

Smailyn Burgos.

NODE JS

CONCEPTO DE NODE JS

Node.js es un entorno de tiempo de ejecución de JavaScript que se utiliza para crear aplicaciones escalables del lado del servidor y de red a través de servidores privados virtuales. Ofrece operaciones de entrada/salida (E/S) no bloqueantes y está construido según una arquitectura asincrónica basada en eventos para ayudar a los desarrolladores a crear diversos proyectos de forma eficiente y sencilla.

ORIGENES DE NODE.JS

Node.js fue creado por Ryan Dahl y se lanzó por primera vez en 2009. Dahl desarrolló Node.js con la intención de proporcionar un entorno de ejecución de JavaScript en el lado del servidor que fuera rápido y eficiente. Estaba motivado por la necesidad de un modelo de programación no bloqueante para manejar operaciones de entrada/salida en tiempo real, como las aplicaciones web en tiempo real y las transmisiones de datos. Node.js ha sido impulsado por una comunidad activa de desarrolladores y ha crecido para convertirse en una herramienta popular para construir aplicaciones web escalables y en tiempo real.

FILOSOFÍA DE NODE.JS

1. **Eficiencia y rendimiento:** Optimización para ejecución rápida y manejo eficiente de operaciones de entrada/salida.
2. **Unificación de JavaScript:** Uso de JavaScript tanto en el lado del cliente como en el servidor para simplificar el desarrollo.
3. **Modularidad y reutilización:** Promoción de la creación de código modular y su reutilización a través de npm.
4. **Colaboración comunitaria:** Impulso de una comunidad activa y colaborativa para enriquecer el ecosistema de Node.js.
5. **Escalabilidad:** Diseño pensado en la escalabilidad para manejar aplicaciones web que crecen con las necesidades del negocio.

ARQUITECTURA DEL NODE.JS

La arquitectura de Node.js se basa en un modelo de ejecución asíncrona y basado en eventos que aprovecha el motor V8 de Google Chrome para ejecutar JavaScript de manera eficiente en el lado del servidor. Utiliza la biblioteca Libuv para gestionar operaciones de entrada/salida de manera no bloqueante, lo que permite manejar múltiples solicitudes concurrentes de forma eficiente. Node.js proporciona una amplia gama de API y un sistema de módulos para facilitar el desarrollo de aplicaciones escalables y modulares, y npm es utilizado como gestor de paquetes para instalar y gestionar dependencias fácilmente.

FUNCIONAMIENTO DE NODE.JS

Node.js funciona ejecutando código JavaScript en el lado del servidor mediante un modelo de ejecución asíncrona y basado en eventos. Utiliza el motor V8 de Google Chrome para compilar y ejecutar el código de manera eficiente. La biblioteca Libuv maneja las operaciones de entrada/salida de manera asíncrona y no bloqueante, permitiendo que Node.js maneje múltiples solicitudes simultáneamente sin bloquear el hilo principal de ejecución. Un bucle de eventos gestiona la secuencia de eventos y las devoluciones de llamada asociadas, asegurando un procesamiento eficiente de las operaciones asíncronas. Node.js proporciona una amplia gama de API y un sistema de módulos para facilitar el desarrollo de aplicaciones escalables y modulares, lo que lo convierte en una opción popular para construir aplicaciones web y de red.

OBJETO GLOBAL DE NODE.JS

El objeto global es '**global**', que proporciona un ámbito global para todas las variables y funciones en un programa. Es similar al objeto '**window**' en el navegador, pero está disponible en el entorno de Node.js. Todas las variables y funciones que no están explícitamente declaradas dentro de un ámbito local se adjuntan al objeto global. Esto significa que se puede acceder a estas variables y funciones desde cualquier parte del código. Sin embargo, es importante tener cuidado con el uso excesivo de variables y funciones globales, ya que puede conducir a problemas de legibilidad y mantenimiento del código. Es recomendable modularizar el código y limitar el alcance de las variables y funciones según sea necesario para una mejor organización y evitando posibles conflictos.

MODULOS DE NODE JS

Los módulos en Node.js son unidades de código reutilizable que encapsulan funcionalidades específicas. Pueden ser módulos integrados en Node.js, como fs o http, que no requieren instalación adicional, módulos personalizados creados por los desarrolladores para sus propias aplicaciones, o módulos de terceros disponibles en el registro npm. Estos módulos se pueden importar en las aplicaciones utilizando la función require() o, en versiones más recientes de Node.js, la sintaxis de módulos de ECMAScript (ESM) con import y export. Los módulos en Node.js siguen la especificación CommonJS para la inclusión y exportación de módulos, lo que permite una organización eficiente y la reutilización de código en el desarrollo de aplicaciones.

Algunos ejemplos de módulos de Node.js son:

fs: Permite interactuar con el sistema de archivos.

http: Utiliza sus funciones para crear un servidor y muchas más operaciones.

path: Permite analizar rutas de archivos del sistema de una forma consistente en diferentes plataformas.

querystring: Analiza los datos del query string

MODULE WRAPPER FUNCTION

La "Función de Envoltura de Módulo" en Node.js encapsula el código de cada archivo de módulo dentro de una función de ámbito, lo que permite la exportación de funcionalidades a otros módulos, la inclusión de módulos externos con **require**, y proporciona información útil sobre el módulo actual a través de parámetros como **exports**, **require**, **module**, **__filename**, y **__dirname**. Este enfoque garantiza la modularidad, evita conflictos de nombres y mantiene la privacidad de las variables y funciones dentro de cada módulo en las aplicaciones de Node.js.

PATH MODULE

El módulo **path** en Node.js proporciona utilidades para trabajar con rutas de archivos y directorios de manera consistente entre diferentes sistemas operativos

- **path.join()**: Concatena segmentos de ruta en una sola ruta. Esta función se encarga de manejar las diferencias en las barras diagonales y los caracteres de separación de rutas entre los sistemas operativos.
- **path.resolve()**: Resuelve una ruta absoluta a partir de rutas relativas o absolutas.
- **path.basename()**: Devuelve el último componente de una ruta.
- **path.dirname()**: Devuelve el directorio de una ruta.
- **path.extname()**: Devuelve la extensión de un archivo en una ruta.
- **path.parse()**: Parsea una ruta en un objeto con sus componentes (**root, dir, base, ext, name**).

OS MODULE

El módulo os en Node.js proporciona métodos para interactuar con el sistema operativo en el que se está ejecutando Node.js.

- **os.platform()**: Devuelve el sistema operativo en el que se está ejecutando Node.js, como "darwin" para macOS, "win32" para Windows, o "linux" para sistemas basados en Linux.
- **os.arch()**: Devuelve la arquitectura del procesador del sistema, como "x64" o "arm".
- **os.cpus()**: Devuelve un array de objetos que contienen información sobre los núcleos de la CPU del sistema, como el modelo, velocidad, y tiempos de uso.
- **os.totalmem()**: Devuelve la cantidad total de memoria del sistema en bytes.
- **os.freemem()**: Devuelve la cantidad de memoria libre en el sistema en bytes.
- **os.hostname()**: Devuelve el nombre del host del sistema.
- **os.userInfo()**: Devuelve información sobre el usuario actual del sistema, como el nombre de usuario, el UID, el directorio de inicio, y más.

FIRCE SYSTEM MODULE

El módulo "fs" en Node.js, que significa "sistema de archivos", proporciona funcionalidades para interactuar con el sistema de archivos del sistema operativo. Con este módulo, puedes realizar tareas como leer y escribir archivos, crear y eliminar directorios, obtener información sobre archivos y directorios, entre otras operaciones relacionadas con el sistema de archivos. Es esencial para el manejo de archivos en aplicaciones Node.js.

EVENT MODULE

El módulo "events" en Node.js proporciona una manera de implementar el patrón de diseño de observador en tu código. Este módulo permite la comunicación entre objetos en un estilo de publicación/suscripción, lo que significa que un objeto (el "emisor" o "emitter") puede emitir eventos y otros objetos (los "escuchadores" o "listeners") pueden suscribirse para manejar esos eventos cuando se emiten.

ARGUMENTOS DE EVENTOS

En Node.js, los manejadores de eventos pueden recibir argumentos cuando se emiten. Los argumentos se pueden pasar como parámetros al método emit() y se pueden capturar en los manejadores de eventos registrados para ese evento.