Katherine Mazariegos

20180513

**Node.js**

NodeJS es un código abierto de JavaScript que está diseñado para generar aplicaciones web de forma altamente optimizada. Node.js es una forma de ejecutar JavaScript en el servidor, es un entorno JavaScript del lado del servidor, basado en eventos. Node ejecuta JavaScript utilizando el motor V8, desarrollado por Google para uso de su navegador Chrome. Este motor proporciona un entorno de ejecución del lado del servidor que compila y ejecuta JavaScript a velocidades increíbles. El aumento de velocidad es importante debido a que V8 compila Javascript en código de máquina nativo, en lugar de interpretarlo o ejecutarlo como bytecode. Node es de código abierto, y se ejecuta en Mac OS X, Windows y Linux. Uno de sus objetivos es la escalabilidad de conexiones, grandes números de conexiones.

¿Qué problema resuelve Node?

Proporciona una manera fácil para construir programas de red escalables. El problema con los programas de servidores actuales es que cada conexión genera un nuevo hilo que viene acompañado de 2 MB de memoria, si el sistema tiene 8 GB de RAM, da un número de conexiones de 4000 usuarios. Conforme la base de clientes va creciendo y se desea que la aplicación soporte mas usuarios se debe de agregar más servidores, lo cual genera costos.

Node resuelve este problema, pues su manera de realizar la conexión con el servidor es distinta. No genera un nuevo hilo de OS para cada conexión, ni memoria, sino que cada conexión dispara una ejecución de evento dentro del proceso del motor de Node. Tampoco se quedará en un punto muerto, porque no se permiten bloqueos y porque no se bloquea directamente para llamados E/S. Node afirma que un servidor que lo ejecute puede soportar decenas de miles de conexiones concurrentes.

¿Cómo funciona Node¿

Node ejecuta V8 JavaScript. El motor V8 JavaScript es el motor JavaScript subyacente que Google usa con su navegador Chrome. Este motor interpreta el código y lo ejecuta.

Programación orientada por eventos

Node utiliza lo que se conoce como modelo de programación orientado por eventos. Si ser realiza una conexión, es un evento, se reciben datos a través de la conexión es otro evento, se dejan de recibir datos por la conexión, otro evento.

JavaScript es un lenguaje para programación orientada por eventos, porque permite funciones y cierres anónimos y la sintaxis es similar. Las funciones de devolución de llamado que se llaman cuando ocurre un evento pueden escribirse en el mismo punto que se captura el evento. Es fácil de codificar, fácil de mantener. No hay infraestructuras complicadas Orientadas a Objeto, no hay interfaces, no hay potencial para sobre arquitectura de nada. Simplemente esperar por un evento y escribir una función de devolución de llamado.

Node, ¿para qué sirve?

Está diseñado para situaciones en que se espera una gran cantidad de tráfico y donde la lógica del lado del servidor y el procesamiento requeridos, no sean necesariamente grandes antes de responder al cliente. Ejemplos en lo que se haría buen uso de node:

* Una API RESTful: Un servicio Web que proporcione una API RESTful toma algunos parámetros, los interpreta, arma una respuesta y descarga esa respuesta (usualmente una cantidad relativamente pequeña de texto) de vuelta al usuario. Esta es una situación ideal para Node, porque puede construirse para que maneje decenas de miles de conexiones. Tampoco requiere una gran cantidad de lógica y básicamente sólo busca valores de una base de datos y los reúne como una respuesta. Como la respuesta es una pequeña cantidad de texto y la solicitud entrante es una pequeña cantidad de texto, el volumen de tráfico no es alto, y una máquina probablemente puede manejar las demandas de API de incluso la API de la más ocupada de las empresas.
* Fila de Twitter: Twitter recibe tweets y los escribe en una base de datos. Hay miles de tweets llegando cada segundo. y la base de datos posiblemente no puede seguir el ritmo del número de escrituras necesarias durante los horarios pico de uso. Node se convierte en una pieza clave de la solución a este problema. Como hemos visto, Node puede manejar decenas de miles de tweets entrantes. Luego puede escribirlos rápida/fácilmente en un mecanismo de cola en memoria (memcached, por ejemplo), desde donde otro proceso separado puede escribirlos en la base de datos. El rol de Node en esto es reunir rápidamente el tweet y pasar esta información hacia otro proceso responsable de escribirlo.
* Estadísticas de videojuegos: El juego Call of Duty on-line, algunas cosas le habrán llamado la atención inmediatamente cuando observó las estadísticas del juego, principalmente el hecho de que deben estar rastreando toneladas de información sobre el juego para poder producir tal nivel de estadísticas. Luego, multiplique esto por los millones de personas que lo juegan en cualquier momento, y tendrá una idea de la inmensa cantidad de información que se genera con bastante rapidez. Node es una buena solución para este escenario, porque puede capturar los datos que están generando los juegos, hacer un mínimo de consolidación con ellos y luego ponerlos en una fila para escribirlos en una base de datos.

Módulos Node

Se puede expandir la función de Node instalando módulos, estos módulos pueden mejorar en gran medida el producto. Algunos módulos podrían ser: uno para escribir páginas creadas dinámicamente como PHP, uno para facilitar el trabajo con MySQL, un módulo para ayuda con WebSockets y un módulo para asistir el análisis de texto y de parámetros.

Node presenta el Node Package Module, que es una forma integrada de instalar y administrar los módulos Node que esté usando. Este maneja automáticamente dependencias, de manera que usted puede estar seguro(a) de que cualquier módulo que usted desee instalar se instalará correctamente con todas sus partes necesarias. También sirve como una forma para publicar sus propios módulos en la comunidad Node, si usted opta por vincularse y escribir su propio módulo.

Uno de los puntos fuertes de nodeJS es su Node Package Manager (un los paquetes NPM). Una especie de gestor que da acceso a un conjunto de librerías muy extenso que además son gratuitas y generadas a partir de la colaboración de los usuarios de su comunidad. Algunos de los paquetes más conocidos que podrás encontrar en su plataforma son React, Gulp, Browserify, Grunt-cli, Bower, Grunt, Express, Cordova o Forever.

¿Qué es evento loop?

La cantidad de solicitudes así como los procesos entrantes y salientes se convierten en uno de los factores limitantes y Node.js ha sido concebido para optimizar este hándicap. La propuesta de node se basa en el tratamiento de forma unificada a partir de un único hilo complementado con un **bucle de eventos -Event Loop-** de tipo asíncrono. De este modo las peticiones que se vayan haciendo reciben un tratamiento en forma de eventos y pertenecen a este único bucle. Node ofrece una alta velocidad de procesamiento, pero además de esto la verdadera magia de Node.js es lo que se llama **bucle de eventos**. Pues para escalar grande volúmenes de clientes, todas las operaciones intensivas I/O en Node.js se llevan a cabo de forma asincrónica. El enfoque tradicional para generar código asíncrono es engorroso y crea un espacio en memoria no trivial para un gran número de clientes(cada cliente genera un hilo, y el uso de memoria de cada uno se suma). Para evitar esta ineficiencia,así como la dificultad conocida de las aplicaciones basadas en hilos, (programming threaded applications), Node.js mantiene un event loop que gestiona todas las operaciones asíncronas.

Cuando una aplicación Node.js necesita realizar una operación de bloqueo (operaciones I/O como trabajo con archivos …etc) envía una tarea asíncrona al event loop, junto con un **callback**(definido después), y luego continúa.

Operaciones asíncronas

Un servidor se encarga de ejecutar diferentes tareas para facilitar la comunicación con los diferentes clientes. Estas tareas se aglutinan bajo las siglas I/O que hacen referencia a aquellas que están destinadas a la entrada (input) y a la salida (output) de información. En los lenguajes clásicos de programación como Java este tipo de tareas se desarrollaban de una forma sincrónica. ¿Qué quiere decir esto? Que todas se ejecutaban de una forma lineal por lo que hasta que no se llevaba a cabo un proceso no se podía ejecutar el siguiente. Esto trae bastantes inconvenientes como por ejemplo el alargamiento innecesario de los procesos de trabajo y la tendencia a que se produzcan bloqueos. Sin embargo NodeJS emplea un I/O de tipo asíncrono. Esto significa que todas las tareas que se desarrollan por el servidor se hacen de forma paralela por lo que pueden efectuarse de forma simultánea y sin que se produzca ningún tipo de bloqueo en el flujo de trabajo. El resultado es una ventaja competitiva considerable que proporciona a la arquitectura de las aplicaciones web una mayor potencia y velocidad de procesamiento.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Node, trabaja con un solo hilo. Esto significa que solo ejecutará un proceso a la vez.

Como desarrolladores de Javascript debemos tener mucho cuidado al programar de no bloquear este único hilo. Aquí entra la parte de Blocking y Non-Blocking.

**Blocking**

Es básicamente esperar a que un proceso termine para después continuar con la ejecución del programa. Esto se hace usando operaciones síncronas, ciclos infinitos, etc.

**Non-Blocking**

Permite que tu programa siga corriendo sin la obligación de esperar a que un proceso termine su ejecución. Esto se lleva a cabo usando operaciones asíncronas.

**¿Pero qué son las operaciones síncronas y asíncronas?**

Las operaciones síncronas nos obligan a esperar a que la tarea termine por completo su ejecución. En la librería de nodejs, muchas de las operaciones cuentan con métodos síncronos y asíncronos a la vez.

Las operaciones asíncronas nos dan la capacidad de diferir una tarea para seguir ejecutando las demás. Es decir, el programa se encuentra con una operación que va a llevar tiempo en completarse, entonces deja que esta corra y continua con lo demás. Una vez se complete la operación en espera, la ejecuta.

**¿Qué es call back?**

Los callbacks son funciones que se pasan como parámetro de otra función. Son parte esencial de las operaciones asíncronas.

Por ejemplo, al escribir una función que se encarga de leer un archivo del disco duro, puedo pasarle una función o mejor llamado callback, el cual va a ser llamado una vez termine de leer el archivo y va a ejecutar lo que esté dentro de él. Por ello, el nombre callback.

**¿Qué es Evento Loop?**

El event loop es el que se encarga de implementar las operaciones asíncronas o el non-blocking. El event loop corre en el único hilo que existe en Node y como mencionamos anteriormente, al bloquear el único hilo de node, estamos bloqueando el event loop.

<<Libuv (una librería escrita en C), es el que permite que el event loop funcione y todo el comportamiento asíncrono en Node.>>

**Call Stack**

Cada vez que una función va a ser ejecutada pasa por el call stack. Como ya sabemos, al trabajar con operaciones asíncronas, estas poseen callbacks, que se ejecutarán una vez el proceso de la operación haya terminado y que se irán añadiendo al callback queue.

**Callback Queue**

Aquí se agregan los callback o funciones que se ejecutan una vez las operaciones asíncronas hayan terminado. Se utiliza el método FIFO (first input, first output), traducido, primero en entrar, primero en salir.

El event loop es el que se encarga de revisar que el call stack este vacío para añadir lo que está dentro del callback queue y ejecutarlo.

Referencias

Hernandez, F. (2018, diciembre 22). Nodejs y el Event Loop. Recuperado el 9 de octubre de 2019, de Medium website: <https://medium.com/@_ferh97/nodejs-y-el-event-loop-21b33fea6b03>

Node.js: ¿Qué es y para que sirve NodeJS? (2015, septiembre 30). Recuperado el 9 de octubre de 2019, de Desarrollo web Alicante y Valencia | Marketing Online website: <https://www.netconsulting.es/blog/nodejs/>

¿Simplemente qué es Node.js? (2011, junio 14). Recuperado el 9 de octubre de 2019, de <http://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/index.html>