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

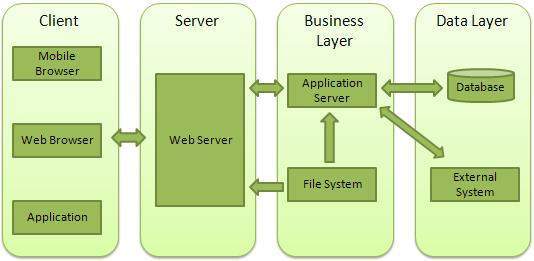
**Web Server**

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>

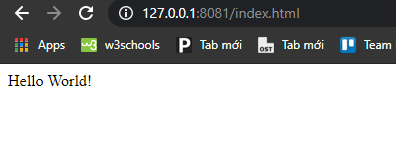
</html>

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

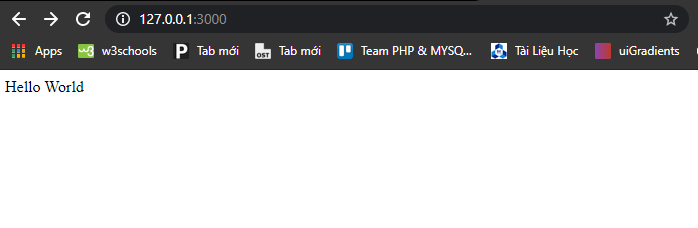
  var port = server.address().port

  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)

})

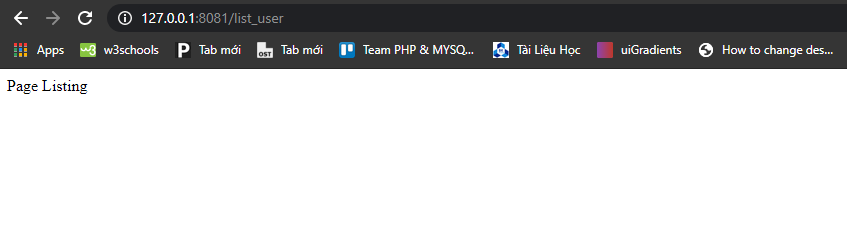
Kết quả:

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

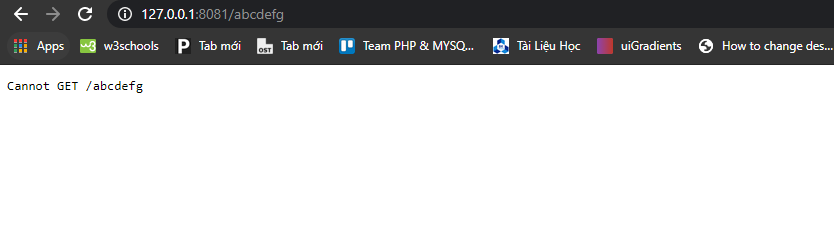
Bây giờ, bạn có thể thử các Request khác tại địa chỉ http:127.0.0.1:8081

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 () {

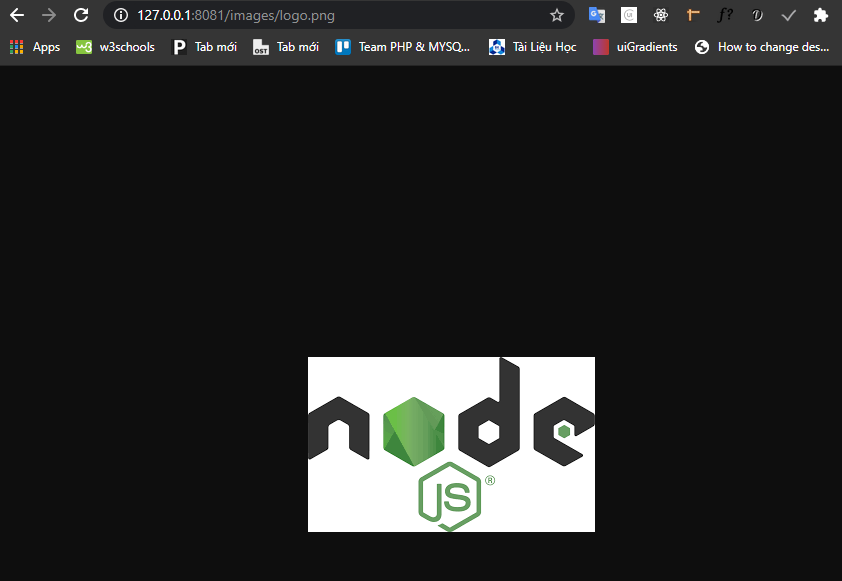
  var host = server.address().address

  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 () {

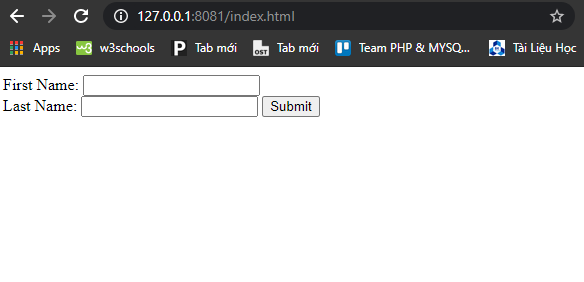
  var host = server.address().address

  var port = server.address().port

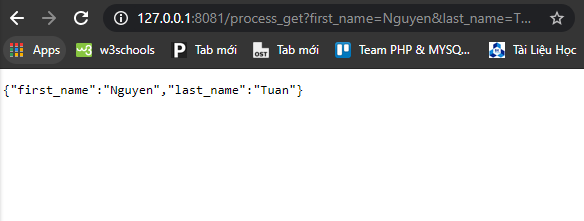
  console.log("Ung dung Node.js dang lang nghe tai dia chi: http://%s:%s", host, port)

})

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 () {

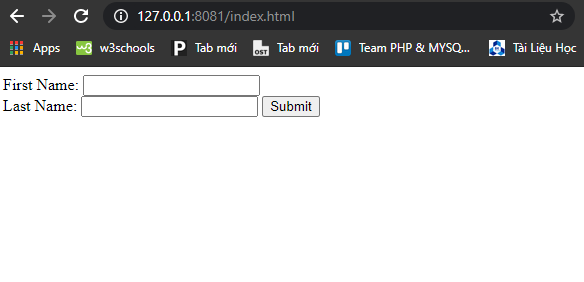
  var host = server.address().address

  var port = server.address().port

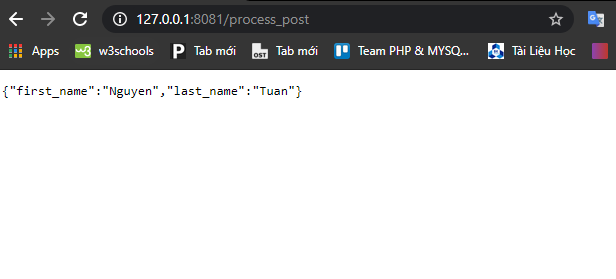
  console.log("Ung dung Node.js dang lang nghe tai dia chi: http://%s:%s", host, port)

})

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)