell me that I should not do too many callbacks to other callbacks, "if you're doing it, you're doing smth wrong".

I have come to conclusion that it's like saying "you should drive home to work on the road that is short, straight and does not have many traffic lights".

Well, life has different ideas on what we need and should be doing. The very reason to use programming language and code is ability to implement complex logic, and with growing systems that have proven themselves working that's pretty much inevitable.

Nobody has any problem debugging 50-level deep stack traces in Java or Python: it's a non-issue, the complexity of the problem and particular input data set is a problem.

For some reasons, brains of too many people are not well suited to async callback programming model. This limits it.

Erlang, D or even Go language concurrency combined with high performance seem so much more straightforward in terms of brain cycles. And the brain cycles are what's expensive today.

2. Illusion of Javascript being common ground for front end and back end.

Front end and back end require different knowledge, focus and interests. The two shall not mingle much. Common language will not help significantly. Choosing common language is almost a solution in search of a problem.

Written Apr 16, 2014 · View Upvotes

Upvote | 7 Downvote Comments 4+

Asadujzaman Shamim, Muslim, software engineer, brainless.
374 Views

Lack of good frameworks like ruby on rails. There is no good framework to code in mvc pattern. I am not very expert. But i can see nodejs has some very immature frameworks like sails js. And they use the worst ORM waterline. From my little and awful experience, it

is really great for developing api. But to make a website or cms like we do using RoR really needs much more effort.

Written Mar 18

Upvote Downvote Comment

Murali Vivekanandan
2.3k Views

- My 2 cents:

- Ecosystem is still not there compared to Java. Its fast maturing, but its going to be a while before you see a Kafka or Drools in Node.js.
- You are going to miss simple things like Database migrations. Nothing like Rails migrations or Play Framework Evolutions is available.
- I have given few more not so great features of Node here: http://bit.ly/U3HwPZ

Written Jul 16, 2014

Upvote Downvote Comment

Tim Spann
2.4k Views

It looks great and can definitely scale. It has Yahoo behind it with Mojito. But I always tend to keep my eye on any technology that is too hyped and has too much media push before it is a 1.0. It's definitely fun to work with. I did server side javascript at the turn of the century with Netscape Enterprise server and it was ok. Vert.x is looking pretty cool because it supports JavaScript, Java and Python. The performance specs on Vert.x look great. Node.JS has so many great modules, frameworks and tools it's hard to ignore. I am prototyping a few small systems now. We could have used this for Database America's yellow pages engine!

Written Jun 2, 2012 · View Upvotes

Upvote | 2 Downvote Comment

Akshay Bagule, i drink some node.js everyday..
103 Views

Nodejs is currently preferred for light weight application ,and i would like to explain that single threaded is advantage of node.js and not a disadvantage,if you are using it for heavy or large scale applications then your choice is wrong,you cant blame node for that.
I will not expalin every point here ,but the only disadvantage i faced in nodejs is load balancing,it is not aware of internal load balancing,you have to go for external resource like HAProxy ,etc.
Thank you.

Written Aug 8 · Answer requested by Nikita Benjamin

Upvote Downvote Comment

Pablo Cúbico
2.4k Views

I think the disadvantages pointed out here are fair, but seeing companies like Paypal (a top player with a very sensitive mission) embrace NodeJS is definitely saying that there's people working seriously on pushing Node to the next level.

Lenny Markus, Paypal engineer, showed off KrakenJS on the JSConf Uruguay, which seems to be a serious effort to, as he said, make node "not a toy language". They went through a lot of criticism for switching to Node but it seems that is really taking off.

Checkout KrakenJS:

Upvote Downvote Comment 1

Andrej Bartko
2k Views

I oppose to other answers I wouldn't say single thread asynchronous programming is a disadvantage. As a mater of fact it is a huge advantage. Bear in mind that Node.js wasn't build as intention to replace everything else. It was built mainly for IO bound application and it shines in it and it is highly scalable.

Javascript as a language is in my opinion an advantage as well. Fast learning curve and seamless transition for many front end devs.

However should I mention any disadvantages it would be lack of maturity. A lot of APIs are still marked as unstable and could change in the future. ES6 support. The community is still somehow divided about a lot of parts of ES6 not to mention ES7... Node.js is in my opinion quite slow to adopt all the new ES6 features and many of them still suffer from performance issues.

Last but not least Node.js is still mainly a domain of open source projects and javascript enthusiasts. The demand in corporate environment is right now more less limited to front end build scripting with few pioneer companies realising it's potential.

Written Sep 6, 2015 · View Upvotes

Upvote | 2 Downvote Comment 1

7 Answers Collapsed (Why?)