|  |
| --- |
|  |
| **TRƯỜNG CAO ĐẲNG NGHỀ CÔNG NGHỆ CAO HÀ NỘI**  **---------------------------------------** |
|  |
|  |
| Description: Logo HHT 2 |
|  |
|  |
|  |
|  |
| **HƯỚNG DẪN CƠ BẢN NODE.JS** |
|  |
|  |
|  |

**Mục lục**

[**Giới thiệu** 3](#_Toc60253042)

[Node.js là gì? 3](#_Toc60253043)

[**Đặc điểm của Node.js** 3](#_Toc60253044)

[Với Node.js, bạn phải làm mọi thứ 4](#_Toc60253045)

[Các thành phần quan trọng trong Node.js 4](#_Toc60253046)

[Khi nào sử dụng Node.js 4](#_Toc60253047)

[Khi nào không nên dùng Node.js 5](#_Toc60253048)

[Tạo ứng dụng Node.js 5](#_Toc60253049)

[Tạo một request đến một Node.js Server 6](#_Toc60253050)

[**REPL là gì?** 6](#_Toc60253051)

[Bắt đầu với REPL trong Node.js 7](#_Toc60253052)

[Các biểu thức đơn giản trong Node.js 7](#_Toc60253053)

[Sử dụng biến trong Node.js 7](#_Toc60253054)

[Biểu thức trên nhiều dòng trong Node.js 7](#_Toc60253055)

[Biến có dấu gạch dưới trong Node.js 8](#_Toc60253056)

[Giới thiệu một số lệnh REPL trong Node.js 8](#_Toc60253057)

[Kết thúc REPL trong Node.js 9](#_Toc60253058)

[**NPM trong Node.js** 9](#_Toc60253059)

[NPM là gì? 9](#_Toc60253060)

[Cài đặt các Module bởi sử dụng npm trong Node.js 9](#_Toc60253061)

[Cài đặt Global và cài đặt local 10](#_Toc60253062)

[Sử dụng gói package.json 10](#_Toc60253063)

[Các thuộc tính của Package.json 10](#_Toc60253064)

[Gỡ cài đặt một module trong Node.js 11](#_Toc60253065)

[Cập nhật một Module trong Node.js 11](#_Toc60253066)

[ìm kiếm một Module trong Node.js 11](#_Toc60253067)

[Tạo một Module trong Node.js 11](#_Toc60253068)

[**Callback trong Node.js** 12](#_Toc60253069)

[**Event Loop trong Node.js** 15](#_Toc60253070)

[**Lớp Event Emitter trong Node.js** 18](#_Toc60253071)

[**Buffer trong Node.js** 21](#_Toc60253072)

[**TRUY VẤN TỆP TIN** 29](#_Toc60253073)

[**Utility Module trong Nodejs** 33](#_Toc60253074)

[Giá trị rrtype của dns Module trong Node.sj 44](#_Toc60253075)

[Một số Error Code của dns Module trong Node.js 45](#_Toc60253076)

[**Web Server** 48](#_Toc60253077)

[**Express Framework** 52](#_Toc60253078)

# **Giới thiệu**

Node.js là gì?

NodeJS là một nền tảng Server side được xây dựng dựa trên Javascript Engine (V8 Engine). Node.js được phát triển bởi Ryan Dahl năm 2009. NodeJs được định nghĩa như sau:

Node.js là một nền tảng (platform) dựa vào Chrome Javascript runtime để xây dựng các ứng dụng nhanh, có độ lớn. Node.js sử dụng các thành phần hướng sự kiện (event-driven), mô hình non-blocking I/O để tạo ra các ứng dụng nhẹ và hiệu quả cho các ứng dụng về dữ liệu thời gian thực chạy trên các thiết bị phân tán.

Node.js = Môi trường Runtime + Các thư viện Javascript

NodeJs là một platform mã nguồn mở, đa nền tảng cho phát triển các ứng dụng phía Server và các ứng dụng liên quan đến mạng. Ứng dụng Node.js được viết bằng Javascript và có thể chạy trong môi trường Node.js trên hệ điều hành Window, Linux...

Node.js cũng cung cấp cho chúng ta các module Javascript đa dạng, có thể đơn giản hóa sự phát triển của các ứng dụng web sử dụng Node.js với các phần mở rộng.

# **Đặc điểm của Node.js**

Dưới đây là vài đặc điểm quan trọng biến Node.js trở thành sự lựa chọn hàng đầu trong phát triển phần mềm:

* Không đồng bộ và hướng sự kiện (Event Driven): Tất các các APIs của thư viện Node.js đều không đồng bộ, nghĩa là không blocking (khóa). Nó rất cần thiết vì Node.js không bao giờ đợi một API trả về dự liệu. Server chuyển sang một API sau khi gọi nó và có cơ chế thông báo về sự kiện của Node.js giúp Server nhận được phản hồi từ các API gọi trước đó.
* Chạy rất nhanh: Dựa trên V8 Javascript Engine của Google Chrome, thư viện Node.js rất nhanh trong các quá trình thực hiện code.
* Các tiến trình đơn giản nhưng hiệu năng cao: Node.js sử dụng một mô hình luồng đơn (single thread) với các sự kiện lặp. Các cơ chế sự kiện giúp Server trả lại các phản hồi với một cách không khóa và tạo cho Server hiệu quả cao ngược lại với các cách truyền thống tạo ra một số lượng luồng hữu hạn để quản lý request. Nodejs sử dụng các chương trình đơn luồng và các chương trình này cung cấp các dịch vụ cho số lượng request nhiều hơn so với các Server truyền thống như Apache HTTP Server.
* Không đệm: Ứng dụng Node.js không lưu trữ các dữ liệu buffer.
* Có giấy phép: Node.js được phát hành dựa vào MIT License.

Với Node.js, bạn phải làm mọi thứ

Node.js chỉ là một môi trường – điều này có nghĩa bạn tự phải làm mọi thứ. Sẽ chẳng có bất kỳ máy chủ mặc định nào cả. Một đoạn script xử lý tất cả các kết nối với Client. Điều này làm giảm đáng kể số lượng tài nguyên được sử dụng trong ứng dụng.

Các thành phần quan trọng trong Node.js

Hình ảnh dưới đây mô tả các thành phần (component) quan trọng của Node.js mà chúng ta sẽ học ở các bài tiếp theo.



Khi nào sử dụng Node.js

Dưới đây là các lĩnh vực mà Node.js được sử dụng như là một sự lựa chọn hoàn hảo:

* Các ứng dụng về I/O
* Các ứng dựng về luồng dữ liệu
* Các ứng dụng về dữ liệu hướng đến thời gian thực
* Các ứng dụng dựa vào JSON APIs
* Các ứng dụng Single Page Application

Khi nào không nên dùng Node.js

Nó không nên sử dụng trong các ứng dụng đòi hỏi về CPU.

Ứng dụng hello world trong nodejs

Trước khi tạo ứng dụng "Hello world" trong Node.js, hãy cùng xem các phần chính của chương trình Node.js. Một chương trình Node.js bao gồm các phần quan trọng đưới đây:

* Các Module cần thiết: Chúng ta sử dụng require directive để tải một Node.js module.
* Tạo Server: Một Server lắng nghe yêu cầu từ phía client tương tự như các Server Apache HTTP.
* Đọc request và trả về các response: Server được tạo một các dễ dàng bên trên sẽ đọc các HTTP request bởi client từ các trình duyệt hay màn hình console để trả về các phản hồi.

## Tạo ứng dụng Node.js

Bước 1: Import các Module cần thiết

Chúng ta sử dụng require directive để tải các Module http và trả về các sự thể hiện đến biến http như sau:

|  |  |
| --- | --- |
|  | var http = require("http"); |

Bước 2: Tạo Server

Bước tiếp theo, chúng ta sẽ tạo http và gọi phương thức http.createServer() để tạo mới một Server và trả về một Server Instance và sau đó gắn kết nó trên cổng 8081. Truyền vào nó với các tham số request và response. Viết đoạn ví dụ sau đây về chương trình "Hello World".

http.createServer(function (request, response) {

response.writeHead(200, {'Content-Type': 'text/plain'});

response.end('Hello World\n');

}).listen(8080, "127.0.0.1");

Đoạn code này đủ cho việc tạo một HTTP Server để lắng nghe và đợi các phản hồi trên 8081 của máy local.

Bước 3: Kiểm tra Request & Response

Cùng đặt bước 1 và 2 với nhau trong 1 file với tên gọi main.js và bật HTTP Server như sau:

console.log('Server running at http://127.0.0.1:80801');

Start Server và xem kết quả như sau:

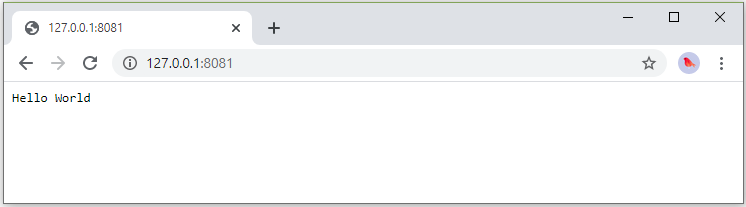
$ node main.js

Kiểm tra kết quả. Server đã được bật.

Server running at <http://127.0.0.1:8081/>

## Tạo một request đến một Node.js Server

Mở http://127.0.0.1:8080/ trong bất kỳ trình duyệt nào và xem kết quả:.



Chúc mừng bạn đã tạo thành công ứng dụng Node.js đầu tiên. Theo dõi tiếp các chương tiếp theo để hiểu sâu hơn về Node.js nhé.

# **REPL là gì?**

REPL là từ viết tắt của Read Eval Print Loop (Đọc - Đánh giá - In - Lặp) và nó biểu diễn môi trường máy tính như màn hình console trong Linux shell nơi bạn có thể gõ các dòng lệnh và hệ thống sẽ trả về các kết quả. Node.js cũng có môi trường REPL. Nó để thực hiện các tác vụ mong muốn:

* Read: Đọc các thông tin input của người dùng, chuyển đổi thành các dữ liệu Javascript và lưu trữ trong bộ nhớ.
* Eval: Đánh giá các cấu trúc dữ liệu này.
* Print: In các kết quả.
* Loop: Lặp các dòng lệnh đến khi người dùng gõ ctrl-c hai lần.

Đặc tính REPL của Node js rất hữu ích khi bạn sử dụng Node.js với mục đích debug code.

## Bắt đầu với REPL trong Node.js

REPL có thể bắt đầu bằng cách đơn giản trên màn hình shell/console mà không sử dụng bất cứ tham số nào như sau:

> node

Bạn sẽ thấy màn hình dòng nhắc lệnh REPL >. Tại đây, bạn có thể gõ bất cứ lệnh nào trong Node.js.

>

### Các biểu thức đơn giản trong Node.js

Dưới đây mình giới thiệu một số biểu thức đơn giản có thể được sử dụng tại dòng nhắc lệnh REPL trong Node.js:

$ node

> 1 + 3

4

> 1 + ( 2 \* 3 ) - 4

3

>

### Sử dụng biến trong Node.js

Bạn có thể sử dụng các biến để lưu trữ giá trị và sau đó in giá trị của biến giống như trong các đoạn mã truyền thống. Nếu không sử dụng từ khóa var, giá trị sẽ được lưu trữ trong biến và in ra. Trong khi đó nếu từ khóa var được sử dụng, giá trị được lưu trữ và có thể không được in. Bạn có thể in biến sử dụng console.log().

$ node

> x = 10

10

> var y = 10

undefined

> x + y

20

> console.log("Hello World")

Hello World

undefined

### Biểu thức trên nhiều dòng trong Node.js

Node REPL hỗ trợ dạng biểu thức nằm trên nhiều dòng tương tự như trong Javascript. Cùng kiểm tra biểu thức do-while trong hành động sau đây:

Ví dụ:

$ node

> var x = 0

undefined

> do {

... x++;

... console.log("x: " + x);

... }

while ( x < 5 );

x: 1

x: 2

x: 3

x: 4

x: 5

undefined

>

Các dấu gạch chấm ... hiển thị tự động khi bạn nhấn phím Enter sau dấu ngoặc mở. Node.js sẽ tự động kiểm tra xem biểu thức có được tiếp tục nữa hay không.

### Biến có dấu gạch dưới trong Node.js

Bạn có thể sử dụng dấu gạch dưới \_ để nhận kết quả cuối cùng của phép tính:

Output:

$ node

> var x = 10

undefined

> var y = 20

undefined

> x + y

30

> var sum = \_

undefined

> console.log(sum)

30

undefined

>

## Giới thiệu một số lệnh REPL trong Node.js

* ctrl + c - Kết thúc lệnh hiện tại.
* Nhấn ctrl + c hai lần - Kết thúc Node REPL.
* ctrl + d - Kết thúc Node REPL.
* Các phím mũi tên Up/Down - Xem lịch sử các lệnh, kiểm tra lệnh trước đó và có thể sửa đổi các lệnh đã soạn trước đó.
* Phím tab - Danh sách các lệnh hiện tại.
* .help - Danh sách tất cả các lệnh.
* .break - Thoát khỏi một biểu thức nằm trên nhiều dòng (ví dụ: do-while).
* .clear - Thoát khỏi một biểu thức nằm trên nhiều dòng
* .save ten\_file - Lưu phiên làm việc của Node.js REPL hiện tại vào một ten\_file nào đó.
* .load ten\_file - Tải nội dung của ten\_file của phiên làm việc Node.js REPL hiện tại.

## Kết thúc REPL trong Node.js

Như đã đề cập bên trên, bạn cần nhấn ctr + c 2 lần để kết thúc Node.js REPL.

Output:

> node

>

(^C again to quit)

>

# **NPM trong Node.js**

## NPM là gì?

NPM viết tắt của Node package manager là một công cụ tạo và quản lý các thư viện lập trình Javascript cho Node.js.

Trong các dự án với javascript thì chắc hẳn các bạn sẽ cần nhiều thư viện của javascript. Điển hình như jquery, bootstrap, express, vue, react, angular, ... thì việc quản lý các thư viện này như thế nào, dùng phiên bản bao nhiêu, cần require thêm những module gì. Những công việc này nếu làm thủ công thì quả là cực, đặc biệt là những dự án lớn.

NPM được cài đặt cùng với Node.js từ phiên bản v0.6.3. Để xác nhận việc này, mở cửa sổ dòng lệnh, gõ những lệnh sau và xem kết quả:

Kiểm tra phiên bản npm: npm --version

$ npm --version

2.7.1

Có thể dễ dàng cập nhật bản mới nhất. Sử dụng lệnh sau đây dưới quyền Administrator: npm install npm –g

## Cài đặt các Module bởi sử dụng npm trong Node.js

Dưới đây là cú pháp đơn giản để cài đặt bất kỳ Node.js Module nào:

> npm install [Module Name]

Ví dụ: lệnh để cài đặt một module rất phổ biến trong Node.js, đó là **express** Module:

> npm install express

Để sử dụng module express trong file js của bạn, sử dụng cú pháp sau đây:

var express = require('express');

## Cài đặt Global và cài đặt local

Mặc định, npm cài đặt bất kì phụ thuộc nào trên máy tính local. Ở đây, module này hướng đến các gói cài đặt trong thư mục node\_modules trong thư mục ứng dụng của Node. Gói local này có thể triển khai dựa vào phương thức require(). Ví dụ, khi bạn cài đặt Module express, bạn tạo thư mục node\_modules trong thư mục hiện tại nơi bạn có thể cài đặt Module này.

Cách khác, bạn có thể sử dụng lệnh npm ls để liệt kê các thư mục Module cài đặt.

Gói cài đặt toàn cục được lưu trữ trong các file hệ thống. Các phụ thuộc có thể được sử dụng với các hàm CLI (Command Line Interface) của node.js nhưng không thể import sử dụng require() của ứng dụng Node một cách trực tiếp.

Bây giờ cùng thử cài đặt Module express sử dụng cài đặt toàn cục như sau: npm install express -g

> npm install express -g

Điều này sẽ cho kết quả tương tự nhưng Module được cài đặt toàn cục. Ở đây, dòng đầu tiên nói với chúng ta về Module version và nơi chứa các bộ cài khi bắt đầu.

Bạn cũng có thể sử dụng lệnh sau đây để kiểm tra các phiên bản cài đặt toàn cục: npm ls -g

> npm ls -g

## Sử dụng gói package.json

Package.json hiển thị trong thư mục root của bất cứ ứng dụng Node nào và được sử dụng để định nghĩa thuộc tính của gói. Tập tin package.json của gói express hiển thị trong **node\_modules/express/**:

## Các thuộc tính của Package.json

* **name** - tên của package
* **version** - phiên bản của package
* **description** - phần mô tả của package
* **homepage** - trang chủ của package
* **author** - tác giả của package
* **contributors** - tên người đóng góp cho package
* **dependencies** - Danh sách các gói phụ thuộc, tự động được cài theo.
* **repository** - loại repository và url của package
* **main** - điểm vào của package
* **keywords** - các từ khóa

Gỡ cài đặt một module trong Node.js

Sử dụng các dòng lệnh dưới đây để gỡ một Module trong Node.js

> npm uninstall express

Khi gỡ cài đặt gói này, bạn có thể xác nhận nội dung trong thư mục /node\_modules/ bằng cách sử dụng lệnh sau đây:

> npm ls

Cập nhật một Module trong Node.js

Cập nhật gói package.json và thay đổi phiên bản của các ràng buộc mà có thể cập nhật sử dụng lệnh sau đây:

> npm update express

ìm kiếm một Module trong Node.js

Sử dụng npm, bạn có thể tìm kiếm một Module như sau:

> npm search express

Tạo một Module trong Node.js

Quá trình tạo Module yêu cầu gói package.json được tạo ra.

> npm init

This utility will walk you through creating a package.json file.

It only covers the most common items, and tries to guess sane defaults.See 'npm help json' for definitive documentation on these fields

and exactly what they do.Use 'npm install --save' afterwards to install a package and

save it as a dependency in the package.json file.Press ^C at any time to quit.

name: (webmaster)

Bạn cần cung cấp tất cả các thông tin được yêu cầu trong việc sử dụng Module. Bạn có thể sử dụng các bước bên trên trong file package.json để hiểu về ý nghĩa của các thông tin yêu cầu. Khi gói package.json được tạo ra, sử dụng các lệnh dưới đây đăng ký với trang npm bởi sử dụng địa chỉ email hợp lệ.

> npm adduser

Username: mcmohd

Password:

Email: (this IS public) mcmohd@gmail.com

Bây giờ publish Module của bạn:

> npm publish

Nếu mọi thứ ok, nó sẽ được publish và thư mục này sẽ được truy cập bởi những người khác sử dụng Module Node.js

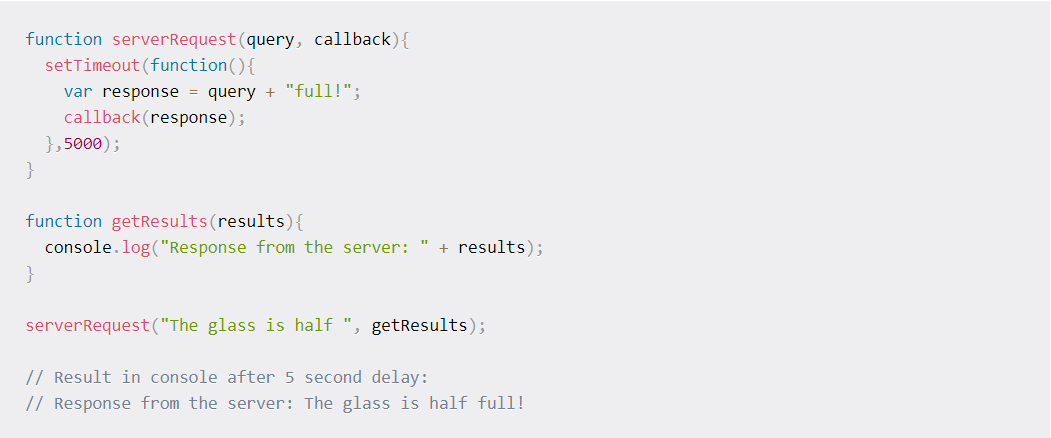
**Callback trong Node.js**

1. Callback là gì?

* Callback có tính chất không đồng bộ. Một hàm callback được gọi khi hoàn thành một nhiệm vụ cụ thể. Tất cả các API của Node đều được viết theo các cách của hàm callback. Xem ví dụ để hiểu callback trong node.js
* Ví dụ, với một hàm đọc file bắt đầu với việc đọc file và hệ thống tiếp tục thực thi lệnh tiếp theo. Khi phần đọc file được hoàn thành, nó sẽ gọi về một hàm callback, với nội dung của file là tham số. Do đó sẽ không có blocking hoặc chờ khi đọc File mà hệ thống vẫn tiếp tục thực thi các lệnh tiếp theo. Nó làm cho Node.js có hiệu năng cao hơn, như có số lượng request cao hơn mà không cần phải chờ kết quả trả về.

1. Tại sao lại cần Callback?

* Phần lớn thời gian chúng ta đang tạo ra các chương trình và ứng dụng hoạt động một cách **đồng bộ** . Nói cách khác, một số hoạt động của chúng tôi chỉ được bắt đầu sau khi những hoạt động trước đó đã hoàn thành. Thông thường, khi chúng tôi yêu cầu dữ liệu từ các nguồn khác, chẳng hạn như API bên ngoài, chúng tôi không phải lúc nào cũng biết khi nào dữ liệu của mình sẽ được phục vụ trở lại. Trong những trường hợp này, chúng tôi muốn đợi phản hồi, nhưng không phải lúc nào chúng tôi cũng muốn toàn bộ ứng dụng của mình tạm dừng trong khi dữ liệu của chúng tôi đang được tìm nạp. Những tình huống này là nơi các hàm gọi lại có ích.
* Hãy xem một ví dụ mô phỏng một yêu cầu đến máy chủ:



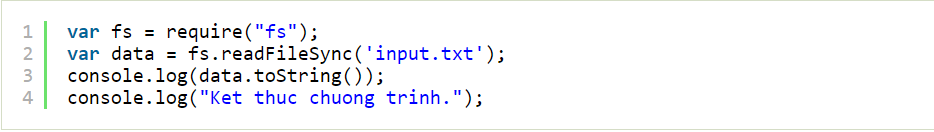
* Trong ví dụ trên, chúng tôi thực hiện một yêu cầu giả tới một máy chủ. Sau 5 giây trôi qua, phản hồi được sửa đổi và sau đó hàm gọi lại của chúng tôi **getResults** được thực thi. Để xem điều này đang hoạt động, bạn có thể sao chép / dán đoạn mã trên vào công cụ dành cho nhà phát triển của trình duyệt và thực thi nó.
* Ngoài ra, nếu bạn đã quen thuộc với **setTimeout**, thì bạn đã sử dụng các hàm gọi lại. Đối số hàm ẩn danh được truyền vào **setTimeout** lệnh gọi hàm của ví dụ trên cũng là một lệnh gọi lại! Vì vậy, lệnh gọi lại ban đầu của ví dụ thực sự được thực thi bởi một lệnh gọi lại khác. Hãy cẩn thận không lồng quá nhiều lệnh gọi lại nếu bạn có thể giúp được, vì điều này có thể dẫn đến một thứ gọi là "địa ngục gọi lại"! Như tên của nó, nó không phải là một niềm vui để giải quyết.

1. Ví dụ Blocking Code

* Tạo một dòng text với tên input.txt với nội dung sau đây:



* Tạo một file js với tên callback1.js với nội dung sau đây:



* Bây giờ chạy lệnh sau để xem kết quả:



* Kết quả:

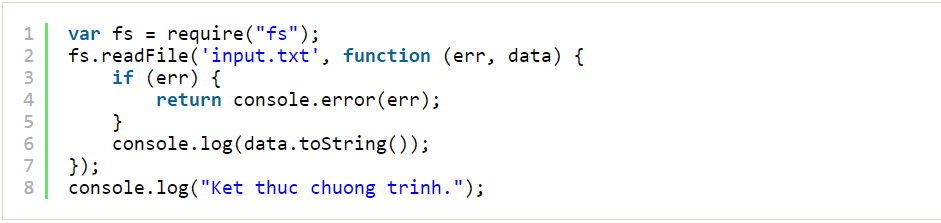


1. Ví dụ Non-Blocking Code

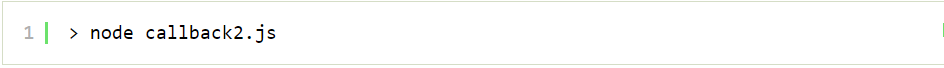
* Tạo một file với tên input.txt với nội dung sau đây:



* Cập nhật callback2.js với dòng code sau:



* Bây giờ chạy callback2.js để xem kết quả:



* Kết quả:



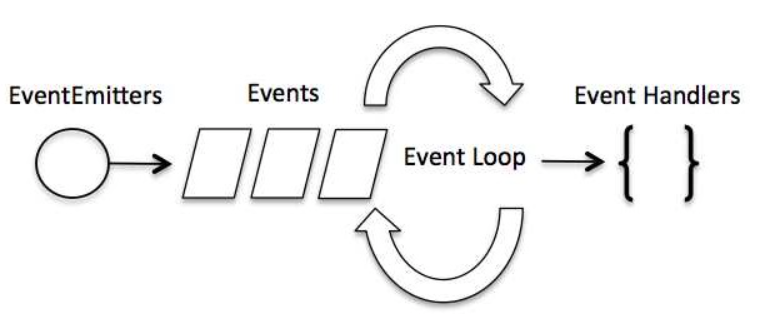
* Hai ví dụ trên giải thích định nghĩa cách gọi blocking và non-blocking. Ví dụ đầu tiên chỉ ra rằng chương trình bị khóa cho đến khi nó hoàn thành nhiệm vụ đọc file và chỉ tiếp tục chạy vài giây sau đó, chương trình thứ 2 không đợi cho việc đọc file và tiếp tục in "Ket thuc chuong trinh." cùng thời điểm thực hiện chương trình.

**Event Loop trong Node.js**

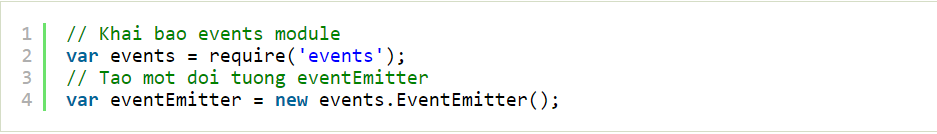
Node.js là ứng dụng đơn luồng nhưng có hỗ trợ việc xử lí đồng thời thông qua các định nghĩa về event (sự kiện) và callback. Tất cả các API của Node.js có tính chất không đồng bộ và được xử lí đơn luồng, nó sử dụng hàm **async** để duy trì sự đồng thời. Node.js sử dụng Observer Pattern. Các Thread trong Node.js có một Event Loop và bất cứ khi nào có tác vụ nào hoàn thành, nó sẽ kích hoạt sự kiện tương ứng để báo cho Event Listener sẵn sàng thực hiện.

1. Khái niệm Event Driven trong Node.js

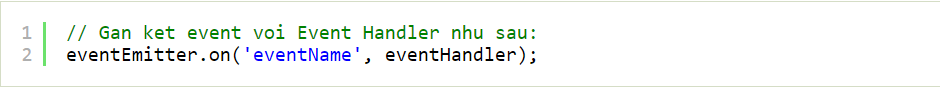
* Node.js sử dụng rất nhiều sự kiện, đó là lý do tại sao Node.js khá nhanh so với các công nghệ khác. Ngay khi Node khởi động Server của nó, nó sẽ nhanh chóng khởi tạo các biến, khai báo các hàm và sau đó đơn giản là chờ đợi các sự kiện xảy ra.
* Trong ứng dụng xử lý sự kiện, nhìn chung vòng lặp chính lắng nghe các sự kiện, và sau đó trigger đến hàm callback khi một trong những sự kiện được phát hiện.



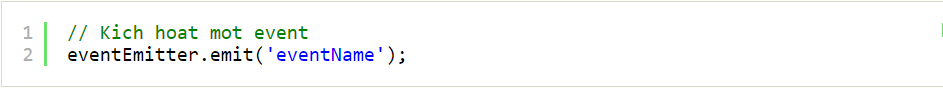
* Nhìn chung các Event là tương tự như các hàm callback. Điểm khác nhau nằm ở chỗ, hàm callback gọi khi một hàm không đồng bộ và trả về kết quả của nó trong khi phần xử lí sự kiện làm việc trên Observer Pattern. Hàm này sẽ lắng nghe các sự kiện, đóng vai trò như một Observers (Người quan sát). Bất cứ khi nào một sự kiện phát sinh, các hàm Listener của nó sẽ bắt đầu thực thi. Node.js có nhiều sự kiện có sẵn thông qua **events** Module và lớp **EventEmitter** có thể dựa vào để bind sự kiện và lắng nghe sự kiện.
* Trước khi sử dụng event Module, bạn sử dụng phương thức require() để khai báo như sau:



* Sau đó, để gắn kết Event Handler với một sự kiện, bạn sử dụng cú pháp sau:

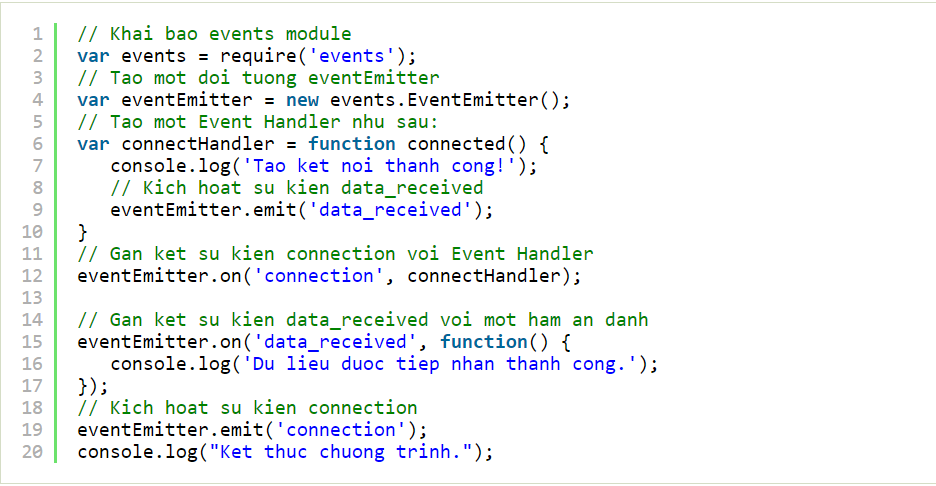


* Bạn có thể kích hoạt một sự kiện bởi sử dụng phương thức emit() của EventEmitter:



1. Ví dụ sử dụng Event Loop trong Node.js

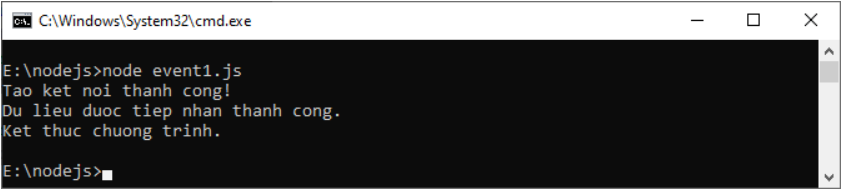
* Tạo một file js với tên là event1.js có đoạn code sau:



* Chạy chương trình trên như sau:

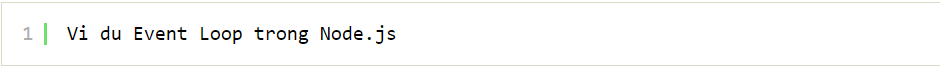


* Kết quả là:

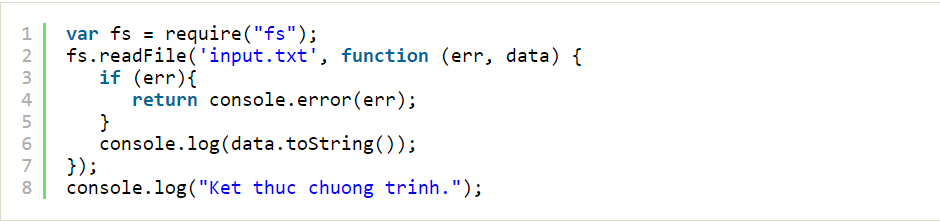


1. Ứng dụng Node hoạt động như thế nào?

* Trong ứng dụng Node.js, một hàm không đồng bộ chấp nhận một callback như tham số cuối cùng và hàm callback chấp nhận error như tham số đầu tiên. Cùng xem lại ví dụ trước. Tạo một text file với tên input.txt với nội dung sau đây:



* Trong ví dụ này, chúng ta sử dụng Module **fs** để xử lý các hoạt động File I/O. Đầu tiên, bạn tạo một file js với tên là event2.js như sau:



* Ở đây, fs.readFile() là một hàm không đồng bộ với mục đích để đọc file. Nếu có một lỗi xảy ra trong quá trình đọc file, đối tượng err sẽ chứa lỗi đó, nếu không thì data sẽ chứa các phần nội dung của file đó. Hàm readFile truyền err và data đến hàm callback sau khi quá trình đọc file đã hoàn thành, và cuối cùng sẽ in ra nội dung.
* Kết quả sẽ là:

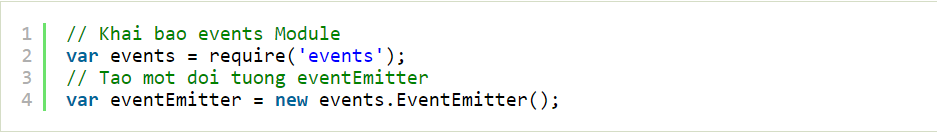


**Lớp Event Emitter trong Node.js**

Nhiều đối tượng trong Node.js sinh ra các sự kiện, ví dụ **net.Server** sinh ra một sự kiện mỗi khi có một kết nối ngang hàng đến nó, hay **fs.readStream** sinh ra sự kiện khi một file được mở. Tất cả các đối tượng này đều là sự thể hiện của lớp **EventEmitter** trong Node.js.

1. Lớp EventEmitter trong Node.js

* Lớp events.EventEmitter nằm trong events Module. Lớp này được truy cập qua cú pháp sau:



* Khi một EventEmitter gặp bất kì lỗi nào, nó sẽ sinh ra một Error Event. Khi một Listener mới được thêm, sự kiện 'newListener' sẽ được kích hoạt và một Listener sẽ bị loại bỏ, sự kiện 'removeListener' sẽ được kích hoạt.
* Event Emitter cung cấp nhiều thuộc tính như **on** hay **emit**. Thuộc tính *on* được sử dụng để gắn kết một hàm với sự kiện, và *emit* dược sử dụng để kích hoạt một sự kiện.

1. Các phương thức của lớp EventEmitter trong Node.js

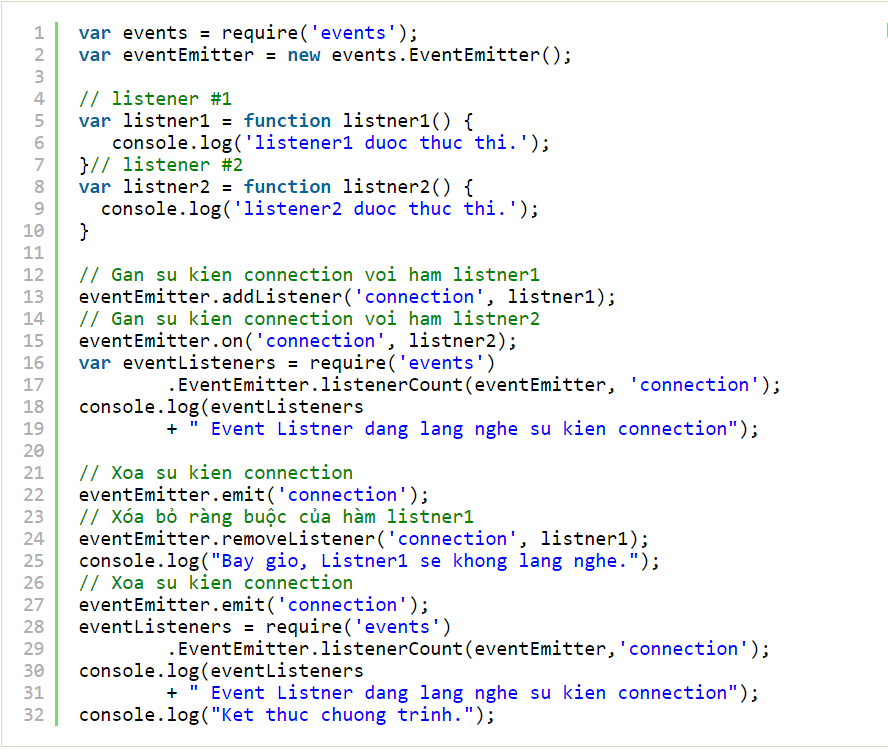
|  |  |
| --- | --- |
| **Stt Phương thức & Mô tả** | |
| 1 | **addListener(event,listener)** Thêm một Listener vào phần cuối của mảng các Listener cho một sự kiện cụ thể |
| 2 | **on(event,listener)** Thêm một Listener vào phần cuối của mảng các Listener cho một sự kiện cụ thể |
| 3 | **once(event,listener)** Thêm một One-Time Listener cho sự kiện. Listener dạng này sẽ chỉ được gọi khi sự kiện được kích hoạt, sau đó nó sẽ bị xóa |
| 4 | **removeListener(event,listener)** Xóa một Listener ra khỏi mảng các Listener cho một sự kiện nào đó. |
| 5 | **removeAllListeners([event])** Xóa tất cả Listener của một sự kiện |
| 6 | **setMaxListeners(n)** Theo mặc định, lớp EventEmitters sẽ in một lời cảnh báo nếu bạn thêm nhiều hơn 10 Listener cho một sự kiện cụ thể. Việc này khá hữu ích, bởi vì nó sẽ giúp tìm ra các lỗi gây rò rỉ bộ nhớ. Tất nhiên, không phải tất cả các Emitters đều cần được giới hạn với con số là 10. Hàm này cho phép bạn tăng con số đó. Thiết lập nó về 0 để không giới hạn lượng Listener cần thêm |
| 7 | **listeners(event)** Trả về một mảng bao gồm các Listener cho một sự kiện cụ thể nào đó |
| 8 | **emit(event,[arg1],[arg2],[...])** Thực thi từng Listener với các tham số đã cho. Trả về true nếu sự kiện có các Listener, và false nếu không có |

1. Các sự kiện của lớp EventEmitter trong Node.js

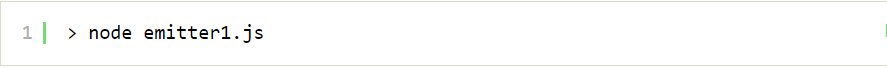
|  |  |
| --- | --- |
| **Stt Sự kiện & Mô tả** | |
| 1 | **newListener**   * **event** - Dạng chuỗi, biểu diễn tên sự kiện * **listener** - Tên hàm xử lý sự kiện   Sự kiện này được sinh bất cứ khi nào bạn thêm một Listener. Khi sự kiện này được kích hoạt, Listener có thể sẽ chưa được thêm vào mảng Listener của sự kiện |
| 2 | **removeListener**   * **event** - Dạng chuỗi, biểu diễn tên sự kiện * **listener** - Tên hàm xử lý sự kiện   Sự kiện này xảy ra bất cứ khi nào có ai đó xóa một Listener. Khi một sự kiện được kích hoạt, Listener này chưa được xóa khỏi mảng Listener của sự kiện |

1. Ví dụ về lớp EventEmitter trong Node.js

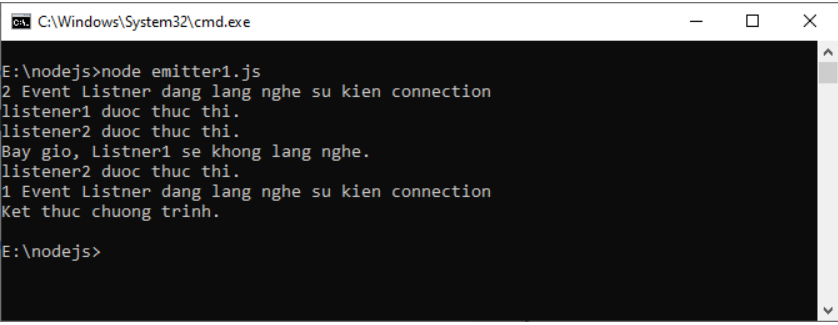
* Tạo một file js với tên là emitter1.js với nội dụng Node.js như dưới đây. Trong emitter1.js, đầu tiên bạn khai báo Module **events**. Tiếp đó, bạn sử dụng phương thức addListener() để thêm một Listener cho một sự kiện nào đó, và sử dụng các thuộc tính on và emit để thực hiện các tính năng đã trình bày ở trên.



* Chạy emitter1.js để xem kết quả:



* Kết quả:



**Buffer trong Node.js**

Javascript thuần được mã hóa Unicode, nhưng đối với các dữ liệu nhị phân thì không thật sự tốt. Khi làm việc với các luồng TCP hoặc file dữ liệu, cần thiết phải xử lý các luồng dữ liệu bát phân. Node.js cung cấp các lớp Buffer cho phép lưu trữ các dữ liệu thô như một mảng các số nguyên tương ứng với phần cấp phát bộ nhớ thô bên ngoài V8 heap.

**Lớp Buffer trong Node.js** là các lớp toàn cục và có thể được truy cập trong ứng dụng mà không cần khai báo các Module Buffer bởi phương thức require() như các Module khác.

* 1. Tạo các Buffer trong Node.js

Buffer trong Node.js có thể được xây dựng theo nhiều cách khác nhau.

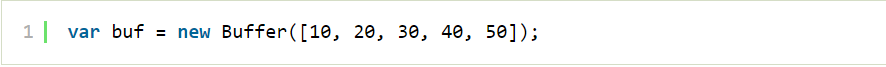
**Cách 1**

Cú pháp cho việc tạo một Buffer cỡ 10:



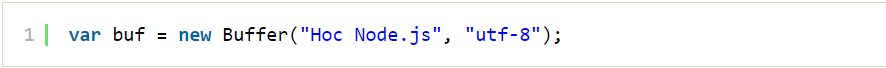
**Cách 2**

Cú pháp tạo một Buffer từ một mảng cho trước:



**Cách 3**

Cú pháp tạo một Buffer từ một chuỗi cho trước và với kiểu mã hóa tùy ý:

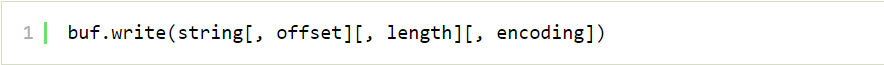


Mặc dù "utf8" là cách mã hóa mặc định nhưng bạn có thể sử dụng các cách mã hóa khác như "ascii", "utf8","base64", …

* 1. Ghi dữ liệu vào Buffer trong Node.js

**Cú pháp**

Cú pháp để ghi một Buffer trong Node.js là:



**Chi tiết về tham số**

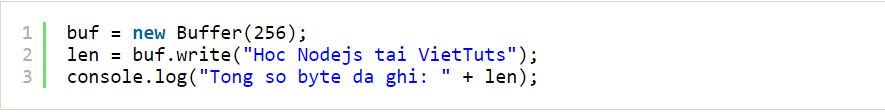
* **string**: Đây là một dữ liệu dạng chuỗi được ghi tới buffer.
* **offset**: Đây là chỉ mục để buffer bắt đầu ghi tại đó. Giá trị mặc định là số 0.
* **length**: Số lượng các byte được ghi. Mặc định là buffer.length.
* **encoding**vMã hóa được sử dụng. "utf8" là mã hóa mặc định.

**Giá trị trả về**

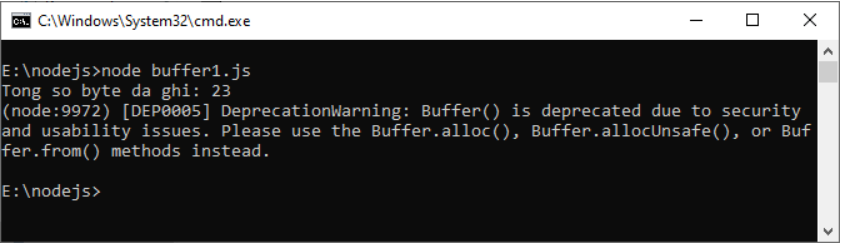
Phương thức sẽ trả về số lượng các byte được ghi. Nếu bộ nhớ trong buffer là không đủ để đáp ứng cho toàn bộ chuỗi, nó sẽ ghi một phần của chuỗi đó.

**Ví dụ cách ghi dữ liệu vào Buffer trong Node.js**

Ở ví dụ này, mình sử dụng phương thức write() nhận tham số là dữ liệu dạng chuỗi để ghi dữ liệu đó tới Buffer.



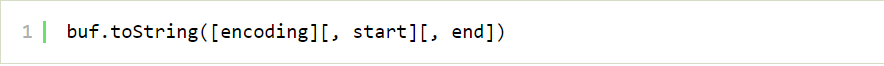
Khi chương trình trên được thực thi sẽ cho kết quả:



* 1. Đọc dữ liệu từ Buffer trong Node.js

**Cú pháp**

Cú pháp để đọc dữ liệu từ Buffer trong Node.js như sau:



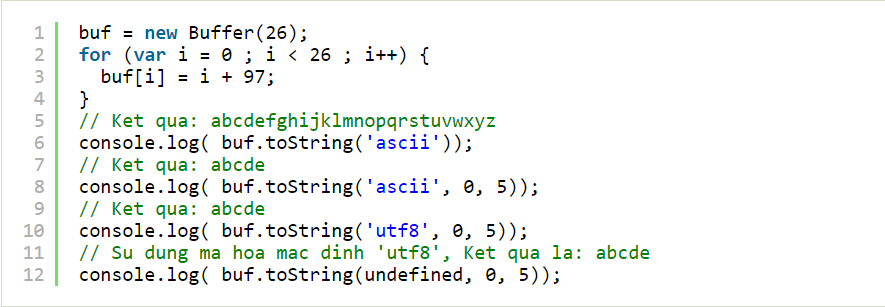
**Chi tiết về tham số**

* **encoding**: Là mã hóa để sử dụng. Mã hóa mặc định là 'utf8'.
* **start**: Chỉ mục để bắt đầu hoạt động đọc, giá trị mặc định là 0.
* **end**: Chỉ mục để kết thúc hoạt động đọc, giá trị mặc định là độ dài của Buffer.

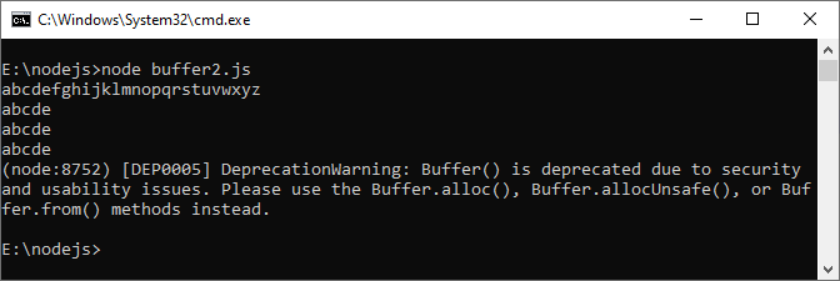
**Giá trị trả về**

Phương thức này giải mã và trả về một chuỗi từ dữ liệu đã được mã hóa trong Buffer bởi sử dụng bộ mã hóa cụ thể.

**Ví dụ cách đọc dữ liệu từ Buffer trong Node.js**



Khi chương trình trên được thực thi sẽ cho kết quả:



* 1. Chuyển đổi Buffer thành JSON trong Node.js

**Cú pháp**

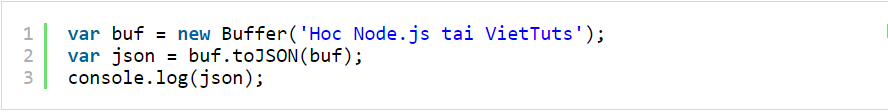
Để chuyển đổi một Buffer trong Node.js thành đối tượng JSON, bạn sử dụng phương thức toJSON() có cú pháp như sau:



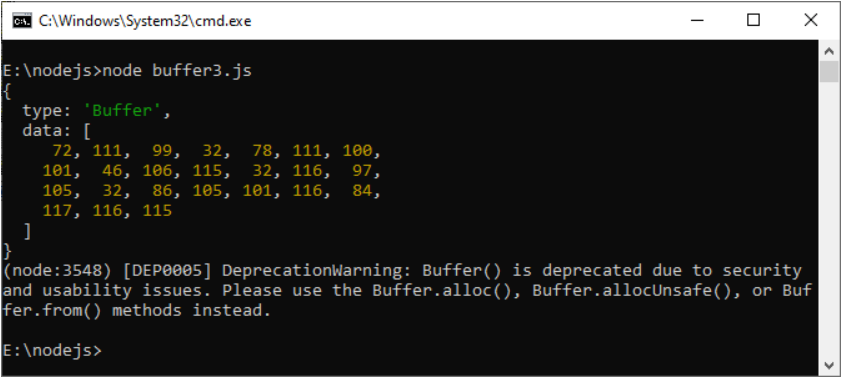
**Giá trị trả về**

Phương thức này trả về một biểu diễn JSON cho đối tượng Buffer đã cho.

**Ví dụ cách chuyển đổi Buffer thành JSON**



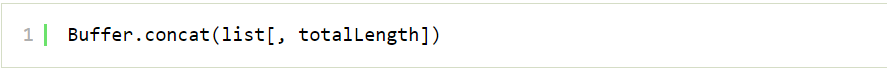
Khi chương trình trên được thực thi sẽ cho kết quả:



* 1. Ghép nối các Buffer trong Node.js

**Cú pháp**

Để nối ghép hai hoặc nhiều Buffer thành một Buffer trong Node.js, bạn sử dụng phương thức concat() như sau:



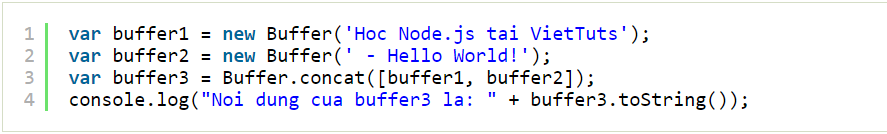
**Chi tiết về tham số**

* **list** - Xác định một mảng các Buffer được sử dụng để ghép nối thành một Buffer.
* **totalLength** - Là tổng độ dài của các Buffer sau khi đã được ghép nối.

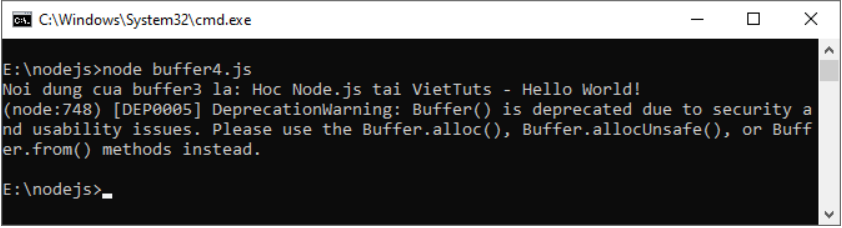
**Giá trị trả về**

Phương thức này trả về một Buffer mới.

**Ví dụ cách ghép nối các Buffer**



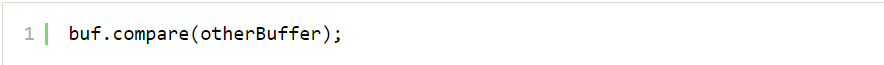
Khi chương trình trên được thực thi sẽ cho kết quả:



* 1. So sánh các Buffer trong Node.js

**Cú pháp**

Để so sánh hai Buffer trong Node.js, bạn sử dụng phương thức compare() như sau:



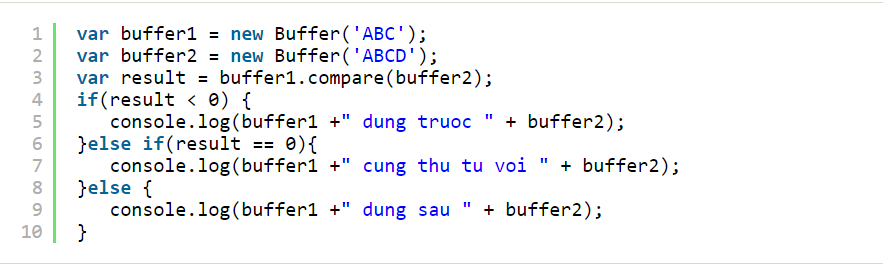
**Chi tiết về tham số**

* **otherBuffer**: Một Buffer khác để được so sánh với Buffer có tên là **buf**.

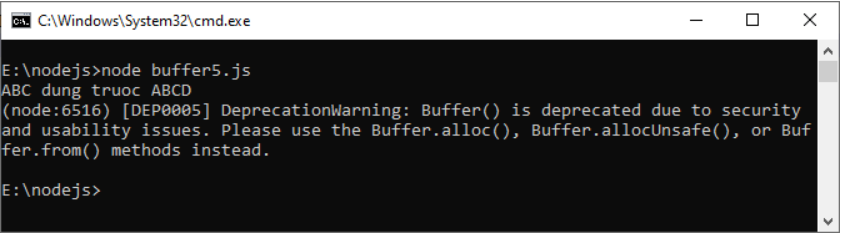
**Giá trị trả về**

Trả về một giá trị dạng số thể hiện Buffer này là đứng trước, sau hay cùng thứ tự với Buffer kia.

**Ví dụ minh họa cách so sánh hai Buffer trong Node.js**



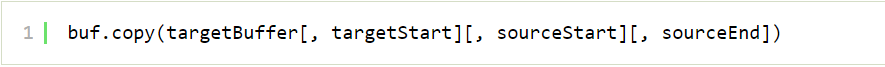
Khi chương trình trên được thực thi sẽ cho kết quả:



* 1. Sao chép Buffer trong Node.js

**Cú pháp**

Để sao chép Buffer trong Node.js, bạn sử dụng phương thức copy() như sau:



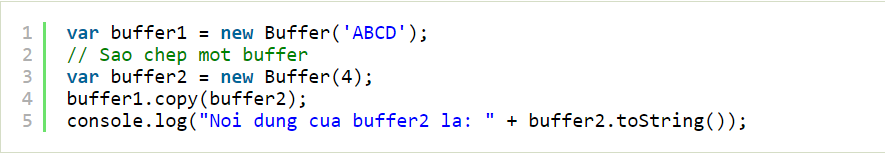
**Chi tiết về tham số**

* **targetBuffer**: Đối tượng Buffer, đây là nơi Buffer sẽ được sao chép.
* **targetStart**: Dạng số, mặc định là 0.
* **sourceStart**: Dạng số, mặc định là 0.
* **sourceEnd**: Dạng số, mặc định là độ dài của buffer.

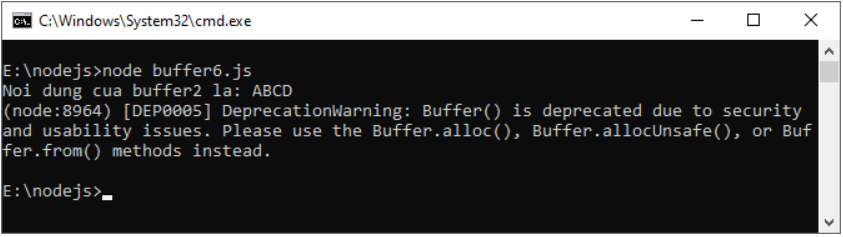
**Giá trị trả về**

Phương thức copy() này không trả về bất kỳ giá trị nào.

**Ví dụ minh họa cách sao chép Buffer**



Khi chương trình trên được thực thi sẽ cho kết quả:



* 1. Chia nhỏ Buffer trong Node.js

**Cú pháp**

Để lập một Buffer con của một Buffer trong Node.js, bạn sử dụng phương thức slice() như sau:



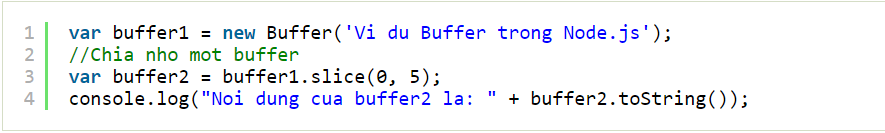
**Chi tiết về tham số**

* **start**: Dạng số, giá trị mặc định là 0.
* **end**: Dạng số, giá trị mặc định là buffer.length

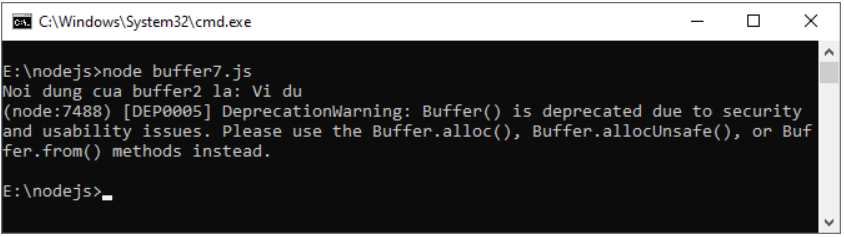
**Giá trị trả về**

Trả về một Buffer mới mà tham chiếu tới cùng vùng bộ nhớ như Buffer cũ.

**Ví dụ**



Khi chương trình trên được thực thi sẽ cho kết quả:



* 1. Độ dài Buffer trong Node.js

**Cú pháp**

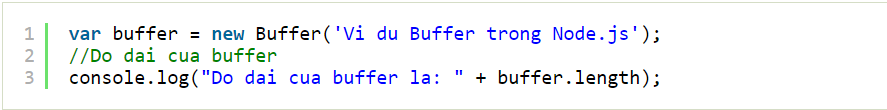
Để lấy độ dài (bằng giá trị byte) của một Buffer trong Node.js, bạn sử dụng thuộc tính length như sau:



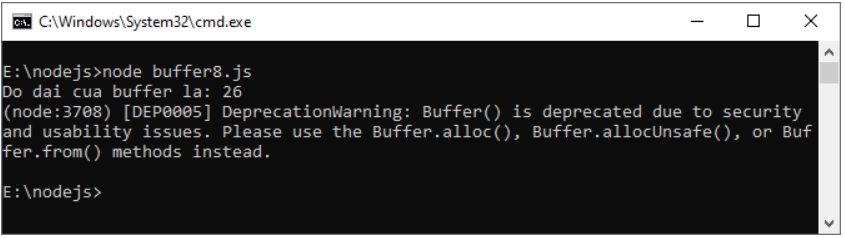
**Giá trị trả về**

Trả về độ dài bằng byte của một Buffer.

**Ví dụ**



Khi chương trình trên được thực thi sẽ cho kết quả:



# **TRUY VẤN TỆP TIN**

**\*Kiến thức chung:** Công cụ: Var fs = require(‘fs’) ;  
Mọi phương thức trong mô-đun đều có 2 dạng: Đồng bộ và không đồng bộ

-Đồng bộ **(synchronous)**: code sẽ chạy tuần tự theo trình tự đã viết sẵn

-Không đồng bộ **(asynchronous)**: code ở dưới có thể tiếp tục chạy mặc dù đoạn code ở trên chưa được xử lí hết và trả về kết quả.

🡪 Tốt hơn là sử dụng một phương thức không đồng bộ thay vì một phương thức đồng bộ, vì phương thức thứ nhất không bao giờ chặn một chương trình trong quá trình thực thi, trong khi phương thức thứ hai thì có.

1. **Cú pháp đọc tệp**

Các fs.readFile() phương pháp được sử dụng để đọc các tập tin trên máy tính của bạn.

Ví dụ:

Ta có file :

filedemo.html

<html>  
<body>  
<h1>My Header</h1>  
<p>My paragraph.</p>  
</body>  
</html>

Đọc tệp trên:

var http = require('http');  
var fs = require('fs');  
http.createServer(function (req, res) {  
**fs.readFile('filedemo.html',**function**(err, data) {**    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write(data);  
    return res.end();  
  });  
}).listen(8080);

1. **Cú pháp tạo tệp**

Mô-đun Hệ thống Tệp có các phương pháp để tạo tệp mới như:

* fs.appendFile()
* fs.open()
* fs.writeFile()

Các fs.appendFile()phương pháp gắn thêm nội dung cụ thể vào một tập tin. Nếu tệp không tồn tại, tệp sẽ được tạo

Ví dụ:

var fs = require('fs');  
  
fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {  
  if (err) throw err;  
  console.log('Saved!');  
});

Các fs.open()phương pháp có một "flag" như là đối số thứ hai, nếu lá cờ là "w" cho "writing", các tập tin định được mở ra cho văn bản. Nếu tệp không tồn tại, một tệp trống sẽ được tạo:

Ví dụ:

var fs = require('fs');  
  
fs.open('mynewfile2.txt', 'w', function (err, file) {  
  if (err) throw err;  
  console.log('Saved!');  
});

Các fs.writeFile()phương pháp thay thế các tập tin và nội dung cụ thể nếu nó tồn tại. Nếu tệp không tồn tại, một tệp mới, chứa nội dung được chỉ định, sẽ được tạo:

Ví dụ:

var fs = require('fs');  
  
fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {  
  if (err) throw err;  
  console.log('Saved!');  
});

1. **Cập nhật tệp**

Mô-đun Hệ thống Tệp có các phương pháp để cập nhật tệp:

* fs.appendFile()
* fs.writeFile()

Các fs.appendFile()phương pháp gắn thêm các nội dung quy định tại phần cuối của tập tin nhất định:

Ví dụ: // Nối “Văn bản của tôi” vào “mynew” //

var fs = require('fs');  
  
fs.appendFile('mynew.txt', ' Văn bản của tôi', function (err) {  
  if (err) throw err;  
  console.log('Updated!');  
});

Các fs.writeFile()phương pháp thay thế các tập tin và nội dung cụ thể:

Ví dụ: // Thay thế nội dung tệp “mynew.txt” //

var fs = require('fs');  
  
fs.writeFile('mynew.txt', 'This is my text', function (err) {  
  if (err) throw err;  
  console.log('Replaced!');  
});

1. **Xóa tệp tin**

Để xóa tệp bằng mô-đun Hệ thống tệp, hãy sử dụng fs.unlink() phương pháp.

Các fs.unlink()phương pháp xóa các tập tin nhất định:

Ví dụ: // Xóa file “mynew.txt” //

var fs = require('fs');  
  
fs.unlink('mynewfile2.txt', function (err) {  
  if (err) throw err;  
  console.log('File deleted!');  
});

1. **Đổi tên tệp**

Để đổi tên tệp bằng mô-đun Hệ thống tệp, hãy sử dụng fs.rename() phương pháp này.

Các fs.rename()phương pháp đổi tên các tập tin nhất định:

Ví dụ: // Đổi tên “myname” thành “myrename”

var fs = require('fs');  
  
fs.rename('myname.txt', 'myrenamed.txt', function (err) {  
  if (err) throw err;  
  console.log('File Renamed!');  
});

1. **Cú pháp viết tệp**

Sau đây là cú pháp của một trong các phương pháp để ghi vào tệp: fs.writeFile (tên tệp, dữ liệu [, tùy chọn], gọi lại)   
Phương thức này sẽ ghi đè tệp nếu tệp đã tồn tại. Nếu bạn muốn ghi vào một tệp hiện có, thì bạn nên sử dụng một phương pháp khác có sẵn. Đây là mô tả của các tham số được sử dụng:   
  
• path - Đây là chuỗi có tên tệp bao gồm cả đường dẫn.   
• dữ liệu - Đây là Chuỗi hoặc Bộ đệm được ghi vào tệp.   
• options - Tham số thứ ba là một đối tượng sẽ chứa {encoding, mode, flag}. Theo mặc định, mã hóa là utf8, chế độ là giá trị bát phân 0666 và flag là 'w'   
• callback - Đây là hàm gọi lại nhận một lỗi tham số duy nhất trả về lỗi trong trường hợp có bất kỳ lỗi viết nào

# **Utility Module trong Nodejs**

Thư viện Nodejs Modejs cung cấp nhiều Utility Module giúp ích bạn rất nhiều trong khi phát triển các ứng dụng về Nodejs. Bảng dưới đây liệt kê và mô tả một số Utility Module tiêu biểu.

|  |  |
| --- | --- |
| **STT** | **Tên Module và Miêu tả** |
| 1 | [**os Module**](https://vietjack.com/nodejs/os_module_trong_nodejs.jsp) Cung cấp các hàm tiện ích cơ bản liên quan tới hệ điều hành. |
| 2 | [**path Module**](https://vietjack.com/nodejs/path_module_trong_nodejs.jsp) Cung cấp các tiện ích để xử lý và biến đổi đường dẫn tới file. |
| 3 | [**net Module**](https://vietjack.com/nodejs/net_module_trong_nodejs.jsp) Hoạt động như là Network Wrapper, biến Server và Client thành các Stream. |
| 4 | [**dns Module**](https://vietjack.com/nodejs/dns_module_trong_nodejs.jsp) Cung cấp các hàm để thực hiện DNS Lookup. |
| 5 | [**domain Module**](https://vietjack.com/nodejs/domain_module_trong_nodejs.jsp) Được sử dụng để can thiệp các lỗi chưa được xử lý |

**\* Giải thích các thuật ngữ:**

Network Wrapper là danh sách điều khiển truy cập dựa trên máy và được sử dụng để lọc truy cập mạng với các dịch vụ. nó có lợi thế hơn so với tường lửa thông thường: chúng hoạt động trong lớp 7(ứng dụng),do đó, chúng có thể lọc các truy vấn ngay cả khi sử dụng mã hóa.

Stream là một collection của dữ liệu giống như string hay array, sự khác nhau duy nhất là các streams không tồn tại cùng lúc. Điều kiển streams thực sự mạnh đó là khả năng làm việc với dữ liệu lớn(big data) hay nguồn dữ liệu từ bên ngoài.(đã được trình bày rồi ).

DNS là hệ thống phân giải tên miền, viết tắc của Domain Name Server, nó dịch tên miền Internet và tên máy chủ sang địa chỉ IP(giúp các máy chủ và thiết bị mạng có thể hiểu được) và ngược lại. Trên Internet, Dns tự động chuyển đổi các tên miền chúng ta gõ vào thanh địa chỉ trên trình duyệt web thành địa chỉ IP.

DNS Lookup chính là quá trình tìm kiếm địa chỉ IP của bất kỳ URL, đường dẫn nào trên Internet được gọi là DNS lookup

1.OS Module

OS module trong Nodejs cung cấp các hàm tiện ích cơ bản liên quan đến hệ điều hành. Module này có thể được import bởi cú pháp sau đây :

var os = require("os")

Phương thức của OS Module

|  |  |
| --- | --- |
| **STT** | **Phương Thức và Miêu tả** |
| 1 | **os.tmpdir()** Trả về thư mục mặc định cho các file tạm thời |
| 2 | **os.hostname()** Trả về hostname của Hệ điều hành. |
| 3 | **os.type()** Trả về tên của Hệ điều hành. |
| 4 | **os.platform()** Trả về nền tảng của Hệ điều hành. |
| 5 | **os.arch()** Trả về cấu trúc CPU của Hệ điều hành. |
| 6 | **os.totalmem()** Trả về tổng dung lượng bộ nhớ (đơn vị byte). |
| 7 | **os.freemem()** Trả về tổng dung lượng bộ nhớ chưa dùng đến (đơn vị byte). |
| 8 | **os.cpus()** Trả về mảng đối tượng chứa thông tin về CPU/core: model, speed (giá trị MHz), và thời gian. |
| 9 | **os.networkInterfaces()** Lấy danh sách các Network Interface. |

Thuộc tính của OS Module

|  |  |
| --- | --- |
| **STT** | **Thuộc Tính và Miêu tả** |
| 1 | **os.EOL** Một hằng số định nghĩa End-Of-Line cho Hệ điều hành. |

Ví Dụ

var os = require("os");

*// In thong tin ve kieu cua OS*

console.log('OS Type la: ' + os.type());

*// In thong tin ve OS platform*

console.log('OS Platform : ' + os.platform());

*// In thong tin ve tong dung luong bo nho*

console.log('Tong dung luong bo nho : ' + os.totalmem() + " byte.");

*// In thong tin ve tong dung luong bo nho roi*

console.log('Tong dung luong bo nho roi : ' + os.freemem() + " byte.");

Chạy main.js để xem kết quả:

node main.js

Kiểm tra kết quả:

OS Type la: Windows\_NT

OS Platform : win32

Tong dung luong bo nho : 4151881728 byte.

Tong dung luong bo nho roi : 765489152 byte.

2.Path Module

Path Module được sử dụng để xử lý và chuyển đổi các đường dẫn tới file. Module này được import theo cú pháp:

Phương thức của Path Module trong Nodejs

|  |  |
| --- | --- |
| **STT** | **Phương Thức và Miêu tả** |
| 1 | **path.normalize(p)** Chuẩn hóa đường dẫn |
| 2 | **path.join([path1][, path2][, ...])** Kết hợp tất cả các tham số là đường dẫn với nhau và sau đó chuẩn hóa đường dẫn kết quả tạo ra |
| 3 | **path.resolve([from ...], to)** Resolve một đường dẫn tuyệt đối. |
| 4 | **path.isAbsolute(path)** Xác định xem đó có phải là đường dẫn tuyệt đối không. Một đường dẫn tuyệt đối luôn luôn trỏ đến cùng một vị trí, bất kể thư mục đang làm việc là gì. |
| 5 | **path.relative(from, to)** Resolve một đường dẫn tương đối. |
| 6 | **path.dirname(p)** Trả về tên thư mục của một đường dẫn. |
| 7 | **path.basename(p[, ext])** Trả về phần cuối cùng của một đường dẫn |
| 8 | **path.extname(p)** Trả về đuôi của đường dẫn (ví dụ txt) |
| 9 | **path.parse(pathString)** Trả về một đối tượng từ một chuỗi biểu diễn đường dẫn |
| 10 | **path.format(pathObject)** Trả về một chuỗi biểu diễn đường dẫn từ một đối tượng, phương thức này trái ngược với phương thức path.parse trên. |

Thuộc Tính của Path Module

|  |  |
| --- | --- |
| **STT** | **Thuộc tính và Miêu tả** |
| 1 | **path.sep** Biểu thị File Separator của nền tảng cụ thể. Nó có thể là '\\' hoặc '/'. |
| 2 | **path.delimiter** Biểu thị Path Delimiter của nền tảng cụ thể, có thể là ';' hoặc ':'. |

Ví Dụ

Ví dụ dưới đây minh họa các sử dụng một số phương thức và thuộc tính của Path Module:

var path = require("path");

*// Chuan hoa duong dan voi phuong thuc normalize()*

console.log('Phuong thuc NORMALIZE : ' + path.normalize('/test/test1//2slashes/1slash/tab/..'));

*// Ket hop cac tham so dau vao de tao mot duong dan*

console.log('Phuong thuc JOIN : ' + path.join('/test', 'test1', '2slashes/1slash', 'tab', '..'));

*// Resolve mot duong dan tuyet doi*

console.log('Phuong thuc RESOLVE : ' + path.resolve('main.js'));

*// Lay thong tin cua duoi duong dan*

console.log('Phuong thuc EXTNAME : ' + path.extname('main.js'));

Kết Quả:

Phuong thuc NORMALIZE : \test\test1\2slashes\1slash

Phuong thuc JOIN : \test\test1\2slashes\1slash

Phuong thuc RESOLVE : E:\Nodejs\main.js

Phuong thuc EXTNAME : .js

3 Net Module

Net Module được sử dụng để tạo Server và Client. Module này cung cấp một Network Wrapper không đồng bộ và có thể được import với cú pháp sau:

var net = require("net")

Phương thức của Net Module

|  |  |
| --- | --- |
| **STT** | **Phương Thức và Miêu tả** |
| 1 | **net.createServer([options][, connectionListener])** Tạo một TCP Server mới. Tham số connectionListener tự động được thiết lập để thành một Listener cho sự kiện 'connection'. |
| 2 | **net.connect(options[, connectionListener])** Đây là một phương thức factory, trả về một 'net.Socket' mới và kết nối tới address và port đã cho. |
| 3 | **net.createConnection(options[, connectionListener])** Đây là một phương thức factory, trả về một 'net.Socket' mới và kết nối tới address và port đã cho. |
| 4 | **net.connect(port[, host][, connectListener])** Tạo một kết nối TCP tới port trên host đã cho. Nếu host không được cung cấp, thì giá trị mặc định là 'localhost'. Tham số connectListener sẽ được thêm vào như là Listener cho sự kiện 'connect'. |
| 5 | **net.createConnection(port[, host][, connectListener])** Tạo một kết nối TCP tới port trên host đã cho. Nếu host không được cung cấp, thì giá trị mặc định là 'localhost'. Tham số connectListener sẽ được thêm vào như là Listener cho sự kiện 'connect'. |
| 6 | **net.connect(path[, connectListener])** Tạo một kết nối Unix Socket tới đường dẫn đã cho. Tham số connectListener sẽ được thêm vào như là Listener cho sự kiện 'connect'. |
| 7 | **net.createConnection(path[, connectListener])** Tạo một kết nối Unix Socket tới đường dẫn đã cho. Tham số connectListener sẽ được thêm vào như là Listener cho sự kiện 'connect'. |
| 8 | **net.isIP(input)** Kiểm tra xem input có phải là một địa chỉ IP address không. Trả về giá trị 0 cho một chuỗi không hợp lệ, 4 cho phiên bản địa chỉ IP v4, và trả về 6 cho địa chỉ IP v6. |
| 9 | **net.isIPv4(input)** Trả về true nếu input là địa chỉ IP v4, nếu không là false. |
| 10 | **net.isIPv6(input)** Trả về true nếu input là địa chỉ IP v6, nếu không là false. |

\*Giải thích thuật ngữ:

**TCP/IP viết tắt của Transmission Control Protocol (TCP) và Internet Protocol (IP) là giao thức cài đặt truyền thông, chồng giao thức mà hầu hết các mạng máy tính ngày nay đều sử dụng để kết nối.**

Unix Socket là một điểm giao tiếp để trao đổi dữ liệu các ứng dụng trên cùng một máy tính. Khác với giao thức TCP/IP thực hiện ở giao thức mạng, Unix socket thực hiện ở nhân hệ điều hành, nhờ vậy có thể tránh được các bước kiểm tra hoặc routing, đem lại tốc độ kết nối nhanh hơn và nhẹ hơn so với TCP/IP.

Lớp net.Server

Lớp net.Server được sử dung để tạo một TCP Server hoặc Local Server.

Phương thức của lớp net.Server trong Nodejs

|  |  |
| --- | --- |
| **STT** | **Phương Thức và Miêu tả** |
| 1 | **server.listen(port[, host][, backlog][, callback])** Bắt đầu chấp nhận kết nối trên port và host đã cho. Nếu không cung cấp tham số host, Server sẽ chấp nhận các kết nối trực tiếp tới bất kỳ địa chỉ IPv4 nào (INADDR\_ANY). Nếu giá trị port là 0 sẽ gán một port ngẫu nhiên. |
| 2 | **server.listen(path[, callback])** Khởi động một Local Server để nghe các kết nối trên path đã cho. |
| 3 | **server.listen(handle[, callback])** Đối tượng handle có thể được thiết lập cho Server hoặc Socket. Việc này làm cho Server chấp nhận kết nối tới một handle cụ thể. |
| 4 | **server.listen(options[, callback])** Tham số options có thể là các thuộc tính port, host, và backlog. Tham số callback là một hàm callback tùy ý mà làm việc như khi gọi tới phương thức server.listen(port, [host], [backlog], [callback]) . |
| 5 | **server.close([callback])** Đóng tất cả các kết nối đã kết thúc và Server sinh một sự kiện 'close'. |
| 6 | **server.address()** Trả về một địa chỉ bound address và port của Server, như đã được báo cáo bởi Hệ điều hành. |
| 7 | **server.unref()** Việc gọi phương thức unref trên một Server sẽ cho phép chương trình thoát ra. |
| 8 | **server.ref()** Trái ngược với phương thức unref(), việc gọi phương thức ref() sẽ không cho phép chương trình thoát ra |
| 9 | **server.getConnections(callback)** Lấy số các kết nối đồng thời trên một Server một cách không đồng bộ. Hàm callback nên nhận hai tham số err (để Resolve lỗi) và count (để đếm). |

\*Giải thích Thuật Ngữ:

**INADDR\_ANY** được sử dụng khi bạn không cần liên kết ổ cắm với một IP cụ thể. Khi bạn sử dụng giá trị này làm địa chỉ khi gọi bind() , ổ cắm chấp nhận kết nối với tất cả IP của máy

Sự kiện của lớp net.Server trong Nodejs

|  |  |
| --- | --- |
| **STT** | **Sự Kiện và Miêu tả** |
| 1 | **listening** Xảy ra khi Server đã được gắn kết sau một lời gọi tới server.listen. |
| 2 | **connection** Xảy ra khi tạo một kết nối mới |
| 3 | **close** Xảy ra khi đóng Server |
| 4 | **error** Xảy ra khi xuất hiện bất kỳ lỗi nào. Sự kiện 'close' sẽ trực tiếp được gọi sau sự kiện này. |

Lớp net.Socket

Đối tượng này là lớp trừu tượng của TCP hoặc Local Socket. net.Socket kế thừa duplex Stream interface. Chúng có thể được tạo bởi người dùng hoặc bởi một Client (bởi phương thức connect()) hoặc có thể được tạo bởi Node.js và được truyền tới người dùng thông qua sự kiện 'connection' của một Server.

Sự kiện của net.Socket trong Nodejs

net.Socket là một eventEmitter và nó sinh các sự kiện sau.

|  |  |
| --- | --- |
| **STT** | **Sự Kiện và Miêu tả** |
| 1 | **lookup** Xảy ra sau khi đã Resolve một hostname nhưng trước khi kết nối |
| 2 | **connect** Xảy ra khi một kết nối Socket được thiết lập thành công |
| 3 | **data** Xảy ra khi dữ liệu đã được nhận. Tham số data sẽ là một Buffer hoặc String. Phần mã hóa của data được thiết lập bởi socket.setEncoding(). |
| 4 | **error** Xảy ra khi xuất hiện bất kỳ lỗi nào. Sự kiện 'close' sẽ được gọi trực tiếp sau sự kiện này. |
| 5 | **close** Xảy ra khi Socket đã được đóng |

Thuộc tính của net.Socket trong Nodejs

net.Socket có nhiều thuộc tính hữu ích giúp bạn điều khiển tốt hơn trong việc tương tác với Socket.

|  |  |
| --- | --- |
| **STT** | **Thuộc tính và Miêu tả** |
| 1 | **socket.bufferSize** Thuộc tính này chỉ số lượng ký tự đã được đệm |
| 2 | **socket.remoteAddress** Biểu diễn chuỗi của địa chỉ Remote IP. |
| 3 | **socket.remoteFamily** Biểu diên chuỗi của Remote IP Family. Đó là 'IPv4' hoặc 'IPv6'. |
| 4 | **socket.remotePort** Biểu diễn dạng số của Remote Port. Ví dụ 80 hoặc 21. |
| 5 | **socket.localAddress** Biểu diễn chuỗi của địa chỉ Local IP mà một Remote Client kết nối tới. Ví dụ, nếu bạn đang lắng nghe trên '0.0.0.0' và Client kết nối trên '192.168.1.1', thì giá trị sẽ là '192.168.1.1'. |
| 6 | **socket.localPort** Biểu diễn dạng số của Local Port. Ví dụ 80 hoặc 21. |
| 7 | **socket.bytesRead** Số lượng byte đã nhận được. |
| 8 | **socket.bytesWritten** Số lượng byte đã được gửi. |

Phương thức của net.Socket trong Nodejs

|  |  |
| --- | --- |
| **STT** | **Phương Thức và Miêu tả** |
| 1 | **new net.Socket([options])** Xây dựng một đối tượng Socket mới. |
| 2 | **socket.connect(port[, host][, connectListener])** Mở kết nối cho một Socket đã cho. Nếu bạn cung cấp hai tham số port và host, thì khi đó Socket sẽ được mở dưới dạng như là một TCP Socket. Nếu bạn không cung cấp host, thì giá trị mặc định là localhost. Nếu bạn cung cấp tham số path, thì Socket sẽ đươc mở dưới dạng như một Unix Socket tới đường dẫn path đó. |
| 3 | **socket.connect(path[, connectListener])** Mở kết nối cho một Socket đã cho. Nếu bạn cung cấp hai tham số port và host, thì khi đó Socket sẽ được mở dưới dạng như là một TCP Socket. Nếu bạn không cung cấp host, thì giá trị mặc định là localhost. Nếu bạn cung cấp tham số path, thì Socket sẽ đươc mở dưới dạng như một Unix Socket tới đường dẫn path đó. |
| 4 | **socket.setEncoding([encoding])** Thiết lập mã hóa encoding cho Socket dưới dạng như một Readable Stream. |
| 5 | **socket.write(data[, encoding][, callback])** Gửi dữ liệu trên Socket. Tham số thứ hai xác định mã hóa trong trường hợp dữ liệu dạng chuỗi. Mã hóa mặc định là UTF8. |
| 6 | **socket.destroy()** Bảo đảm rằng không có bất kỳ hoạt động I/O xảy ra trên Socket này. Phương thức này chỉ cần thiết khi xuất hiện lỗi. |
| 7 | **socket.pause()** Tạm dừng việc đọc dữ liệu. Do đó sự kiện 'data' không được sinh |
| 8 | **socket.resume()** Tiếp tục việc đọc dữ liệu sau khi đã tạm dừng với phương thức pause(). |

Ví Dụ

Dưới đây là ví dụ minh họa một số phương thức và thuộc tính của net Module trong Node.js.

*Tạo server.js có nội dung:*

var net = require('net');

var server = net.createServer(function(connection) {

   console.log('Ket noi voi Client');

   connection.on('end', function() {

      console.log('Mat ket noi voi Client');

   });

   connection.write('Hello World!\r\n');

   connection.pipe(connection);

});

server.listen(8080, function() {

  console.log('Server dang lang nghe');

});

Kết quả:

Server dang lang nghe

Tiếp đó, bạn tạo client.js như dưới đây:

var net = require('net');

var client = net.connect({port: 8080}, function() {

   console.log('Da ket noi voi Server!');

});

client.on('data', function(data) {

   console.log(data.toString());

   client.end();

});

client.on('end', function() {

   console.log('Mat ket noi voi Server');

});

Kết quả:

Da ket noi voi Server!

Hello World!

Mat ket noi voi Server

4. DNS Module

**dns** Module trong Node.js được sử dụng để thực hiện DNS Lookup. Module cung cấp một Network Wrapper không đồng bộ và có thể được import với cú pháp.

var dns = require("dns")

Phương thức của DNS module

|  |  |
| --- | --- |
| **STT** | **Phương Thức và Miêu tả** |
| 1 | **dns.lookup(hostname[, options], callback)** Resolve một hostname (ví dụ 'google.com') vào trong bản ghi A (IPv4) hoặc AAAA (IPv6) đầu tiên được tìm thấy. Tham số options có thể là một đối tượng hoặc một số nguyên. Nếu bạn không cung cấp tham số options, thì các địa chỉ IP v4 và v6 đều hợp lệ. Nếu options là một số nguyên, thì nó phải là 4 hoặc 6. |
| 2 | **dns.lookupService(address, port, callback)** Resolve một address và port đã cho vào trong một hostname |
| 3 | **dns.resolve(hostname[, rrtype], callback)** Resolve một hostname (ví dụ 'google.com') vào trong một mảng các kiểu bản ghi đã được xác định bởi tham số **rrtype**. |
| 4 | **dns.resolve4(hostname, callback)** Giống như dns.resolve(), nhưng chỉ cho các truy vấn IPv4 (một truy vấn A). |
| 5 | **dns.resolve6(hostname, callback)** Giống như dns.resolve4(), nhưng chỉ cho các truy vấn (một truy vấn AAAA). |
| 6 | **dns.resolveMx(hostname, callback)** Giống như dns.resolve(), nhưng chỉ cho các truy vấn Mail Exchange. |
| 7 | **dns.resolveTxt(hostname, callback)** Giống như dns.resolve(), nhưng chỉ cho các truy vấn Text |
| 8 | **dns.resolveSrv(hostname, callback)** Giống như dns.resolve(), nhưng chỉ cho các truy vấn SRV |
| 9 | **dns.resolveSoa(hostname, callback)** Giống như dns.resolve(), nhưng chỉ cho các truy vấn SOA |
| 10 | **dns.resolveNs(hostname, callback)** Giống như dns.resolve(), nhưng chỉ cho các truy vấn NS |
| 11 | **dns.resolveCname(hostname, callback)** Giống như dns.resolve(), nhưng chỉ cho các truy vấn CNAME |
| 12 | **dns.reverse(ip, callback)** Đảo ngược việc resolve một địa chỉ IP thành một mảng bao gồm các hostname |

## Giá trị rrtype của dns Module trong Node.sj

Danh sách dưới liệt kê các giá trị rrtypes được sử dụng bởi phương thức dns.resolve():

* **A** - Giá trị mặc định là các địa chỉ IPV4
* **AAAA** - Các địa chỉ IPV6
* **MX** - Các bản ghi về Mail Exchange
* **TXT** - Các bản ghi về Text
* **SRV** - Các bản ghi về SRV
* **PTR** - Được sử dụng để đảo chiều IP Lookup
* **NS** - Các bản ghi về Name Server
* **CNAME** - Các bản ghi về Canonical Name
* **SOA** - Là viết tắt của Start of Authority Record

## Một số Error Code của dns Module trong Node.js

Mỗi truy vẫn DNS có thể trả về một trong các Error Code sau:

* **dns.NODATA** - DNS Server trả về phản hồi mà không bao gồm dữ liệu nào.
* **dns.FORMERR** - DNS Server thông báo truy vấn sai định dạng.
* **dns.SERVFAIL** - DNS Server trả về một lỗi chung.
* **dns.NOTFOUND** - Không tìm thấy tên miền.
* **dns.NOTIMP** - DNS Server không triển khai hoạt động request.
* **dns.REFUSED** - DNS Server từ chối truy vấn.
* **dns.BADQUERY** - Truy vấn sai định dạng.
* **dns.BADNAME** - Hostname sai định dạng.
* **dns.BADFAMILY** - Address Family không được hỗ trợ.
* **dns.BADRESP** - Phản hồi từ DNS sai định dạng.
* **dns.CONNREFUSED** - Không thể kết nối tới DNS Server.
* **dns.TIMEOUT** - Timeout trong khi kết nối tới DNS Server.
* **dns.EOF** - Viết tắt của End of file.
* **dns.FILE** - Xảy ra lỗi trong khi đọc file.
* **dns.NOMEM** - Hết bộ nhớ.
* **dns.DESTRUCTION** - Channel đang bị hủy.
* **dns.BADSTR** - Chuỗi sai định dạng.
* **dns.BADFLAGS** - Các Flag không hợp lệ.
* **dns.NONAME** - Hostname không phải dưới dạng số.

Ví Dụ

Ví dụ sau minh họa một số phương thức của dns Module trong Node.js. Bạn tạo main.js có nội dung sau:

var dns = require('dns');

dns.lookup('www.google.com', function onLookup(err, address, family) {

   console.log('Dia chi:', address);

   dns.reverse(address, function (err, hostnames) {

*if* (err) {

      console.log(err.stack);

   }

   console.log('Phuong thuc REVERSE cho dia chi: ' + address + ' thanh mang cac hostname: ' + JSON.stringify(hostnames));

});

});

Kết quả:

Dia chi: 216.58.200.68

Phuong thuc REVERSE cho dia chi: 216.58.200.68 thanh mang cac hostname: ["hkg07s30-in-f4.1e100.net"]

Domain Module

**domain** Module trong Node.js được sử dụng để can thiệp các lỗi chưa được xử lý. Các lỗi này có thể được can thiệp bằng cách sử dụng Internal Binding hoặc External Binding.

**Internal Binding** - Error Emmitter đang thực thi phần code của nó bên trong phương thức run của một Domain.

**External Binding** - Error Emmitter được thêm vào một Domain bởi sử dụng phương thức add.

Để sử dụng domain Module trong Node.js, bạn cần import theo cú pháp sau.

var domain = require("domain")

Lớp Domain của domain Module được sử dụng để cung cấp tính năng định tuyến các Error và Exception tới một đối tượng Domain. Lớp này là lớp con của lớp EventEmitter. Lớp Domain này xử lý các lỗi mà nó bắt được và lắng nghe sự kiện error của nó. Đối tượng này có thể được tạo bởi cú pháp:

var domain = require("domain");

var child = domain.create();

Phương Thức của Domain Module

|  |  |
| --- | --- |
| **STT** | **Phương Thức và Miêu tả** |
| 1 | **domain.run(function)** Chạy hàm function trong ngữ cảnh của domain, bind tất cả các Event Emitter, Timer, và các Request tầm thấp mà đã được tạo trong ngữ cảnh đó. Đây là cách cơ bản nhất để sử dụng một domain. |
| 2 | **domain.add(emitter)** Thêm Emitter tới domain. Nếu có bất kỳ Event Handler nào được gọi bởi Emitter mà ném một Error, thì nó sẽ được định tuyến tới sự kiện error của domain. |
| 3 | **domain.remove(emitter)** Trái ngược với phương thức domain.add(emitter). Xóa Emitter nào đó khỏi domain. |
| 4 | **domain.exit()** Thoát khỏi domain hiện tại |

Thuộc Tính của Domain Module

|  |  |
| --- | --- |
| **STT** | **Thuộc Tính và Miêu tả** |
| 1 | **domain.members** Một mảng các Timer và Event Emitter đã được thêm tới domain. |

Ví Dụ

Ví dụ sau minh họa một số phương thức của domain Module trong Node.js. Bạn tạo main.js có nội dung sau:

var EventEmitter = require("events").EventEmitter;

var domain = require("domain");

var emitter1 = new EventEmitter();

*// Tao mot domain boi su dung phuong thuc create()*

var domain1 = domain.create();

domain1.on('error', function(err){

   console.log("domain1 se xu ly loi: ("+err.message+")");

});

*// Them mot Emitter toi domain boi phuong thuc add()*

domain1.add(emitter1);

emitter1.on('error',function(err){

   console.log("listener se xu ly loi: ("+err.message+")");

});

emitter1.emit('error',new Error('Duoc xu ly boi listener'));

emitter1.removeAllListeners('error');

emitter1.emit('error',new Error('Duoc xu ly boi domain1'));

var domain2 = domain.create();

domain2.on('error', function(err){

   console.log("domain2 se xu ly loi: ("+err.message+")");

});

*// Them mot Emitter toi domain boi phuong thuc run()*

domain2.run(function(){

   var emitter2 = new EventEmitter();

   emitter2.emit('error',new Error('Duoc xu ly boi domain2'));

});

domain1.remove(emitter1);

emitter1.emit('error', new Error('Da chuyen doi thanh Exception. He thong bi pha vo!'));

Kết quả:

listener se xu ly loi: (Duoc xu ly boi listener)

domain1 se xu ly loi: (Duoc xu ly boi domain1)

domain2 se xu ly loi: (Duoc xu ly boi domain2)

events.js:292

*throw* er; *// Unhandled 'error' event*

      ^

Error: Da chuyen doi thanh Exception. He thong bi pha vo!

    at Object.<anonymous> (E:\Nodejs\domain\main.js:40:24)

    at Module.\_compile (internal/modules/cjs/loader.js:1063:30)

    at Object.Module.\_extensions..js (internal/modules/cjs/loader.js:1092:10)

    at Module.load (internal/modules/cjs/loader.js:928:32)

    at Function.Module.\_load (internal/modules/cjs/loader.js:769:14)

    at Function.executeUserEntryPoint [as runMain] (internal/modules/run\_main.js:72:12)

    at internal/main/run\_main\_module.js:17:47

Emitted 'error' event at:

    at Object.<anonymous> (E:\Nodejs\domain\main.js:40:10)

    at Module.\_compile (internal/modules/cjs/loader.js:1063:30)

    [... lines matching original stack trace ...]

    at internal/main/run\_main\_module.js:17:47

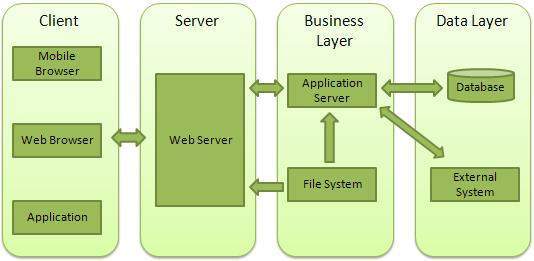
# **Web Server**

Web Server là gì

Web Server là một ứng dụng phần mềm có thể xử lý các HTTP request được gửi bởi HTTP Client (ví dụ: trình duyệt web) và trả về một trang web trong phản hồi tới Client. Web Server thường gửi các tài liệu html bên cạnh các ảnh cũng như style sheet và các đoạn Javascript.

Cấu trúc ứng dụng web server

Một ứng dụng web thường được chia thành 4 lớp như sau:



**Client** - Lớp này bao gồm các trình duyệt web và các ứng dụng có thể tạo ra các HTTP request đến Web Server.

**Server** - Lớp này bao gồm các Web Server có thể can thiệp các request được tạo bởi Client và trả về các phản hồi (response).

**Business** - Lớp này bao gồm ứng dụng trên Server có thể được tận dụng bởi các Web Server để thực hiện các tiến trình xử lý cần thiết. Lớp này tương tác với lớp Data qua các chương trình bên ngoài.

**Data** - Lớp này bao gồm các Database và bất kì các nguồn dữ liệu nào.

Tạo Web Server

Node.js cung cấp **http** Module có thể được sử dụng để tạo các HTTP client và server. Dưới đây là phần kiến trúc thu nhỏ của HTTP Server được lắng nghe trên cổng 8081.

Đầu tiên, bạn tạo *server.js* có nội dung như sau:

var http = require('http');

var fs = require('fs');

var url = require('url');

*// Tao mot Server*

http.createServer( function (request, response) {

*// Parse request co chua ten file*

   var pathname = url.parse(request.url).pathname;

*// In thong tin ve ten file ma tu do Request duoc tao.*

   console.log("Request cho " + pathname + " da duoc nhan.");

*// Doc noi dung tu File da duoc yeu cau boi Request*

   fs.readFile(pathname.substr(1), function (err, data) {

*if* (err) {

         console.log(err);

*// HTTP Status: 404 : NOT FOUND*

*// Content Type: text/plain*

         response.writeHead(404, {'Content-Type': 'text/html'});

      }*else*{

*// HTTP Status: 200 : OK*

*// Content Type: text/plain*

         response.writeHead(200, {'Content-Type': 'text/html'});

*// Ghi noi dung cua File toi phan BODY cua Response*

         response.write(data.toString());

      }

*// Gui phan BODY cua Response*

      response.end();

   });

}).listen(8081);

*// In thong bao sau tren console*

console.log('Server dang chay tai dia chi: http://127.0.0.1:8081/');

Bây giờ tạo *index.htm* cùng thư mục với server.js:

<html>

<head>

<title>Vi du Web Module trong Node.js</title>

</head>

<body>

Hello World!

</body>

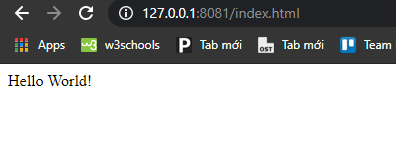
</html>

Kiểm tra kết quả:

Server dang chay tai dia chi: http://127.0.0.1:8081/

Tạo Request

Mở http://127.0.0.1:8081/index.htm trong bất kỳ trình duyệt nào và xem kết quả.



Kiểm tra kết quả

Request cho /index.html da duoc nhan.

Request cho /favicon.ico da duoc nhan.

Tạo Web Client

Để tạo web client, bạn có thể sử dụng http Module. Dưới đây là ví dụ minh họa

var http = require('http');

var fs = require('fs');

var url = require('url');

*// Tao mot Server*

http.createServer( function (request, response) {

*// Parse request co chua ten file*

   var pathname = url.parse(request.url).pathname;

*// In thong tin ve ten file ma tu do Request duoc tao.*

   console.log("Request cho " + pathname + " da duoc nhan.");

*// Doc noi dung tu File da duoc yeu cau boi Request*

   fs.readFile(pathname.substr(1), function (err, data) {

*if* (err) {

         console.log(err);

*// HTTP Status: 404 : NOT FOUND*

*// Content Type: text/plain*

         response.writeHead(404, {'Content-Type': 'text/html'});

      }*else*{

*// HTTP Status: 200 : OK*

*// Content Type: text/plain*

         response.writeHead(200, {'Content-Type': 'text/html'});

*// Ghi noi dung cua File toi phan BODY cua Response*

         response.write(data.toString());

      }

*// Gui phan BODY cua Response*

      response.end();

   });

}).listen(8081);

*// In thong bao sau tren console*

console.log('Server dang chay tai dia chi: http://127.0.0.1:8081/');

Chạy client trên một màn Terminal khác để xem kết quả:

Server dang chay tai dia chi: http:*//127.0.0.1:8081/*

Request cho /index.htm da duoc nhan.

# **Express Framework**

**Giới Thiệu về Express Framework**

Express là một framework nhỏ và tiện ích để xây dựng các ứng dụng web, cung cấp một lượng lớn các tính năng mạnh mẽ. Nó rất dễ dàng để phát triển các ứng dụng nhanh dự trên Node.js cho các ứng dụng Web. Dưới đây là các tính năng cơ bản của Express Framework.

Cho phép thiết lập các lớp trung gian để trả về các HTTP request.

Định Nghĩa bảng Routing có thể được sử dụng với các hành động khác nhau dựa trên phương thức HTTP và URL

Cho phép trả về các trang HTML dựa vào các tham số truyền vào đến template.

Cài đặt Express Framework

Đầu tiên, cài đặt Express framework sử dụng npm như sau:

$ npm install express --save

Lệnh trên lưu phần cài đặt trong thư mục node\_module và tạo thư mục express bên trong thư mục đo. Dưới đây là các thành phần module quan trọng được cài đặt cùng với express:

Body-parser : Đây là một lớp trung gian node.js để xử lý JSON, dữ liệu thô, text và mã hóa URL.

Cookie-parser: Chuyển đổi header của cookie và phân bố đến các req.cookies

Multer: Đây là một thành phần trung gian trong node.js để xử lý phần multipart/form-data.

$ npm install body-parser --save

$ npm install cookie-parser --save

$ npm install multer --save

Ví Dụ Helloworld trong node.js

Dưới đây là một ví dụ cơ bản của Express minh họa cách bật Server và lắng nghe các kết nối trên cổng 3000. Ứng dụng này trả về Hello World cho các requests đến trang chủ. Đối với các đường dẫn khác nó sẽ trả về một 404 Not Found.

Tạo server.js có nội dung như sau:

var express = require('express');

var app = express();

app.get('/', function (req, res) {

   res.send('Hello World');

})

var server = app.listen(3000, function () {

  var host = server.address().address

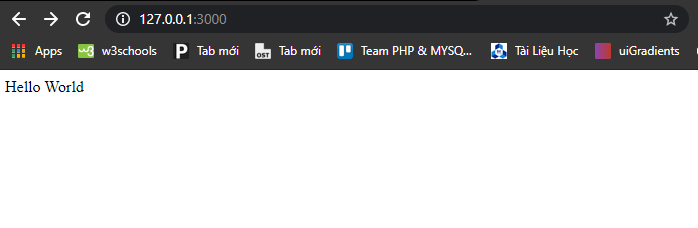
  var port = server.address().port

  console.log("Ung dung Node.js dang lang nghe tai dia chi: http://%s:%s", host, port)

})

Kết quả:

Ung dung Node.js dang lang nghe tai dia chi: http:*//:::3000*



Đối tượng Request và Response

Ứng dụng Express sử dụng một hàm callback có các tham số là các đối tượng request và response

app.get('/', function (req, res) {

*// --*

 })

Bạn có thể tham khảo chi tiết về 2 đối tương này dưới đây:  
Đối tượng request : Đối tượng này biểu diễn một HTTP request và có các thuộc tính cho các request như các chuỗi truy vẫn, tham số, body, HTTP header và những phần khác.

Đối tượng Reponse : Đối tượng này biểu diễn HTTP response được ứng dụng Express giử đi khi nó nhận về một HTTP requset.

Bạn có thể in ra các đối tượng req và res để cung cấp một lượng lớn các thông tin liên quan đến HTTP request vầ trả về các cookie,session, URL…

Định tuyền cơ bản

Ở trên, bạn vừa theo dõi một ứng dụng cơ bản mà Server HTTP request đến một trang chủ. Định tuyến liên quan đến cách xác định cụ thể một ứng dụng trả về một Client Request đển một Endpoint cụ thể, đó là một đường dẫn URL và trả về một phương thức HTTP request(GET, POST và các phương thức khác).

Dựa và chương trình Helloword trên, ta sẽ phát triển thêm một số chức năng bổ sung để xử lý các HTTP request.

var express = require('express');

var app = express();

*// Phuong thuc get() phan hoi mot GET Request ve Homepage*

app.get('/', function (req, res) {

   console.log("Nhan mot GET Request ve Homepage");

   res.send('Hello GET');

})

*// Phuong thuc post() phan hoi mot POST Request ve Homepage*

app.post('/', function (req, res) {

   console.log("Nhan mot POST Request ve Homepage");

   res.send('Hello POST');

})

*// Phuong thuc delete() phan hoi mot DELETE Request ve /del\_user page.*

app.delete('/del\_user', function (req, res) {

   console.log("Nhan mot DELETE Request ve /del\_user");

   res.send('Hello DELETE');

})

*// Phuong thuc nay phan hoi mot GET Request ve /list\_user page.*

app.get('/list\_user', function (req, res) {

   console.log("Nhan mot GET Request ve /list\_user");

   res.send('Page Listing');

})

*// Phuong thuc nay phan hoi mot GET Request ve abcd, abxcd, ab123cd, ...*

app.get('/ab\*cd', function(req, res) {

   console.log("Nhan mot GET request ve /ab\*cd");

   res.send('Page Pattern Match');

})

var server = app.listen(8081, function () {

  var host = server.address().address

  var port = server.address().port

  console.log("Ung dung Node.js dang lang nghe tai dia chi: http://%s:%s", host, port)

})

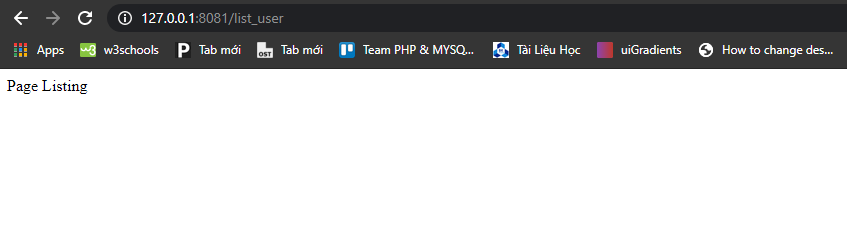
Kết quả:

Ung dung Node.js dang lang nghe tai dia chi: http:*//:::8081*

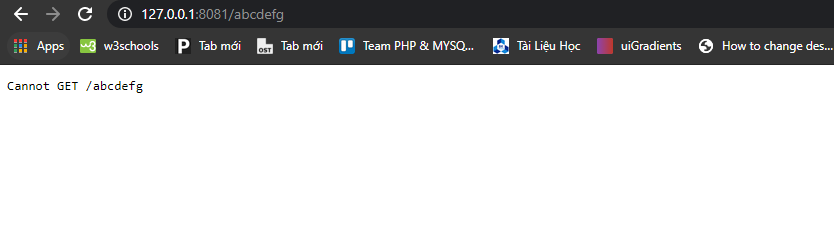
Bây giờ, bạn có thể thử các Request khác tại địa chỉ http:127.0.0.1:8081

Để xem kết quả tạo bởi server.js. Dưới đây là vài màn hình hiển thị các response khác nhau với các URL khác nhau.

Màn hình kết quả cho <http://127.0.0.1:8081/list_user>



Màn hình kết quả cho <http://127.0.0.1:8081/abcdefg>



Đối với các file tĩnh

Express cung cấp các tiện ích lớp trung gian **express.static** để phục vụ cho các file tĩnh như hình ảnh, CSS, Javascript, ...

Về cơ bản, bạn chỉ cần truyền tên thư mục nơi bạn giữ các file này, **express.static** sẽ sử dụng file đó một cách trực tiếp. Ví dụ, bạn muốn giữ hình ảnh, CSS và Javascript trong thư mục public, bạn có thể làm như sau:

app.use(express.static('public'));

Giả sử mình giữ một vài hình ảnh trong thư mục con **public/images** như sau:

node\_modules

server.js

public/

public/images

public/images/logo.png

Sửa đổi ứng dụng "Hello Word" trên để thêm một số tính năng bổ sung để xử lý các file tĩnh:

var express = require('express');

var app = express();

app.use(express.static('public'));

app.get('/', function (req, res) {

   res.send('Hello World');

})

var server = app.listen(8081, function () {

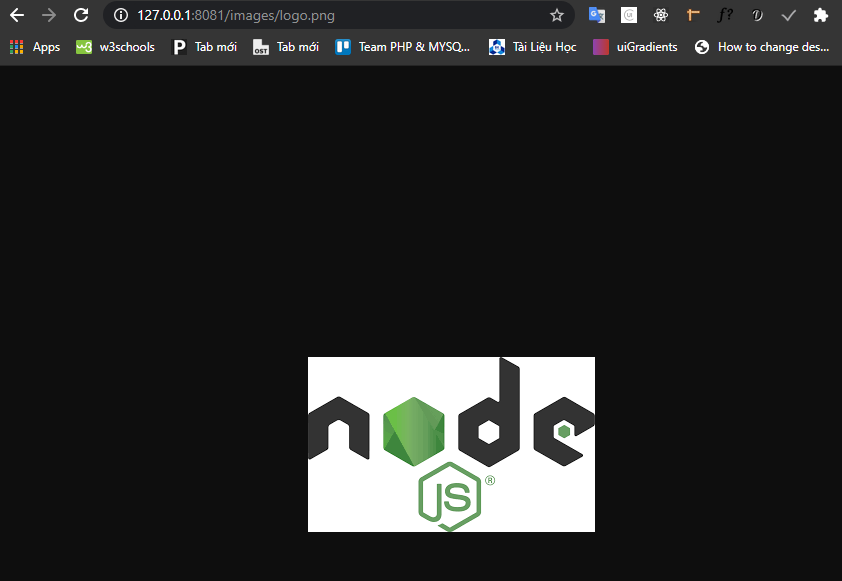
  var host = server.address().address

  var port = server.address().port

  console.log("Ung dung Node.js dang lang nghe tai dia chi: http://%s:%s", host, port)

})

Lưu phần code và chay file rồi mở trình duyệt và gõ địa chỉ sau http://127.0.0.1:8081/images/logo.png để xem kết quả.



Ví dụ phương thức Get

Dưới đây là một ví dụ đơn giản để truyền 2 giá trị sử dụng HTML FORM với phương thức GET. Mình sẽ sử dụng **process\_get** trong server.js để xử lí phần input.

<html>

<body>

<form action="http://127.0.0.1:8081/process\_get" method="GET">

First Name: <input type="text" name="first\_name">  <br>

Last Name: <input type="text" name="last\_name">

<input type="submit" value="Submit">

</form>

</body>

</html>

Lưu đoạn code trên trong index.html và sửa đổi server.js như sau:

var express = require('express');

var app = express();

app.use(express.static('public'));

app.get('/index.html', function (req, res) {

   res.sendFile( \_\_dirname + "/" + "index.html" );

})

app.get('/process\_get', function (req, res) {

*// Chuan bi output trong dinh dang JSON*

   response = {

       first\_name:req.query.first\_name,

       last\_name:req.query.last\_name

   };

   console.log(response);

   res.end(JSON.stringify(response));

})

var server = app.listen(8081, function () {

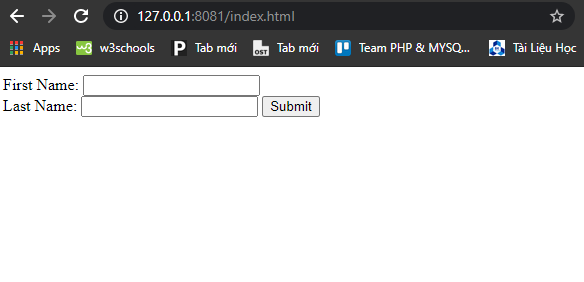
  var host = server.address().address

  var port = server.address().port

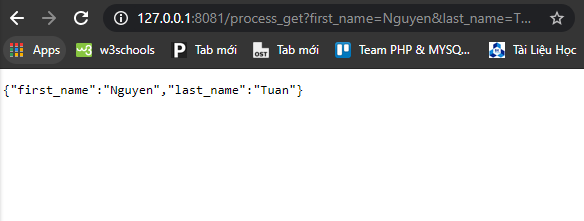
  console.log("Ung dung Node.js dang lang nghe tai dia chi: http://%s:%s", host, port)

})

Mở trình duyệt và gõ địa chỉ *http://127.0.0.1:8081/index.html* để xem kết quả:



Bây giờ, bạn có thể nhập First Name và Last Name, nhấn nút Submit rồi xem kết quả:



Ví dụ về phương thức POST

Dưới đây là ví dụ cơ bản để truyền 2 giá trị sử dụng HTML form bởi sử dụng phương thức POST. Mình sẽ sử dụng **process\_post** trong server.js để xử lí phần input này.

var express = require('express');

var app = express();

var bodyParser = require('body-parser');

*// Tao mot parser co dang application/x-www-form-urlencoded*

var urlencodedParser = bodyParser.urlencoded({ extended: false })

app.use(express.static('public'));

app.get('/index.html', function (req, res) {

   res.sendFile( \_\_dirname + "/" + "index.html" );

})

app.post('/process\_post', urlencodedParser, function (req, res) {

*// Chuan bi output trong dinh dang JSON*

   response = {

       first\_name:req.body.first\_name,

       last\_name:req.body.last\_name

   };

   console.log(response);

   res.end(JSON.stringify(response));

})

var server = app.listen(8081, function () {

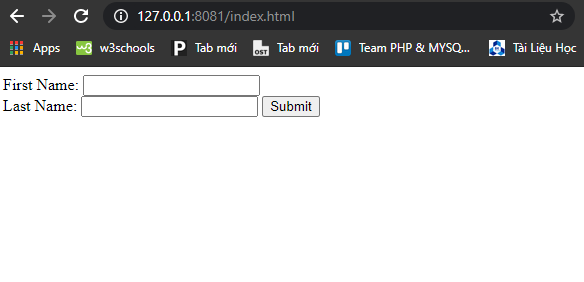
  var host = server.address().address

  var port = server.address().port

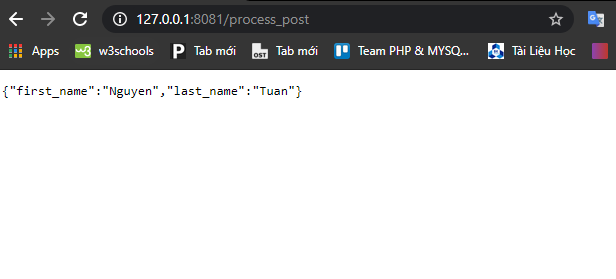
  console.log("Ung dung Node.js dang lang nghe tai dia chi: http://%s:%s", host, port)

})

Mở trình duyệt và gõ địa chỉ *http://127.0.0.1:8081/index.html* để xem kết quả:



Bây giờ, bạn có thể nhập First Name và Last Name, nhấn nút Submit rồi xem kết quả:



Ví dụ về File Upload

Dưới đây là HTML code để tạo một File Upload Form. Form này có những thuộc tính thiết lập đến phương thức **POST** và thuộc tính mã hóa để thiết lập **multipart/form-data**.

<html>

<head>

<title>File Uploading Form</title>

</head>

<body>

<h3>File Upload:</h3>

Select a file to upload: <br />

<form action="http://127.0.0.1:8081/file\_upload" method="POST"

      enctype="multipart/form-data">

<input type="file" name="file" size="50" />

<br />

<input type="submit" value="Upload File" />

</form>

</body>

</html>

Lưu đoạn code trên trong index.htm và sửa đổi server.js như sau:

var express = require('express');

var app = express();

var fs = require("fs");

var bodyParser = require('body-parser');

var multer  = require('multer');

app.use(express.static('public'));

app.use(bodyParser.urlencoded({ extended: false }));

app.use(multer({ dest: '/tmp/'}));

app.get('/index.htm', function (req, res) {

   res.sendFile( \_\_dirname + "/" + "index.htm" );

})

app.post('/file\_upload', function (req, res) {

   console.log(req.files.file.name);

   console.log(req.files.file.path);

   console.log(req.files.file.type);

   var file = \_\_dirname + "/" + req.files.file.name;

   fs.readFile( req.files.file.path, function (err, data) {

        fs.writeFile(file, data, function (err) {

*if*( err ){

              console.log( err );

         }*else*{

               response = {

                   message:'File duoc upload thanh cong!',

                   filename:req.files.file.name

              };

          }

          console.log( response );

          res.end( JSON.stringify( response ) );

       });

   });

})

var server = app.listen(8081, function () {

  var host = server.address().address

  var port = server.address().port

  console.log("Ung dung Node.js dang lang nghe tai dia chi: http://%s:%s", host, port)

})

Bây giờ bạn mở trình duyệt và gõ địa chỉ *http://127.0.0.1:8081/index.htm* để xem kết quả:

Ví dụ với Cookie

Bạn có thể gửi các Cookie tới Node.js Server. Ví dụ dưới đây minh họa cách in tất cả Cookie được gửi bởi Client.

var express      = require('express')

var cookieParser = require('cookie-parser')

var app = express()

app.use(cookieParser())

app.get('/', function(req, res) {

  console.log("Cookies: ", req.cookies)

})

app.listen(8081)