**Node JS Processing Model – Single Threaded Model with Event Loop Architecture**

[Rambabu Posa](http://www.journaldev.com/author/rambabu) April 8, 2015 [Node.js](http://www.journaldev.com/dev/node-js)

In our previous posts, we have discussed about some [Node JS Basics](http://www.journaldev.com/7397/introduction-to-node-js-basics), [Node JS Components](http://www.journaldev.com/7423/node-js-components-modules-npm-install-update-uninstall-example) and [Node JS installation](http://www.journaldev.com/7402/node-js-environment-setup-node-js-installation) on Windows and Mac OS X operating systems. Before starting some Node JS programming examples, we will discuss about “How Node JS works under-the-hood, what type of processing model it is following, How Node JS handles concurrent request with Single-Threaded model” etc. in this post.

**Node.js Single Threaded Model with Event Loop**

As we have already discussed, Node JS applications uses “Single Threaded Model with Event Loop” Architecture to handle multiple concurrent clients.

As a Developer, we have already involved in implementing many web applications by using technologies like JSP, Spring MVC, ASP.NET, HTML, Ajax, jQuery etc. But all these technologies follow “Multi-Threaded Request-Response” Architecture to handle multiple concurrent clients.

We all are already very familiar with “Multi-Threaded Request-Response” Architecture. But why Node JS Platform has chosen different Architecture to develop web applications. What is the major differences and pros and cons between these two Architectures.

Any UI or Web Developer or Programmer can learn Node JS and develop applications very easily. It is very easy to learn and not a big task. However I feel, without understanding and learning Node JS Internals, we cannot design and develop Node JS Applications very well. So before starting developing Node JS Applications, first we will learn Node JS Platform internals.

**Node.js Platform**

Node JS Platform uses “Single Threaded Model with Event Loop” Architecture to handle multiple concurrent clients. Then how it really handles concurrent client requests without using Multiple Threads. What is Event Loop model? We will discuss these things one by one.

Before discussing “Single Threaded Model with Event Loop” Architecture, first we will go through our familiar “Multi-Threaded Request-Response” Architecture.

**Traditional Web Application Processing Model**

Any Web Application developed without Node JS, typically follows “Multi-Threaded Request-Response” model. Simply we can call this model as Request/Response Model.

Client sends Request to the Server, then server do some process based on Client’s Request and prepare Response and send it back to the Client.

This model uses HTTP protocol. As HTTP is a Stateless Protocol, this Request/Response model is also Stateless Model. So we can call this as Request/Response Stateless Model.

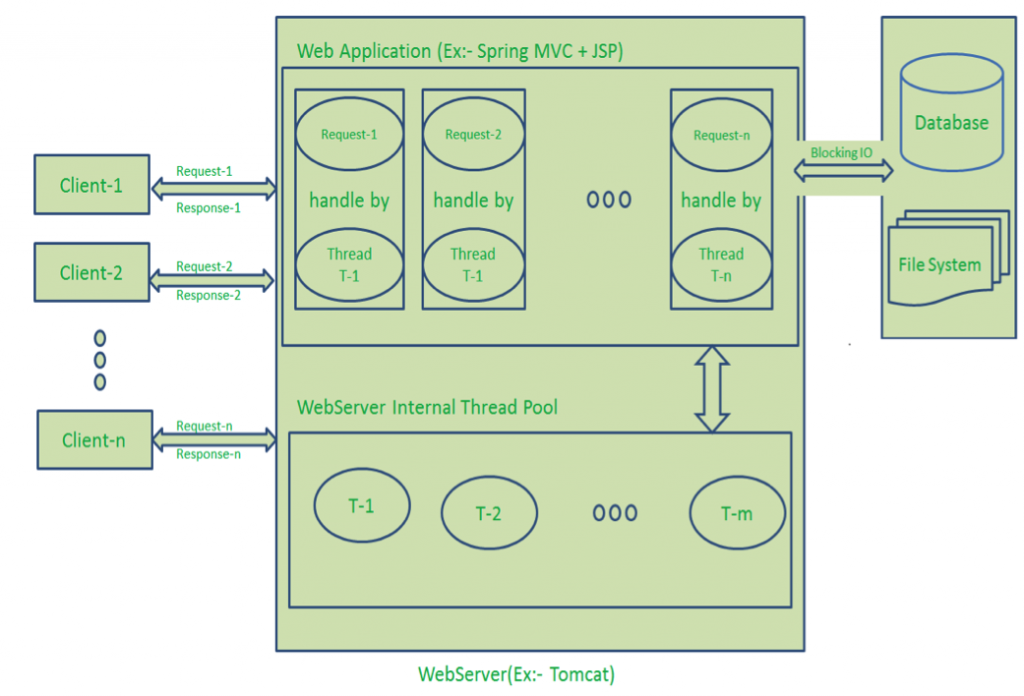
However, this model uses Multiple Threads to handle concurrent client requests. Before discussing this model internals, first go through the diagram below.

**Request/Response Model Processing Steps**:

* Clients Send request to Web Server.
* Web Server internally maintains a Limited Thread pool to provide services to the Client Requests.
* Web Server is in Infinite Loop and waiting for Client Incoming Requests
* Web Server receives those requests.
  + Web Server pickup one Client Request
  + Pickup one Thread from Thread pool
  + Assign this Thread to Client Request
  + This Thread will take care of reading Client request, processing Client request, performing any Blocking IO Operations (if required) and preparing Response
  + This Thread sends prepared response back to the Web Server
  + Web Server in-turn sends this response to the respective Client.

Server waits in Infinite loop and performs all sub-steps as mentioned above for all n clients. That means this model creates one Thread per Client request.

If more Clients’ Requests require Blocking IO Operations, then almost all Threads are busy in preparing their responses. Then remaining Client’s Requests should wait for longer time.

[](http://1988780851.rsc.cdn77.org/wp-content/uploads/2015/04/Request-Response-Model.png)

**Diagram Description**:

* Here “n” number of Clients Send request to Web Server. Let us assume they are accessing our Web Application concurrently.
* Let us assume, our Clients are Client-1, Client-2… and Client-n.
* Web Server internally maintains a Limited Thread pool. Let us assume “m” number of Threads in Thread pool.
* Web Server receives those requests one by one.
  + Web Server pickup Client-1 Request-1, Pickup one Thread T-1 from Thread pool and assign this request to Thread T-1
    - Thread T-1 reads Client-1 Request-1 and process it
    - Client-1 Request-1 does not require any Blocking IO Operations
    - Thread T-1 does necessary steps and prepares Response-1 and send it back to the Server
    - Web Server in-turn send this Response-1 to the Client-1
  + Web Server pickup another Client-2 Request-2, Pickup one Thread T-2 from Thread pool and assign this request to Thread T-2
    - Thread T-2 reads Client-1 Request-2 and process it
    - Client-1 Request-2 does not require any Blocking IO Operations
    - Thread T-2 does necessary steps and prepares Response-2 and send it back to the Server
    - Web Server in-turn send this Response-2 to the Client-2
  + Web Server pickup another Client-n Request-n, Pickup one Thread T-n from Thread pool and assign this request to Thread T-n
    - Thread T-n reads Client-n Request-n and process it
    - Client-n Request-n require heavy Blocking IO and computation Operations
    - Thread T-n takes more time to interact with external systems, does necessary steps and prepares Response-n and send it back to the Server
    - Web Server in-turn send this Response-n to the Client-n

If “n” is greater than “m” (Most of the times, its true), then server assigns Threads to Client Requests up to available Threads. After all m Threads are utilized, then remaining Client’s Request should wait in the Queue until some of the busy Threads finish their Request-Processing Job and free to pick up next Request.

If those threads are busy with Blocking IO Tasks (For example, interacting with Database, file system, JMS Queue, external services etc.) for longer time, then remaining clients should wait longer time.

* Once Threads are free in Thread Pool and available for next tasks, Server pickup those threads and assign them to remaining Client Requests.
* Each Thread utilizes many resources like memory etc. So before going those Threads from busy state to waiting state, they should release all acquired resources.

**Drawbacks of Request/Response Stateless Model**:

* Handling more and more concurrent client’s request is bit tough.
* When Concurrent client requests increases, then it should use more and more threads, finally they eat up more memory.
* Sometimes, Client’s Request should wait for available threads to process their requests.
* Wastes time in processing Blocking IO Tasks.

**Node JS Application Processing Model**

Node JS Platform does NOT follow Request/Response Multi-Threaded Stateless Model. It follows Single Threaded with Event Loop Model. Node JS Processing model mainly based on Java Script Event based model with Java Script Callback mechanism.

You should have some good knowledge about how Java Script’s Events and Callback mechanism works. If you don’t know, Please go through those posts or tutorials first and get some idea before moving to the next step in this post.

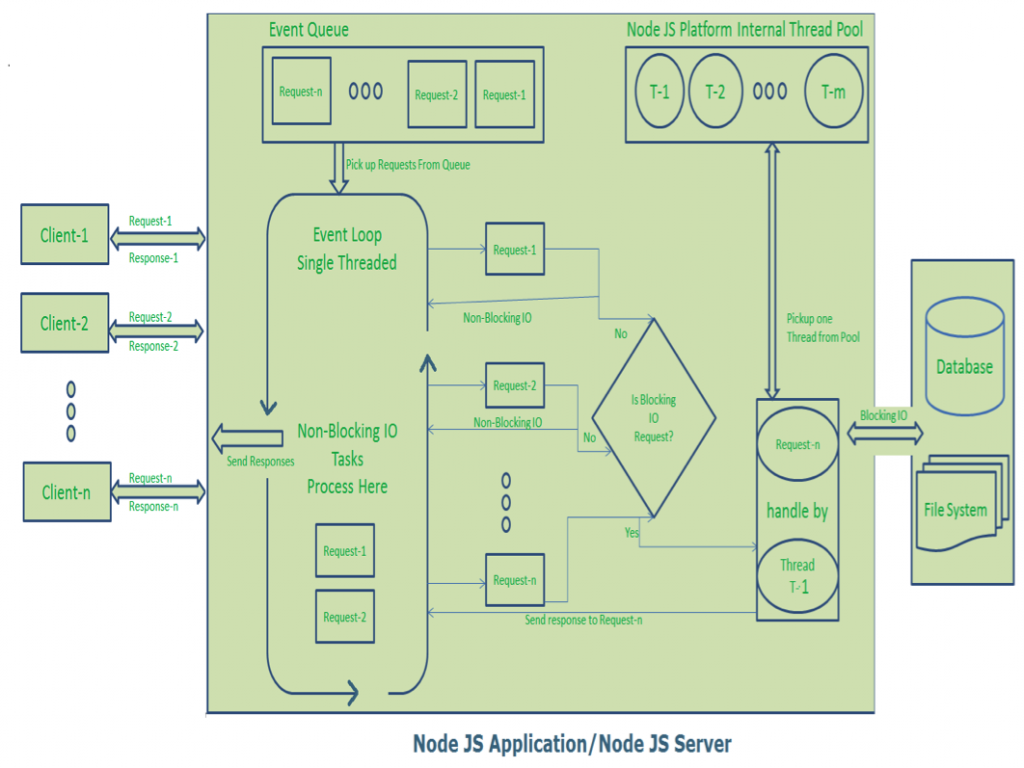
As Node JS follows this architecture, it can handle more and more concurrent client requests very easily. Before discussing this model internals, first go through the diagram below.

I tried to design this diagram to explain each and every point of Node JS Internals.

The main heart of Node JS Processing model is “Event Loop”. If we understand this, then it is very easy to understand the Node JS Internals.

**Single-Threaded Event Loop Model Processing Steps**:

* Clients Send request to Web Server.
* Node JS Web Server internally maintains a Limited Thread pool to provide services to the Client Requests.
* Node JS Web Server receives those requests and places them into a Queue. It is known as “Event Queue”.
* Node JS Web Server internally has a Component, known as “Event Loop”. Why it got this name is that it uses indefinite loop to receive requests and process them. (See some Java Pseudo code to understand this below).
* Event Loop uses Single Thread only. It is main heart of Node JS Platform Processing Model.
* Even Loop checks any Client Request is placed in Event Queue. If no, then wait for incoming requests for indefinitely.
* If yes, then pick up one Client Request from Event Queue
  + Starts process that Client Request
  + If that Client Request Does Not requires any Blocking IO Operations, then process everything, prepare response and send it back to client.
  + If that Client Request requires some Blocking IO Operations like interacting with Database, File System, External Services then it will follow different approach
    - Checks Threads availability from Internal Thread Pool
    - Picks up one Thread and assign this Client Request to that thread.
    - That Thread is responsible for taking that request, process it, perform Blocking IO operations, prepare response and send it back to the Event Loop
    - Event Loop in turn, sends that Response to the respective Client.

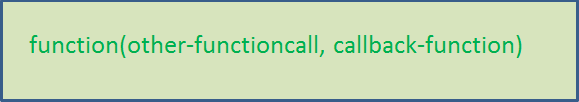
[](http://1988780851.rsc.cdn77.org/wp-content/uploads/2015/04/NodeJS-Single-Thread-Event-Model.png)

**Diagram Description**:

* Here “n” number of Clients Send request to Web Server. Let us assume they are accessing our Web Application concurrently.
* Let us assume, our Clients are Client-1, Client-2… and Client-n.
* Web Server internally maintains a Limited Thread pool. Let us assume “m” number of Threads in Thread pool.
* Node JS Web Server receives Client-1, Client-2… and Client-n Requests and places them in the Event Queue.
* Node JS Even Loop Picks up those requests one by one.
  + Even Loop pickups Client-1 Request-1
    - Checks whether Client-1 Request-1 does require any Blocking IO Operations or takes more time for complex computation tasks.
    - As this request is simple computation and Non-Blocking IO task, it does not require separate Thread to process it.
    - Event Loop process all steps provided in that Client-1 Request-1 Operation (Here Operations means Java Script’s functions) and prepares Response-1
    - Event Loop sends Response-1 to Client-1
  + Even Loop pickups Client-2 Request-2
    - Checks whether Client-2 Request-2does require any Blocking IO Operations or takes more time for complex computation tasks.
    - As this request is simple computation and Non-Blocking IO task, it does not require separate Thread to process it.
    - Event Loop process all steps provided in that Client-2 Request-2 Operation and prepares Response-2
    - Event Loop sends Response-2 to Client-2
  + Even Loop pickups Client-n Request-n
    - Checks whether Client-n Request-n does require any Blocking IO Operations or takes more time for complex computation tasks.
    - As this request is very complex computation or Blocking IO task, Even Loop does not process this request.
    - Event Loop picks up Thread T-1 from Internal Thread pool and assigns this Client-n Request-n to Thread T-1
    - Thread T-1 reads and process Request-n, perform necessary Blocking IO or Computation task, and finally prepares Response-n
    - Thread T-1 sends this Response-n to Event Loop
    - Event Loop in turn, sends this Response-n to Client-n

Here Client Request is a call to one or more Java Script Functions. Java Script Functions may call other functions or may utilize its Callback functions nature.

So Each Client Request looks like as shown below:

[](http://1988780851.rsc.cdn77.org/wp-content/uploads/2015/04/javascript-callback-mechanism.png)

For Example:

|  |  |
| --- | --- |
| 1  2  3 | function1(function2,callback1);  function2(function3,callback2);  function3(input-params); |

**NOTE**: –

* If you don’t understand how these functions are executed, then I feel you are not familiar with Java Script Functions and Callback mechanism.
* We should have some idea about Java Script functions and Callback mechanisms. Please go through some online tutorial before starting our Node JS Application development.

**Advantages of Single-Threaded Event Loop Model**

* Handling more and more concurrent client’s request is very easy.
* Even though our Node JS Application receives more and more Concurrent client requests, there is no need of creating more and more threads, because of Event loop.
* Node JS application uses less Threads so that it can utilize only less resources or memory

**Event Loop Pseudo Code**

As I’m a Java Developer, I will try to explain “How Event Loop works” in Java terminology. It is not in pure Java code, I guess everyone can understand this. If you face any issues in understanding this, please drop me a comment.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public class EventLoop {  while(true){              if(Event Queue receives a JavaScript Function Call){                  ClientRequest request = EventQueue.getClientRequest();                              If(request requires BlokingIO or takes more computation time)                                      Assign request to Thread T1                              Else                                    Process and Prepare response                    }              }  } |

I hope you understand Node JS Platform internals very well. If you face any difficulties in understanding this in first read, try to read this post few more times.

If you still have questions, please drop me a comment. We will try to discuss them in details.

Java vs node

In the history of computing, 1995 was a crazy time. First Java appeared, then close on its heels came JavaScript. The names made them seem like conjoined twins newly detached, but they couldn't be more different. One of them compiled and statically typed; the other interpreted and dynamically typed. That's only the beginning of the technical differences between these two wildly distinct languages that have since shifted onto a collision course of sorts, thanks to Node.js.



[Take control! 30 essential OS X command-line tips](http://www.infoworld.com/resources/95593/mac-os-x/take-control-30-essential-os-x-command-line-tips)

Go beyond the graphical user interface and take full advantage of Mac OS X at the command line

[Read Now](http://www.infoworld.com/resources/95593/mac-os-x/take-control-30-essential-os-x-command-line-tips)

If you’re old enough to have been around back then, you might remember Java’s early, epic peak. It left the labs, and its hype meter pinned. Everyone saw it as a revolution that would stop at nothing less than a total takeover of computing. That prediction ended up being [only partially correct](http://www.infoworld.com/article/2612433/java/java-forever--12-keys-to-java-s-enduring-dominance.html). Today, Java dominates Android phones, enterprise computing, and some embedded worlds like Blu-ray disks.

**[ Get the low-down on the best JavaScript code tools with our review of** [**JavaScript editors**](http://www.infoworld.com/article/2920786/javascript/review-10-javascript-editors-compared.html#tk.ifw-fsb) **and** [**JavaScript IDEs**](http://www.infoworld.com/article/2925050/javascript/review-7-javascript-ides-put-to-the-test.html#tk.ifw-fsb)**. | Streamline your development of fast websites, rich APIs, and real-time apps with these** [**13 fabulous frameworks for Node.js**](http://www.infoworld.com/article/2606426/application-development/153472-13-fabulous-frameworks-for-Node.js.html)**. | Keep up with hot topics in app dev with InfoWorld's** [**Strategic Developer**](http://www.infoworld.com/blog/strategic-developer/) **blog and** [**Application Development newsletter**](http://www.infoworld.com/newsletters/signup.html)**. ]**

For all its success, though, Java never established much traction on the desktop or in the browser. People touted the power of applets and Java-based tools, but gunk always glitched up these combinations. Servers became Java’s sweet spot.

Meanwhile, what programmers initially mistook as the dumb twin has come into its own. Sure, JavaScript tagged along for a few years as HTML and the Web pulled a Borg on the world. But that changed with AJAX. Suddenly, the dumb twin had power.

Then Node.js was spawned, turning developers’ heads with its speed. Not only was JavaScript faster on the server than anyone had expected, but it was often faster than Java and other options. Its steady diet of small, quick, endless requests for data have since made Node.js more common, as Web pages have grown more dynamic.

While it may have been unthinkable 20 years ago, the quasi-twins are now locked in a battle for control of the programming world. On one side are the deep foundations of solid engineering and architecture. On the other side are simplicity and ubiquity. Will the old-school compiler-driven world of Java hold its ground, or will the speed and flexibility of Node.js help JavaScript continue to gobble up everything in its path?

**Where Java wins: Rock-solid foundation**

I can hear the developers laughing. Some may even be dying of heart failure. Yes, Java has glitches and bugs, but relatively speaking, it's the Rock of Gibraltar. The same faith in Node.js is many years off. In fact, it may be decades before the JavaScript crew writes nearly as many regression tests as Sun/Oracle developed to test the Java Virtual Machine. When you boot up a JVM, you get 20 years of experience from a solid curator determined to dominate the enterprise server. When you start up JavaScript, you get the work of an often cantankerous coalition that sometimes wants to collaborate and sometimes wants to use the JavaScript standard to launch passive-aggressive attacks.

[Resources](http://www.infoworld.com/resources)

* White Paper

[4 Strategic Issues To Address Before Embarking On A Big Data Journey](http://resources.infoworld.com/ccd/assets/104366/detail)

* [](http://resources.infoworld.com/ccd/assets/104349/detail)

White Paper

[Cisco ACI and Citrix NetScaler: Opening the Way to Data Center Agility](http://resources.infoworld.com/ccd/assets/104349/detail)

[See All](http://www.infoworld.com/resources)

Top of Form



Bottom of Form

Top of Form

Bottom of Form

**Where Node wins: Ubiquity**

Thanks to Node.js, JavaScript finds a home on the server and in the browser. Code you write for one will more than likely run the same way on both. Nothing is guaranteed in life, but this is as close as it gets in the computer business. It's much easier to stick with JavaScript for both sides of the client/server divide than it is to write something once in Java and again in JavaScript, which you would likely need to do if you decided to move business logic you wrote in Java for the server to the browser. Or maybe the boss will insist that the logic you built for the browser be moved to the server. In either direction, Node.js and JavaScript make it much easier to migrate code.

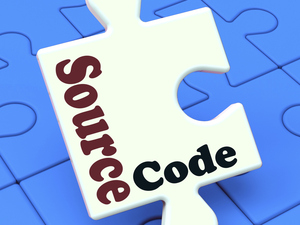
**Where Java wins: Better IDEs**

Java developers have Eclipse, NetBeans, or IntelliJ, three top-notch tools that are well-integrated with debuggers, decompilers, and servers. Each has years of development, dedicated users, and solid ecosystems filled with plug-ins.

Meanwhile, most Node.js developers type words into the command line and code into their favorite text editor. Some use Eclipse or Visual Studio, both of which support Node.js. Of course, the surge of interest in Node.js means new tools are arriving, some of which, like [IBM’s Node-RED](http://nodered.org/) offer intriguing approaches, but they're still a long way from being as complete as Eclipse. WebStorm, for instance, is a solid commercial tool from JetBrains, linking in many command-line build tools.

Of course, if you're looking for an IDE that edits and juggles tools, the new tools that support Node.js are good enough. But if you ask your IDE to let you edit while you operate on the running source code like a heart surgeon slices open a chest, well, Java tools are much more powerful. It's all there, and it's all local.

You Might Also Like

* [](http://www.infoworld.com/article/3043505/ios/government-hints-it-may-demand-ios-source-code-signing-key.html)

[Government hints it may demand iOS source code, signing key](http://www.infoworld.com/article/3043505/ios/government-hints-it-may-demand-ios-source-code-signing-key.html)

* [](http://www.infoworld.com/article/3043064/security/two-year-old-java-flaw-re-emerges-due-to-broken-patch.html)

[Two-year-old Java flaw re-emerges due to broken patch](http://www.infoworld.com/article/3043064/security/two-year-old-java-flaw-re-emerges-due-to-broken-patch.html)

* [Justice Department slams Apple's 'corrosive' rhetoric in its latest court...](http://www.infoworld.com/article/3043028/security/justice-department-slams-apples-corrosive-rhetoric-in-its-latest-court-filing.html)

**Where Node wins: Build process simplified by using same language**

Complicated build tools like Ant and Maven have revolutionized Java programming. But there's only one issue. You write the specification in XML, a data format that wasn't designed to support programming logic. Sure, it's relatively easy to express branching with nested tags, but there's still something annoying about switching gears from Java to XML merely to build something.

**Where Java wins: Remote debugging**

Java boasts incredible tools for monitoring clusters of machines. There are deep hooks into the JVM and elaborate profiling tools to help identify bottlenecks and failures. The Java enterprise stack runs some of the most sophisticated servers on the planet, and the companies that use those servers have demanded the very best in telemetry. All of these monitoring and debugging tools are quite mature and ready for you to deploy.

**Where Node wins: Database queries**

Queries for some of the newer databases, like CouchDB, are written in JavaScript. Mixing Node.js and CouchDB requires no gear-shifting, let alone any need to remember syntax differences.

Meanwhile, many Java developers use SQL. Even when they use the Java DB (formerly Derby), a database written in Java for Java developers, they write their queries in SQL. You would think they would simply call Java methods, but you’d be wrong. You have to write your database code in SQL, then let Derby parse the SQL. It's a nice language, but it's completely different and many development teams need different people to write SQL and Java.

**Where Java wins: Libraries**

There is a huge collection of libraries available in Java, and they offer some of the most serious work around. Text indexing tools like Lucene and computer vision toolkits like OpenCV are two examples of great open source projects that are ready to be the foundation of a serious project. There are plenty of libraries written in JavaScript and some of them are amazing, but the depth and quality of the Java code base is superior.

**Where Node wins: JSON**

When databases spit out answers, Java goes to elaborate lengths to turn the results into Java objects. Developers will argue for hours about POJO mappings, Hibernate, and other tools. Configuring them can take hours or even days. Eventually, the Java code gets Java objects after all of the conversion.

Many Web services and databases return data in JSON, a natural part of JavaScript. The format is now so common and useful that many Java developers use the JSON formats, so a number of good JSON parsers are available as Java libraries as well. But JSON is part of the foundation of JavaScript. You don't need libraries. It's all there and ready to go.

**Where Java wins: Solid engineering**

It's a bit hard to quantify, but many of the complex packages for serious scientific work are written in Java because Java has strong mathematical foundations. Sun spent a long time sweating the details of the utility classes and it shows. There are BigIntegers, elaborate IO routines, and complex Date code with implementations of both Gregorian and Julian calendars.

JavaScript is fine for simple tasks, but there’s plenty of confusion in the guts. One easy way to see this is in JavaScript’s three different results for functions that don't have answers: undefined, NaN, and null. Which is right? Well, each has its role -- one of which is to drive programmers nuts trying to keep them straight. Issues about the weirder corners of the language rarely cause problems for simple form work, but they don't feel like a good foundation for complex mathematical and type work.

**Where Node wins: Speed**

People love to praise the speed of Node.js. The data comes in and the answers come out like lightning. Node.js doesn't mess around with setting up separate threads with all of the locking headaches. There's no overhead to slow down anything. You write simple code and Node.js takes the right step as quickly as possible.

This praise comes with a caveat. Your Node.js code better be simple and it better work correctly. If it deadlocks, the entire server could lock up. Operating system developers have pulled their hair out creating safety nets that can withstand programming mistakes, but Node.js throws away these nets.

**Where Java wins: Threads**

Fast code is great, but it's usually more important that it be correct. Here is where Java’s extra features make sense.

Java's Web servers are multithreaded. Creating multiple threads may take time and memory, but it pays off. If one thread deadlocks, the others continue. If one thread requires longer computation, the other threads aren’t starved for attention (usually).

If one Node.js request runs too slowly, everything slows down. There's only one thread in Node.js, and it will get to your event when it's good and ready. It may look superfast, but underneath it uses the same architecture as a one-window post office in the week before Christmas.

There have been decades of work devoted to building smart operating systems that can juggle many different processes at the same time. Why go back in time to the ’60s when computers could handle only one thread?

**Where Node wins: Momentum**

Yes, all of our grandparents' lessons about thrift are true. Waste not; want not. It can be painful to watch Silicon Valley’s foolish devotion to the “new” and “disruptive,” but sometimes cleaning out the cruft makes the most sense. Yes, Java can keep up, but there's old code everywhere. Sure, Java has new IO routines, but it also has old IO routines. Plenty of applet and util classes can get in the way.

**Where both win: Cross-compiling from one to the other**

The debate whether to use Java or Node.js on your servers can and will go on for years. As opposed to most debates, however, we can have it both ways. Java can be cross-compiled into JavaScript. Google does this frequently with Google Web Toolkit, and some of its most popular websites have Java code running in them -- Java that was translated into JavaScript.

There's a path in the other direction, too. JavaScript engines like [Rhino](https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino) run JavaScript inside your Java application where you can link to it. If you're really ambitious, you can [link](http://rbackhouse.blogspot.com/2011/03/using-google-v8-javascript-engine-in.html) in [Google’s V8 engine](https://code.google.com/p/v8/).

Voilà. All of the code can link to each other harmoniously and you don't need to choose.