Instantly Search Tutorials...

Node JS Architecture – Single Threaded Event Loop

RAMBABU POSA — 87 COMMENTS

Today we will look into Node JS Architecture and Single Threaded Event Loop model. In our previous posts, we have discussed about Node JS Basics, Node JS Components and Node JS installation.

Table of Contents [hide]

Node JS Architecture

Before starting some Node JS programming examples, it's important to have an idea about Node JS architecture. 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 Event Loop Model

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

There are many web application 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 are already familiar with “Multi-Threaded Request-Response” architecture because it’s used by most of the web application frameworks. But why Node JS Platform has chosen different architecture to develop web applications. What is the major differences between multithreaded and single threaded event loop architecture.

Any web developer can learn Node JS and develop applications very easily. However without understanding 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 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 concepts one by one.

Before discussing “Single Threaded Event Loop” architecture, first we will go through famous “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 processing based on clients request, 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 clients Requests should wait for longer time.

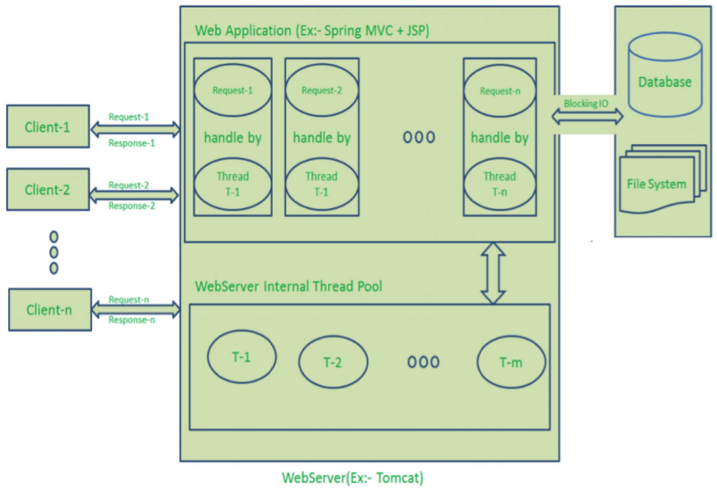


Diagram Description:

- 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 Architecture – Single Threaded Event Loop

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 Javascript Event based model with Javascript callback mechanism.

You should have some good knowledge about how Javascript 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.

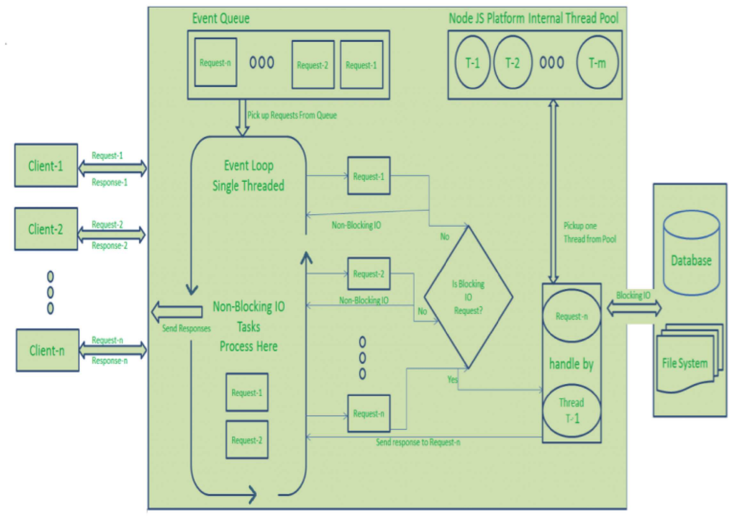
The main heart of Node JS Processing model is "Event Loop". If we understand this, then it is very easy to understand the Node JS Internals.

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.



Node JS Application/Node JS Server

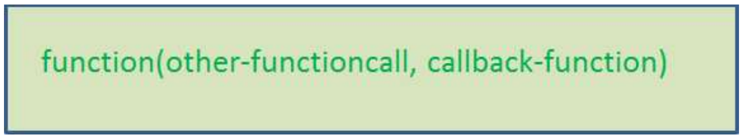
Diagram Description:

- 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:


function(other-functioncall, callback-function)

For Example:

```
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.

Node JS Architecture – Single Threaded Event Loop Advantages

1. Handling more and more concurrent client's request is very easy.
2. 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.
3. 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.

```
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
                  }
            }
}
```

That's all for Node JS Architecture and Node JS single threaded event loop.