**Event Loop trong NodeJS**

Node.js là xử lý đơn luồng, nên nó sẽ chỉ làm một việc một lúc.

Eventloop là một phần quan trọng của hệ thống runtime của Node.js, nó xử lý các sự kiện và callback trong ứng dụng Node.js của bạn. Khi bạn chạy một chương trình Node.js, nó sẽ tạo ra một event loop và một thread để xử lý các sự kiện và callback.

Event loop là một vòng lặp vô tận để kiểm tra các sự kiện và thực hiện các callback tương ứng. Nó cũng điều khiển việc thực hiện các tác vụ khác trong chương trình Node.js, như đọc và ghi từ các I/O, gửi và nhận dữ liệu từ mạng, v.v.

Có hai loại sự kiện mà event loop xử lý: sự kiện đồng bộ và sự kiện bất đồng bộ. Sự kiện đồng bộ được xử lý ngay lập tức trong vòng lặp event loop, trong khi sự kiện bất đồng bộ được đưa vào một hàng đợi và xử lý sau khi các sự kiện đồng bộ đã được xử lý xong.

Đây là cách mà Node.js có thể xử lý hàng ngàn yêu cầu I/O mà không cần sử dụng đến nhiều luồng.

Event loop bao gồm các phần tử sau:

* Call stack: Là một ngăn xếp (stack) chứa các hàm được gọi trong chương trình. Hàm nào được gọi đầu tiên sẽ nằm dưới cùng, và hàm nào được gọi sau cùng sẽ nằm trên cùng. Khi một hàm kết thúc, nó sẽ được loại bỏ khỏi call stack.
* Callback queue: Là một hàng đợi (queue) chứa các hàm gọi lại của các sự kiện không đồng bộ. Các hàm gọi lại này sẽ được đưa vào hàng đợi theo thứ tự FIFO (first in, first out), tức là hàm nào được đưa vào trước sẽ được lấy ra trước. Khi call stack trống, event loop sẽ lấy hàm gọi lại đầu tiên trong callback queue và đưa vào call stack để thực thi.
* Event loop: Là một vòng lặp liên tục, trong đó Node.js kiểm tra xem call stack có trống hay không, và nếu trống thì lấy hàm gọi lại từ callback queue để đưa vào call stack. Event loop cũng sẽ kiểm tra các sự kiện mới từ các nguồn khác nhau, và đưa các hàm gọi lại tương ứng vào callback queue.
* APIs: Là các giao diện lập trình ứng dụng (application programming interface) của Node.js hoặc của hệ điều hành, cho phép Node.js thực hiện các thao tác không đồng bộ với các tài nguyên bên ngoài, như mạng, tệp, cơ sở dữ liệu, v.v. Khi một thao tác không đồng bộ được gọi, Node.js sẽ gửi nó đến một API, và tiếp tục chạy các hàm khác trong call stack. Khi thao tác không đồng bộ hoàn thành, API sẽ tạo một sự kiện và đưa hàm gọi lại vào callback queue.

Quá trình hoạt động của Event Loop có các bước chính như sau:

* **Đọc các sự kiện từ Event Queue**: Event Loop đầu tiên đọc các sự kiện từ Event Queue. Đây là nơi mà các sự kiện và callback được đưa vào sau khi các thao tác I/O hoặc các hành động bất đồng bộ hoàn thành.
* **Xử lý và thực thi sự kiện**: Event Loop lấy các sự kiện từ Event Queue và thực thi chúng một cách tuần tự. Điều này bao gồm việc gọi các callback và xử lý các Promise resolutions.
* **Blocking vs Non-blocking operations**: Trong quá trình thực thi, Node.js có thể thực hiện các thao tác đồng bộ (blocking) và bất đồng bộ (non-blocking). Các thao tác bất đồng bộ không chặn Event Loop, trong khi các thao tác đồng bộ có thể làm chậm quá trình.
* **Event Loop tiếp tục lặp lại**: Sau khi thực hiện xong một sự kiện, Event Loop quay lại đọc và xử lý các sự kiện tiếp theo trong Event Queue. Quá trình này lặp đi lặp lại liên tục trong suốt thời gian Node.js chạy.

Event Loop của Node.js là một phần quan trọng của cách Node xử lý đồng thời nhiều yêu cầu I/O mà không cần sử dụng đa luồng. Điều này giúp cho Node.js rất hiệu quả trong các ứng dụng có nhiều thao tác I/O như server và các ứng dụng mạng.

**REPL**

REPL là viết tắt của Read, Eval, Print và Loop là một đặc tính của NodeJS cho phép lập trình viên viết code và chạy trực tiếp trên màn hình shell/console/terminal để debug, kiểm tra code mà không cần tạo ra bất cứ file hay folder nào.

Khi gõ code JavaScript lên màn hình shell, NodeJS sẽ thực hiện việc đọc thông tin (Read) và tự động lưu trữ trong bộ nhớ; tự động đánh giá cấu trúc dữ liệu và sự hợp lệ của các dòng lệnh (Eval); xử lý thực thi code sau đó in ra kết quả nếu có (Print) và hỗ trợ lặp lại các dòng lệnh trên để thực thi chương trình (Loop).

Lệnh node để vào REPL

**NPM trong NodeJS**

npm là công cụ quản lý thư viện và phiên bản JavaScript cho Node.js. NPM giúp các lập trình viên JavaScript có thể tạo lập, quản lý và chia sẻ các thư viện lập trình một cách dễ dàng. Với NPM, người dùng có thể cài đặt và sử dụng hàng trăm nghìn thư viện có sẵn, giúp tiết kiệm thời gian và tránh tình trạng phải viết lại tất cả các thành phần cơ bản trong dự án.

NPM cung cấp hai chức năng chính:

* Quản lý module: NPM cho phép người dùng tạo, cài đặt cũng như quản lý các module JavaScript trong dự án. Người dùng có thể tìm kiếm và cập nhật các module này một cách đơn giản từ kho lưu trữ NPM.
* Quản lý phiên bản: NPM hỗ trợ quản lý phiên bản của module trong dự án. Người dùng có thể dễ dàng xác định và cài đặt phiên bản cụ thể của module hoặc cho phép NPM tự động cập nhật phiên bản mới nhất.

npm init : Khởi tạo dự án Node.js

npm install : Cài đặt tất cả dependencies từ package.json

npm install <tên thư viện> : Cài đặt thư viện

npm uninstall <tên thư viện> : Gỡ bỏ thư viện

**Package.json**

package.json là một file chứa thông tin giúp bạn biết được để vận hành được project cần đến những modules nào. Có thể coi những modules chính là sức mạnh của nodejs với một bộ mã nguồn mở khổng lồ cũng như cộng đồng lớn mạnh đến nỗi cứ vài phút lại có một module mới.

File package.json được viết bằng json

File package.json được đặt ở thư mục gốc của project.

**Các thành phần của Package.json**

Package.json chứa rất nhiều thông tin, thường thì ta chỉ quan tâm đến vài thuộc tính chính.

name

Tên của project hoặc package, nên viết hoa cho thuộc tính name. Đây là thuộc tính bắt buộc. Ngoài ra bạn có thể public project của bạn, thì thuộc tính name này sẽ là package name, nên cái này phải là duy nhất nhé.

version

Phiên bản của project. Cách ghi version hiện nay được quy đinh bởi 1 ông nào đó tên Semantic Versioning. Ông quy định thế này, phiên bản phải gồm 3 phần MAJOR.MINOR.PATCH Theo nguyên văn, trong đó:

* MAJOR : Khi bạn thay đổi API không tương thích với phiên bản trước (có thể gây lỗi cho người đang dùng phiên bản cũ).
* MINOR : Khi bạn thêm tính năng mới nhưng vẫn tương thích với phiên bản cũ.
* PATCH : Khi bạn sửa lỗi (bug fix) và vẫn tương thích ngược với các phiên bản cũ.

description

Đoạn mô tả của project. Chú ý viết ngắn gọn xúc tích rõ ràng dễ hiểu.

author

Thông tin về tác giả.

dependencies

Cái này quan trọng đây. Trong project, bạn sẽ phải sử dụng rất nhiều gói, những gói này đã được viết sẵn.

Để làm việc này, bạn cần phải install gói đó bằng npm. Thuộc tính dependencies giúp npm biết được cần phải cài đặt những package nào.

**Module.exports**

Node.js sử dụng Module để đơn giản hóa việc tạo ra các ứng dụng phức tạp. Module là giống như các thư viện trong PHP, C, C#,… Mỗi module chứa một tập các hàm chức năng có liên quan đến một đối tượng của Module qua đó giúp việc viết và quản lý mã lệnh của chương trình được dễ dàng hơn. Một module có thể đơn giản là một hàm hay một đối tượng. Mỗi module thường được khai bảo ở một tập tin riêng rẽ.

Module là các đoạn code được đóng gói lại với nhau,... Code trong một Module thường là private – nghĩa là các hàm, biến được định nghĩa và truy cập bởi bên trong của Module. Để sử dụng các hàm hoặc biến ở bên ngoài Module bạn cần sử dụng exports.

**Require**

Node.js sử dụng CommonJS để triển khai hệ thống module và require là lệnh để yêu cầu sử dụng một module trong một file xác định. Chức năng cơ bản của require đó là nó sẽ đọc một file, thực thi và sau đó trả lại các đối tượng được exports.

**Cách hoạt động của require**

Khi gọi require('module\_name'), Node sẽ:

* 1. Tìm file tương ứng (theo đường dẫn hoặc theo node\_modules).
  2. Cache lại module sau lần require đầu tiên (tăng hiệu suất).
  3. Thực thi module (nếu chưa cache).
  4. Trả về giá trị từ module.exports

**Core Modules**

Trong Node.js, core modules (mô-đun lõi) là những mô-đun đã được tích hợp sẵn trong Node.js. Điều đó có nghĩa là không cần phải cài đặt thêm thông qua npm, và có thể sử dụng trực tiếp bằng require hoặc import. Các core modules giúp thực hiện nhiều chức năng quan trọng từ cấp hệ thống đến xử lý dữ liệu, tạo server, và quản lý sự kiện.

Dưới đây là một số core modules phổ biến:

fs (File System)Module này dùng để thao tác với hệ thống tập tin, bao gồm:

* Đọc file (fs.readFile)
* Ghi file (fs.writeFile)
* Tạo, sửa, xóa file hoặc thư mục
* Hỗ trợ cả callback và Promise (fs.promises)

http: Dùng để tạo web server cơ bản mà không cần Express. Hỗ trợ xử lý HTTP requests/responses một cách trực tiếp.

path: Dùng để xử lý đường dẫn file (normalize, join, resolve, dirname, v.v.)

os: Dùng để lấy thông tin hệ điều hành như RAM, CPU, hostname, user info, platform...

event: Dùng để tạo và quản lý các sự kiện tùy chỉnh. Cốt lõi cho Node.js (nhiều phần như http, stream, fs cũng dùng events).

Core modules là thành phần rất quan trọng trong Node.js, giúp bạn xây dựng ứng dụng mạnh mẽ mà không cần phụ thuộc vào thư viện bên ngoài. Việc hiểu và vận dụng tốt core modules sẽ giúp lập trình viên backend tối ưu hiệu suất, bảo trì dễ dàng và kiểm soát tốt hơn hệ thống.