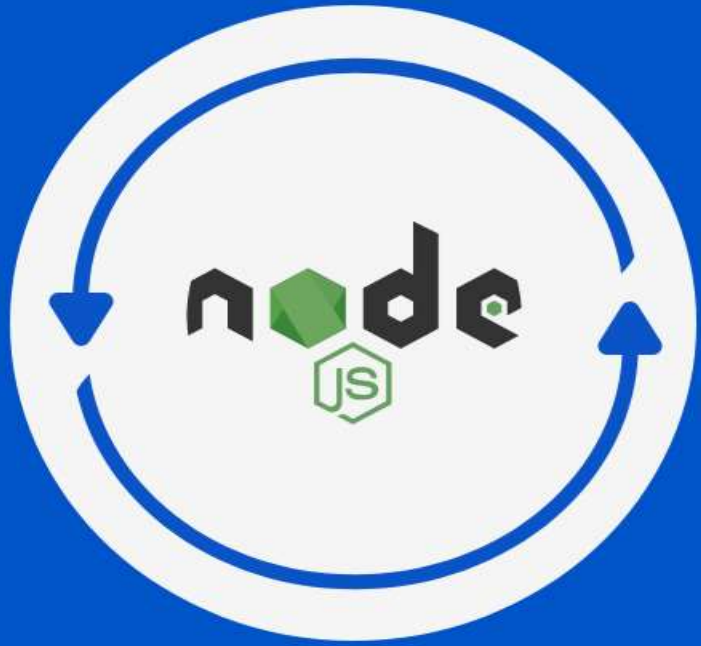
A Beginner's Guide To Eventloops in Node.Js

https://www.fibonalabs.com/

Fibonalabs

A BEGINNER'S GUIDE TO

EVENT LOOPS IN NODE.JS



Node.js turned 12 on May 27th this year, so what's a better time than now to brush up our knowledge about the most important feature of Node.js: Event Loops!

Here’s an index of what you can expect from this blog:

> What is Node.js?
> What is The Event Loop?
> Event Loop Explained
> Phases Overview
> Conclusion

Let’s get started!

> What is Node.js?

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

> What is The Event Loop?

This event loop is something that allows Node.js to perform non-blocking I/O operations which we read about earlier. Event loops allow this despite the fact that JavaScript is single-threaded, by offloading operations to the system kernel whenever possible.

Since most modern operating systems/kernels are multi-threaded, they can handle multiple operations executing in the background. When one of these operations completes, the kernel tells Node.js so that the appropriate callback may be added to the poll queue to eventually be executed.

> Event Loop Explained

The Node.js Event Loop has the following six phases, which are repeated for as long as the application still has code that needs to be executed:

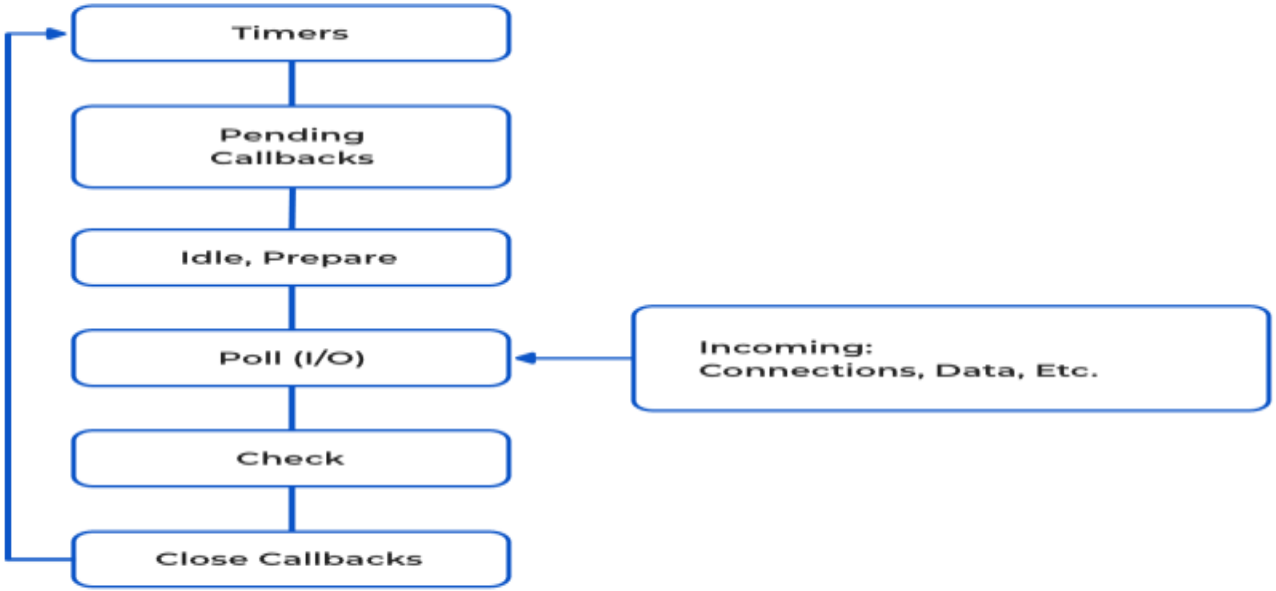
1. **Timers**
2. **Pending Callbacks**
3. **Idle, Prepare**
4. **Poll**
5. **Check**

6. Close Callbacks

Phases of an Event Loop

Each phase has a FIFO (First In First Out) queue of callbacks to execute. When the event loop enters a given phase, it will perform any operations specific to that phase, then execute callbacks in that phase's queue until the queue has been exhausted or the maximum number of callbacks has been executed. When the queue has been exhausted or the callback limit is reached, the event loop will move to the next phase, and so on.

PHASES OF AN EVENT LOOP



> **Phases Overview**

1. **Timers:**

   A timer specifies the threshold after which a provided callback may be executed rather than the exact time a person wants it to be executed. This phase executes callbacks scheduled by setTimeout() and setInterval(). setTimeout(,0) executes after all the current functions in the present queue get executed. SetImmediate() is similar but it doesn't use a queue of functions. It checks the queue of I/O event handlers. If all I/O events in the current snapshot are processed, it executes the callback.

2. Pending Callbacks:

Executes I/O callbacks deferred to the next loop iteration. This phase executes callbacks for some system operations such as types of TCP errors. For example, if a TCP socket receives ECONNREFUSED when attempting to connect, this will be queued to execute in the pending callbacks phase.

3. Idle, Prepare:

Only used internally. This is a housekeeping phase. During this phase, the Event Loop performs internal operations of any callbacks.

4. Poll:

The poll phase has two main functions:

a. Calculating how long it should block and poll for I/O, then
b. Processing events in the poll queue.

Retrieves new I/O events and executes I/O-related callbacks. Executes almost all with the exception of close callbacks, the ones scheduled by timers, and setImmediate()).

5. Check:

This phase allows a person to execute callbacks immediately after the poll phase has been completed. If the poll phase becomes idle and scripts have been queued with setImmediate(), the event loop may continue to the check phase rather than waiting.

6. Close Callbacks:

This phase executes the callbacks of all close events. For example, a close event of web socket callback, or when process.exit() is called. This is when the Event Loop is wrapping up one cycle and is ready to move to the next one. It is primarily used to clean the state of the application.

Conclusion

Understanding the event loop is a vital part of using Node.js, whether we are trying to get more insights about this technology, learn how to improve its performance, or find a new, interesting reason to learn a new tool. Hope this blog helped you in exploring this concept.

THANK YOU