TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**TIỂU LUẬN MÔN LẬP TRÌNH WEB VÀ ỨNG DỤNG**

**NodeJS**

*Người hướng dẫn*: **Thầy MAI VĂN MẠNH**

*Người thực hiện*: **LÊ HOÀNG HUY – 51800397**

**TRẦN TRỊNH HIẾU – 51800772**

**TRẦN LUÂN – 51800436**

**LÊ QUANG LINH – 51800423**

Lớp **: 18050202 - 18050201**

Khoá  **: 22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**TIỂU LUẬN MÔN LẬP TRÌNH WEB VÀ ỨNG DỤNG**

**NodeJS**

Người hướng dẫn: **Thầy MAI VĂN MẠNH**

Người thực hiện: **LÊ HOÀNG HUY**

**TRẦN TRỊNH HIẾU**

**TRẦN LUÂN**

**LÊ QUANG LINH**

Lớp **: 18050202 – 18050201**

Khoá  **: 22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

LỜI CẢM ƠN

Đây là phần tác giả **tự viết** ngắn gọn, thể hiện sự biết ơn của mình đối với những người đã giúp mình hoàn thành Luận văn/Luận án. Tuyệt đối không sao chép theo mẫu những “lời cảm ơn” đã có.

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi / chúng tôi và được sự hướng dẫn của Thầy Mai Văn Mạnh. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày tháng năm*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Nguyễn Văn B*

*Trần Văn C*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

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

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

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

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

Trình bày tóm tắt vấn đề nghiên cứu, các hướng tiếp cận, cách giải quyết vấn đề và một số kết quả đạt được, những phát hiện cơ bản trong vòng 1 -2 trang.

MỤC LỤC

[LỜI CẢM ƠN i](#_Toc54133059)

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii](#_Toc54133060)

[TÓM TẮT iv](#_Toc54133061)

[MỤC LỤC 1](#_Toc54133062)

[CHƯƠNG 1 – GIỚI THIỆU ĐỀ TÀI 3](#_Toc54133063)

[1.1 Đặt vấn đề 3](#_Toc54133064)

[1.2 Giới thiệu đề tài 3](#_Toc54133065)

[1.3 Ý nghĩa 3](#_Toc54133066)

[CHƯƠNG 2 – CƠ SỞ LÝ THUYẾT 4](#_Toc54133067)

[2.1 Khái niệm 4](#_Toc54133068)

[2.2 Đặc tính 4](#_Toc54133069)

[2.2.1 Không đồng bộ 4](#_Toc54133070)

[2.2.2 Tốc độ chạy rất nhanh 5](#_Toc54133071)

[2.2.3 Module 5](#_Toc54133072)

[2.2.4 Không đệm 5](#_Toc54133073)

[2.2.5 Global Scope 6](#_Toc54133074)

[2.2.6 Community 6](#_Toc54133075)

[2.3 Các object cơ bản 7](#_Toc54133076)

[2.3.1 Object 7](#_Toc54133077)

[2.3.2 Console 7](#_Toc54133078)

[2.3.3 Process 8](#_Toc54133079)

[2.3.4 Buffers 10](#_Toc54133080)

[2.3.5 Sự kiện (Event). 12](#_Toc54133081)

[2.3.6 Luồng ( Streams ): 13](#_Toc54133082)

[2.3.7 File System: 15](#_Toc54133083)

[2.3.8 HTTP. 17](#_Toc54133084)

[2.4 Các ứng dụng được xây dựng trên nền Node.js 17](#_Toc54133085)

[2.4.1 Ứng dụng đầu tiên. 17](#_Toc54133086)

[2.4.2 HTTP Server. 18](#_Toc54133087)

[2.4.3 Xử lý các tham số URL: 18](#_Toc54133088)

[2.4.4 Xử lý các tham số URL: 19](#_Toc54133089)

[2.4.6 Ứng dụng tính kết quả biểu thức cho trước. 20](#_Toc54133090)

[2.4.7 Ứng dụng webchat: 21](#_Toc54133091)

[2.5 Mối liên hệ giữa JavaScript và NodeJS 23](#_Toc54133092)

[CHƯƠNG 3 – CƠ SỞ LÝ THUYẾT VÀ THỰC NGHIỆM 24](#_Toc54133093)

[3.1 24](#_Toc54133094)

[3.2 24](#_Toc54133095)

[CHƯƠNG 4 – TỔNG KẾT 25](#_Toc54133096)

[4.1 Kết quả đạt được 25](#_Toc54133097)

[4.2 Những mặt hạn chế 25](#_Toc54133098)

[4.3 Hướng phát triển trong tương lai 25](#_Toc54133099)

CHƯƠNG 1 – GIỚI THIỆU ĐỀ TÀI

* 1. Đặt vấn đề

Khi lướt Facebook thì mỗi khi bạn bình luận hay chia sẽ một bài viết nào đó thì ngay lập tức chủ bài viết và những người đã bình luận trên đó sẽ nhận được thông báo là bạn đã bình luận bài viết đó. Hoặc là ứng dụng chat thời gian thực, khi bạn nhắn tin cho ai đó, thì người đó lập tức nhận được tin nhắn của bạn, và thông báo cho bạn là người đó đã đọc tin nhắn. Để giải thích cho vấn đề trên thì bài tiểu luận dưới đây sẽ giúp các bạn hiểu được phần nào về Nodejs.

* 1. Giới thiệu đề tài

Ở bài tiểu luận này nhóm em xin giới thiệu về những khái niệm cơ bản trong Node.js và ứng dụng của Node js trong phần mềm chat thời gian thực.

* 1. Ý nghĩa

Bài tiểu luận sẽ cung cấp cho người đọc những kiến thức nền tảng về Node js và nắm được nguyên tắc xây dụng một ứng dụng chat thời gian thực.

CHƯƠNG 2 – CƠ SỞ LÝ THUYẾT

2.1 Khái niệm

NodeJS là một mã nguồn được xây dựng dựa trên nền tảng Javascript V8 Engine, được sử dụng để xây dựng các ứng dụng web như các trang video clip, các forum và đặc biệt là trang mạng xã hội phạm vi hẹp.

NodeJS được tạo bởi Ryan Dahl từ năm 2009, và phát triển dưới sự bảo trợ của Joyent. Mục tiêu ban đầu của Dahl là làm cho trang web có khả năng push như trong một số ứng dụng web như Gmail. Sau khi thử với vài ngôn ngữ Dahl chọn Javascript vì một API Nhập/Xuất không đầy đủ. Điều này cho phép anh có thể định nghĩa một quy ước Nhập/Xuất điểu khiển theo sự kiện, non-blocking.

NodeJS có thể chạy trên nhiều nền tảng hệ điều hành khác nhau từ Window cho tới Linux, OS X. Nó cũng cung cấp một bộ thư viện phong phú Javascript Module khác nhau giúp đơn giản hóa việc lập trình và giảm thời gian ở mức thấp nhất. NodeJS được viết bằng JavaScript, sử dụng kỹ thật điều khển theo sự kiện, nhập/xuất không đồng bộ để tối tiểu tổng chi phí và tối đại khả năng mở rộng.

Khi nhắc đến NodeJS thì phải nghĩ ngay đến Realtime, Realtime là xử lý giao tiếp từ client tới máy chủ theo thời gian thực.

2.2 Đặc tính

2.2.1 Không đồng bộ

Tất cả các API của NodeJS đều không đồng bộ (none-blocking), nó chủ yếu dựa trên nền của NodeJS Server và chờ đợi Server trả dữ liệu về. Việc di chuyển máy chủ đến các API tiếp theo sau khi gọi và cơ chế thông báo các sự kiện của Node.js giúp máy chủ để có được một phản ứng từ các cuộc gọi API trước (Realtime).

Là lợi thế nếu bạn đã quen thuộc với các phương pháp lập trình không đồng bộ. Tất cả các hàm trong Node.js là không đồng bộ. Do đó, tất cả chạy như các block thread thông thường thay vì chạy nền. Đây là điều quan trọng nhất để nhớ về Node.js.

Ví dụ, nếu bạn đang đọc một tập tin trên hệ thống tập tin, bạn phải chỉ định một chức năng gọi lại đó là thực hiện khi đã hoàn thành các hoạt động đọc.

2.2.2 Tốc độ chạy rất nhanh

NodeJS được xây dựng dựa vào nền tảng V8 Javascript Engine nên việc thực thi chương trình rất nhanh.

2.2.3 Module

Node.js sử dụng một kiến trúc mô-đun để đơn giản hóa việc tạo ra các ứng dụng phức tạp. Mô-đun giống như các thư viện trong C, hoặc các đơn vị trong Pascal. Mỗi module có chứa một tập hợp các chức năng liên quan đến "đối tượng" của các mô-đun. Ví dụ, các mô-đun http chứa các chức năng cụ thể cho HTTP, Node.js cung cấp một vài mô-đun cơ bản để giúp bạn truy cập các tập tin trên hệ thống tập tin, tạo ra trình điều khiển server HTTP và TCP /. UDP và thực hiện các chức năng hữu ích khác.

Để gọi một modul thật dễ dàng, chỉ cần gọi hàm require() như sau :

*Var http = require(‘http’);*

***Hàm require()*** trả về tham chiếu đến các module quy định. Trong trường hợp của mã này, một tham chiếu đến các module http được lưu trữ trong biển http. Trong đoạn code trên, ta đã truyền tên của module vào trong hàm require(). Việc này chỉ định cho Node.js tìm trong thư mục node\_modules module tương ứng để thực hiện. Nếu Node không thấy module tương ứng trong thư mục thì nó sẽ tìm trên global module cache. Bạn cũng có thể chỉ định một module qua một file vật lý qua đường dẫn tương đối hay tuyệt đối như sau:

*Var myModule = require('./myModule.js');*

Module được đóng gói từng phần mã. Đoạn mã nằm trong một mô-đun chủ yếu là private - có nghĩa là các chức năng và biến được định nghĩa trong họ chỉ có thể truy cập từ bên trong của các mô-đun.

2.2.4 Không đệm

NodeJS không đệm bất kì một dữ liệu nào và các ứng dụng này chủ yếu là đầu ra dữ liệu.

2.2.5 Global Scope

Node là một môi trường chạy javascrip với google V8 engine do đó hỗ trợ chạy được ở server side. Do đó bạn cũng nên tuân thủ các kinh nghiệm mà bạn có trong lập trình với các ứng dụng client-side. Ví dụ khi tạo các biến global trong Node không phải lúc nào cũng có thể tạo. Nhưng bạn có thể tạo dễ dàng các biến hoặc hàm global với cách bỏ từ khóa var trước các biến như sau:

*globalVariable = 1; global Function = function () {...};*

2.2.6 Community

Cộng đồng phát triển Node.js chủ yếu tập trung ở hai nhóm google : NodeJS và NodeJS-dev, một kênh IRC là #node.js trên mạng freenode. Có một hội thảo về Node.js là NodeConf được tổ chức thường niên. Hiện nay Node.js được sử dụng bởi nhiều công ty trong đó có Linkedin, Microsoft và Walmart.

2.3 Các object cơ bản

2.3.1 Object

Như chúng ta đã biết, hệ thống mô-đun của node không khuyến khích việc sử dụng biển toàn cục, tuy nhiên node cung cấp một globals quan trọng để sử dụng.

Việc đầu tiên và quan trọng nhất là tiến trình global, cho thấy nhiều thao tác như quá trình truyền tín hiệu, xuất cảnh, proccess id (pid), và nhiều hơn nữa. Globals khác, chẳng hạn như console objects được cung cấp cho những người sử dụng để viết JavaScript cho trình duyệt web.

* + 1. Console

Các **console objects** sử dụng một số lệnh được sử dụng đề xuất thông tin để stdout hoặc stder. Chúng là các lệnh như: *console.log (data), [...]*

Phương pháp **console objects** được sử dụng thường xuyên nhất là console.log(), mà chỉ đơn giản là viết cho stdout và gắn một nguồn cấp dữ liệu dòng (1 n console.log (wahoo ');

*// => Wahoo console.log ((foo bar'); 11 => [object Object]*

Còn một lệnh có chức năng như **console.log()** đó là:

*console.info (). console.error ([data], [...]*

Giống hệt nhau để console.log(), tuy nhiên viết cho stderr. console.error 'kết nối cơ sở dữ liệu không thành công !};

Còn một lệnh có chức năng như console.error ( ) đó là: console, warn (). console.dir (obj)

Sử dụng phương pháp inspect() của mô-đun svs khá-in các đối tượng đến stdout. console.dir ((foo: 'bar'));

*// => (Foo: 'bar'} console.assert (expression, [message])*

Nếu expression bị đánh giá là có giá trị là false thì AssertionError sẽ đưa ra message được cho.

*Console.assert (connected, “Cơ sở dữ liệu kết nối không thành công ‘}; console.time(label)*

Đánh dấu thời gian bắt đầu. console.timeEnd (label)

Thời gian kết thúc, được ghi vào đầu ra.

Ví dụ:

*console.time ('100-elements'); for (var i = 0; i < 100; i++)*

*console.timeEnd('100-elements');*

*console.trace(label)*

2.3.3 Process

Các process object gắn liền với goodies. Trước tiên, chúng ta sẽ có một cái nhìn tại một số thuộc tỉnh cung cấp thông tin về node process đó: process.version

Chuỗi phiên bản nút, ví dụ:

*console.log('Version:' + process.version); // Version v0.8.16 process.execPath*

Đường dẫn đến thư mục thực thi chính của chương trình "/ 4s / local / bin / node". process.platform

*Các nền tảng bạn đang sử dụng. Ví dụ, “darwin”.*

Process.pid

**Các process ID**. process.stdout)

Một luồng có thể ghi được đến stdout.

Ví dụ:

Định nghĩa về console.log()

*. console.log - function (d) { process.stdout.write (d + '\n');*

**Process.stderro** : tương tự như process,stdout () nhưng ở đây là ghi đến stdear, process.stderr () và process.sthout ) là không giống như luồng khác trong Node, khi viết chúng thường bị blocking.

Chúng bị blocking trong trường hợp mà chúng liên quan đến các tập tin thường xuyên hoặc mô tả tập tin TTY. Trong trường hợp chúng liên quan đến các pipes, chúng không bị blocking như những luồng khác. process.stdin

Một luồng có thể đọc được cho stdin. Các dòng stdin bị tạm dừng theo mặc định, do đó, người ta phải gọi process.stdin.resume () để đọc từ nó.

Ví dụ mở đầu vào chuẩn và lắng nghe cả hai sự kiện:

*process.stdin.resume (); process.stdin.setEncoding ('utf8');*

*process.stdin.on ('data', function (chunk) process.stdout.write ('data: ' + chunk);*

*process.stdin.on (end', function() { process.stdout.write(' end');*

*process.cwd()*

Trả về thư mục làm việc hiện tại. Ví dụ: cd - && node

*node> process.cwd() "/Users/tj" process.chdir*

Thay đổi thư mục làm việc hiện tại.

*process.chdir('/ foo'); process.getuid ()*

Trả về số user ID của process đang chạy. process.setuid (

Thiết lập user ID có hiệu lực cho quá trình đang chạy. Phương pháp này chấp nhận cả một số ID, cũng như một chuỗi. Ví dụ cả hai process.setuid (501), và process.setuid ('tj') đều hợp lệ. process.getgid ()

Trả về số group ID của process đang chạy. process.setgid ()

Tương tự như process.setuid () tuy nhiên được sử dụng trong group, cũng chấp nhận một số giá trị hoặc chuỗi đại diện. Ví dụ, process.setgid (20) hoặc process.setgid('www'). process.chdir (directory)

Thay đổi thư mục làm việc hiện tại của process hoặc đưa một ngoại lệ nếu thất 6:

*console.log('Starting directory: '+ process.cwd()); try { process.chdir ('/tmp'); console.log ('New directory:' + process.cwd 0));*

*catch (err) {*

*Errno*

Process object này lưu trữ của các con số báo hiệu lỗi tại máy chủ, tham khảo những gì bạn sẽ tìm thấy trong C-land. Ví dụ, Process,EPERM đại diện cho một lỗi dựa trên sự cho phép, trong khi process.ENDENT đại diện cho một tập tin hoặc thư mục bị thiếu. Thông thường đây là những được sử dụng trong các ràng buộc để thu hẹp khoăng cách giữa C++ và JavaScript, nhưng chúng hữu ích cho việc xử lý các trường hợp ngoại lệ như:

*if (err.errno === process.ENOENT) { // Display a 404 "Not Found" page*

*} else { // Display a 500 "Internal Server Error" page*

2.3.4 Buffers

Cơ bản JavaScript là Unicode thân thiện, nhưng không phải với dữ liệu nhị phân. Khi giao tiếp với luồng TCP hoặc hệ thống tập tin, liệu nhị phân cần thiết để xử lý các luồng octet. Node cung cấp một số phương pháp cho việc khai thác, tạo và sử dụng luồng octet.

Để xử lý các dữ liệu nhị phân, node cung cấp cho chúng ta với các đối tượng toàn cục. Buffer là tương tự như một mảng các số nguyên, nhưng tương ứng với việc cấp phát bộ nhớ thô bên ngoài V8 heap. Buffer không thể được thay đổi kích cỡ. Có một số cách để xây dựng một trường hợp bộ đệm, và nhiều cách bạn có thể thao tác dữ liệu của nó.

Chuyển đổi giữa Buffers và các đối tượng chuỗi JavaScript đòi hỏi một phương pháp mã hóa rõ ràng. Dưới đây là chuỗi các bảng mã khác nhau:

**Ascii- 7 bit** dữ liệu ASCII duy nhất. Phương pháp mã hóa này là rất nhanh chóng, và sẽ loại bỏ các bit cao nếu thiết lập. Lưu ý rằng việc mã hóa này chuyển đổi một kỷ tự null ( 0' hoặc '\ u0000") vào 0x20 (mã ký tự của một không gian). Nếu bạn muốn chuyển đổi một ký tự null vào 0x00, bạn nên sử dụng 'utf8'.

**'Utf8'** - Nhiều byte mã hóa ký tự Unicode. Nhiều trang web và các định dạng tài liệu khác sử dụng UTF-8.

**'Utf16le** - 2 hoặc 4 byte, ký tự Unicode mã hóa ít về cuối. Các cặp đại diện (U 10.000 FFFF 10 U) được hỗ trợ.

**‘UCS2'** - Tương tự 'utf16le.

**‘Base64'** - Mã hỏa chuỗi Base64.

**'binary**- Một cách mã hóa dữ liệu nhị phân thành chuỗi bằng cách sử dụng 8 bit đầu tiên của mỗi ký tự. Phương pháp mã hóa này bị phản đối và nên tránh sử dụng các đối tượng bộ đệm nếu có thể. Mã hóa này sẽ được loại bỏ trong các phiên bản tương lai của Node.

**'Hex'** - Mã hóa mỗi byte là hai ký tự thập lục phân. Buffer cũng có thể được sử dụng xem măng kiều và DataViews.

*var buff = new Buffer (4) ;*

*var uil6 = new Vint16Array (buff);*

*var view = new DataView (buff);*

*ui16 [0] = 1;*

*uil6[1] = 2;*

*console.log(buff);*

*view.setInt16(0, 1);*

*// set big-endian int16 at byte offset D view.setInt16(2, 2, true);*

*// set little-endian int16 at byte offset 2 console.log(buff);*

*// <Buffer 01 00 02 00>*

*// <Buffer 00 01 02 00>*

Cách đơn giản nhất để xây dựng một bộ đệm từ một chuỗi chỉ đơn giản là sử dụng chuỗi như là tham số đầu tiên. Như bạn có thể nhìn thấy trong đăng nhập, bây giờ chúng ta có một đối tượng bộ đệm có chứa 5 byte dữ liệu được đại diện trong hệ thập lục phân:

*var hello = new Buffer ("Hello');*

*console.log (hello);*

*// => <Buffer 48 65 6c 6c 6f> console.log (hello.toString());*

*// => "Hello"*

Theo mặc định, mã hóa là "utf8", nhưng điều này có thể được thay đổi bằng cách đi qua một chuỗi như là đối số thứ hai. Ví dụ, dấu chấm lửng dưới đây sẽ được in stdout như kí tự "&" khi trong bảng mã " ascii":

*var buf = new Buffer('-');*

*console.log(buf.toString());*

*// => -1 var buf = new Buffer('a-i', 'ascii');*

*console.log(buf.toString());*

2.3.5 Sự kiện (Event).

Khái niệm về một "sự kiện" là rất quan trọng trong node, và được sử dụng rất nhiều trong suốt module chính của chương trình và module của bên thứ 3. Module sự kiện chính của Node cung cấp cho chúng ta với một hàm tạo, EventEmitter.

* 1. **EventEmitter:**

Thông thường một đối tượng kế thừa từ EventEmitter, tuy nhiên ví dụ nhỏ dưới đây minh họa API. Đầu tiên chúng ta tạo ra một emitter, sau đó chúng ta có thể xác định bất kỳ số lượng callbacks sử dụng emitter.on () phương pháp, mà chấp nhận tên của các sự kiện và các đối tượng tùy ý thông qua như là dữ liệu.

Khi emitter.emit() được gọi, chúng ta chi required để truyền các tên sự kiện, theo sau bởi bất kỳ số lượng tham số (trong trường hợp này các chuỗi tên đầu tiên và cuối cùng).  
var EventEmitter - require('events').EventEmitter;

*var emitter = new EventEmitter;  
emitter.on('name', function(first, last) ( console.log(first ! last);});*

*emitter.emit('name', 'tj', 'holowaychuk');*

*emitter.emit('name', 'simon', 'holowaychuk');*

* 1. **Kế thừa từ EventEmitter ( Inheriting From EventEmitter ):**

Được sử dụng phổ biến và thiết thực của EventEmitter là tính kế thừa từ nó. Điều này có nghĩa là chúng ta có thể giữ nguyên EventEmitter nguyên mẫu mà không bị ảnh hưởng trong khi sử dụng API của nó đối với phương tiện riêng của chúng ta.

Để làm như vậy, chúng ta bắt đầu bằng cách xác định các hàm khởi tạo Dog, trong đó tất nhiên sẽ bark từ thời gian đến thời gian (còn được biết đến như một sự kiện).

*var EventEmitter = require('events').EventEmitter;*

*function Dog (name)*

*this.name = name;*

* 1. **Loại bỏ các sự kiện lắng nghe (Removing Event Listeners):**

Như chúng ta đã biết, lắng nghe sự kiện chỉ đơn giản là hàm đó được gọi khi chúng ta emit() một sự kiện. Chúng ta có thể loại bỏ những người nghe bằng cách dùng lệnh removeListener (type, callback), mặc dù điều này không được dùng thường xuyên.

Trong ví dụ dưới đây, chúng ta phát ra thông báo "foo bar" mỗi 300 mili giây, trong đó có một cuộc gọi lại của console.log(). Sau 1000 mili giây, chúng ta gọi removeListener() với các đối số tương tự mà tôi đã thông qua on() ban đầu. Chúng ta cũng có thể sử dụng removeAllListeners (type), trong đó loại bỏ tất cả các người nghe được đăng ký type nhất định.

*var EventEmitter - require('events'). Event Emitter;*

*var emitter - new EventEmitter; emitter.on('message', console.log);*

*setInterval (function() { emitter.emit('message', 'foo bar');}, 300);*

*setTimeout(function() { emitter.removeListener('message', console.log);*

*}, 1000);*

2.3.6 Luồng ( Streams ):

Streams là một khái niệm quan trọng trong nút. Các luồng API là một cách duy nhất để xử lý luồng giống như dữ liệu. Ví dụ, dữ liệu có thể được xem trực tiếp một tập tin, trực tiếp vào một socket để đáp ứng một HTTP request, hoặc trực tiếp từ một nguồn chỉ cho đọc như stdin. Để bây giờ, chúng ta sẽ tập trung vào các API, để lại các chi tiết cụ thể luồng chương sau.

1. **Readable Streams:**

Readable Streams được xem như một HTTP request kế thừa từ EventErmitter để lộ dữ liệu đến qua các sự kiện. Việc đầu tiên của những sự kiện này là sự kiện dữ liệu, là một đoạn tùy ý của các dữ liệu được truyền đi để xử lý sự kiện như là một trường hợp đệm ( Buffer instance).

*req.on('data', function(buf) { 3/ Làm gì đó với Buffer});*

Một sự kiện quan trọng khác là kết thúc, đại diện cho sự kết thúc của dữ liệu sự kiện. Ví dụ, đây là một HTTP echo server, chỉ đơn giãn là "simply" các request body data thông qua các response. Vì vậy, nếu chúng ta POST "hello world", response của chúng ta sẽ là " hello world".

1. **Writable Stream:**

Một lớp cơ sở cho việc tạo ra các Writable Stream. Tương tự như Readable Stream, bạn có có thể tạo ra các lớp con bằng cách ghi đè không đồng bộ khi sử dụng câu lệnh write(chunk, cb).

*stream.writable*

Giá trị boolean được mặc định đó là true, nhưng sẽ thành false sau khi lỗi xảy ra hoặc end () {destroy() được gọi. stream. Write (string, (encoding)

Viết chuỗi với encoding cho luồng. Trả về true nếu chuỗi đã được bỏ vào bộ đệm kernel. Trả về false để cho biết rằng bộ đệm kernel đã đầy, và dữ liệu sẽ được gửi đi trong tương lai. Sự kiện 'drain' sẽ cho biết khi nào bộ đệm kernel là rỗng một lần nữa. Việc mã hóa mặc định 'utf8'. stream. Write (buffer)

Tương tự như trên, ngoại trừ với một bộ đệm thô. stream.end()

Kết thúc dòng với EOF hoặc FIN. Cuộc gọi này sẽ cho phép hàng đợi ghi dữ liệu được gửi trước khi đóng luồng. stream.end (string, encoding)

Gửi chuỗi với mã hóa nhất định và chấm dứt dòng với EOF hoặc FIN. Điều này rất hữu ích để giảm số lượng các gói tin gửi đi. stream.end (buffer)

Tương tự như trên, nhưng với một bộ đệm. stream.destroy 0

Đóng mô tả tập tin cơ bản. Stream là không còn có thể ghi và cũng không thể đọc được. các luồng sẽ không phát ra bất kỳ chi tiết 'data, hoặc sự kiện kết thúc". Bất kỳ hàng đợi dữ liệu ghi sẽ không được gửi đi, các luồng được tải sự kiện 'close' khi tài nguyên của mình đã được xử lý. stream.destroy Soon

Sau khi ghi hàng đợi được giải phóng, đóng tập tin mô tả. destroySoon () vẫn có thể hủy ngay lập tức, miễn là không có dữ liệu còn lại trong hàng đợi để viết.

2.3.7 File System:

Để làm việc với hệ thống tập tin, node cung cấp module "fs". Các lệnh thực thi các hoạt động POSIX, và hầu hết các phương pháp làm việc đồng bộ hoặc không đồng bộ. Chúng ta sẽ xem xét làm thế nào để sử dụng cả hai, sau đó thiết lập lựa chọn tốt hơn.

1. **Làm việc với tập tin hệ thống:**

Cho phép bắt đầu với một ví dụ cơ bản làm việc với tập tin hệ thống. Ví dụ này tạo một thư mục, tạo ra một tập tin bên trong nó, sau đó viết nội dung của tập tin đến console:

*var fs = require('fs');*

*fs.mkdir('./helloDir',0777, function (err) { if (err) throw err;*

*fs.writeFile('./helloDir/message.txt', 'Hello Node', function (err)*

*{ if (err) throw err; console.log('file created with contents:');*

*fs.readFile('./helloDir/message.txt', 'UTF-8' ,function (err, data)*

*{ if (err) throw err; console.log(data); });*

*});*

Rõ ràng trong ví dụ trên, ứng với mỗi callback đều đặt trong callback trước đây được gọi là callbacks chainable. Mô hình này cần được theo sau khi sử dụng phương pháp không đồng bộ, không có đảm bảo rằng các hoạt động sẽ được hoàn thành theo thứ tự họ đang tạo ra. Điều này có thể dẫn đến hành vi không thể đoán trước.

Ví dụ có thể được viết lại để sử dụng một cách tiếp cận đồng bộ

1. **File thông tin:**

Các đối tượng fs.Stats có chứa thông tin về một tập tin hoặc thư mục cụ thể. Điều này có thể được sử dụng để xác định loại đối tượng mà chúng ta đang làm việc. Trong ví dụ này, chúng ta đang nhận được tất cả các đối tượng tập tin trong một thư mục và hiển thị cho dù chúng là một tập tin hoặc một đối tượng thư mục.

*var fs = require('fs');*

*fs.readdir('/etc/', function (err, files)*

*( if (err) throw err;*

*files.forEach( function (file)*

*{ fs.stat('/etc/' + file, function (err, stats)*

*{ if (err) throw err;*

*if (stats.isFile()) ! console.log("%s is file", file);*

*else if (stats.isDirectory () { console.log("%s is a directory", file);*

*console.log('stats:s', JSON.stringify (stats));});*

1. **Xem các tập tin.**
   * Phương pháp fs, watchfile theo dõi một tập tin và thay đổi một sự kiện bất cứ khi nào tập tin được thay đổi.
   * Một tập tin cũng có thể được unwatched bằng cách sử dụng phương pháp gọi fs.unwatchFile. Cách này chỉ nên sử dụng một lần khi tập tin không còn cần được giám sát,

2.3.8 HTTP.

Để sử dụng HTTP server và client phải dùng lệnh require('http').

Các giao diện HTTP trong Node được thiết kế để hỗ trợ nhiều tính năng của các giao thức truyền thống khó sử dụng. Trong đó, có thể là đoạn mã hóa, tin nhắn.

HTTP headers được biểu diễn bởi một đối tượng như thế này:

*{ 'content-length': '123',*

*'content-type': 'text/plain',*

*'connection': 'keep-alive',*

*'accept': \*\*/\*')*

Key được lọwercased và giá trị không được sửa đổi.

*http.STATUS\_CODES*

Một bộ sưu tập của tất cả các mã trạng thái tiêu chuẩn của HTTP response, và mô tả ngắn gọn cho từng cái. Ví dụ, http.STATUS\_CODES [404] === 'Not Found'.

*http.createServer (frequestListener])*

Trả về một đối tượng web server mới, RequestListener là một chức năng được tự động thêm vào sự kiện 'request'.

*http.createClient (sport), [host]*)

* 1. Các ứng dụng được xây dựng trên nền Node.js

Dưới đây là những ứng dụng có thể và nên viết bằng Nodejs: Websocket server: Các máy chủ web socket như là Online Chat, Game Server…, Fast File Upload Client: là các chương trình upload file tốc độ cao, Ad Server: Các máy chủ quảng cáo, Cloud Services: Các dịch vụ đám mây, RESTful API: đây là những ứng dụng mà được sử dụng cho các ứng dụng khác thông qua API, Any Real-time Data Application: bất kỳ một ứng dụng nào có yêu cầu về tốc độ thời gian thực, Micro Services: Ý tưởng của micro services là chia nhỏ một ứng dụng lớn thành các dịch vụ nhỏ và kết nối chúng lại với nhau.

2.4.1 Ứng dụng đầu tiên.

Ứng dụng đầu tiên bạn nên tạo với nodejs là ứng dụng in chữ “Hello World” trên cửa sổ console. Tạo một file với tên hello.js, với nội dung sau:

*console.log('Hello world!');*

Để chạy script, mở cửa sổ command line của node, đưa đường dẫn đến hello.js và chạy lệnh:

*node hello.js*

Bạn sẽ thấy chữ “Hello World” hiện lên trên cửa sổ console.

2.4.2 HTTP Server.

Hãy đến với một ví dụ phức tạp hơn, có thể hơi rối khi bạn tiếp xúc lần đầu. hãy đọc các dòng code sau và các phần ghi chú:

*// Khai báo http module.*

*var http = require("http");*

*| Tạo server. Hàm được thông qua như một tham số được gọi là trên mọi request được tạo ra.*

*| Biến request nắm giữ tất cả các tham số request*

*{} Biến response cho bạn làm bất cứ điều gì với response gửi tới client.*

*http.createServer (function (request, response) {*

*{} Gản listener tại thời điểm kết thúc sự kiện.*

*| Sự kiện này được gọi khi client gửi tất cả dữ liệu và đợi response tu server.*

*request.on("end", function () {*

*// viết headers cho response. 4/ 200 là mã trạng thái của HTTP (điều này có nghĩa là thành công)*

*/ Tham số thứ hai giữ truong header trong đối tượng*

*// Chúng ta đang gửi một văn bản đơn giản, do đó Content-Type nên là text/plain*

*response.writeHead (200, {*

*"Content-Type': 'text/plain'*

*}{ Gửi dữ liệu và kết thúc response. response.end('Hello HTTP!');*

*});*

2.4.3 Xử lý các tham số URL:

Như đã nói trước, ta phải tự làm mọi thứ trong node. Hãy xem code của ví dụ sau: 1/ Khai bảo http module. var http = require("http"), 1/ và url module, nó rất hữu ích trong việc phân tích tham số request.

*url = require("url"); 17 Tao server. http.createServer (function (request, response) {*

*1/ Gán listener và kết thúc sự kiện. request.on('end', function () {*

*|| Phân tích request cho các đối số và lưu chúng trong biến get.*

*// Hàm này phân tích các url từ regest và trả về các đối tượng đại diện.*

*var get - url.parse(request.url, true).query; // Viết headers cho response. response.writeHead (200, (*

*"Content-Type': 'text/plain'*

*1/ Gửi dữ liệu và kiết thúc response.*

*response.end('Here is your data: ' + get['data']);*

*4/ Lắng nghe tại công 8080. }).listen (8080);*

*stream.write('hello\r\n');*

*stream.on('end', function () {*

*stream.end('goodbye\r\n'); });*

*stream.pipe (stream);*

*}).listen (BOBO);*

2.4.4 Xử lý các tham số URL:

Như đã nói trước, ta phải tự làm mọi thứ trong node. Hãy xem code của ví dụ sau: 1/ Khai bảo http module. var http = require("http"), 1/ và url module, nó rất hữu ích trong việc phân tích tham số request.

*url = require("url"); 17 Tao server. http.createServer (function (request, response) {*

*1/ Gán listener và kết thúc sự kiện. request.on('end', function () {*

*|| Phân tích request cho các đối số và lưu chúng trong biến get.*

*// Hàm này phân tích các url từ regest và trả về các đối tượng đại diện.*

*var get - url.parse(request.url, true).query; // Viết headers cho response. response.writeHead (200, (*

*"Content-Type': 'text/plain'*

*1/ Gửi dữ liệu và kiết thúc response.*

*response.end('Here is your data: ' + get['data']);*

*4/ Lắng nghe tại công 8080. }).listen (8080);*

Phần code này dùng phương thức parse() của module url, một module lõi của Nodejs, để convert từ request url thành object. Phương thức trả lại object là phương thức query, sẽ lấy lại thông số của URL, Lưu file này thành getjs và thực thi trên cửa sổ lệnh:

*node get.js*

Sau đó vào đường dẫn: http://localhost:8080/?data-put some\_text\_here để xem kết quả.

2.4.6 Ứng dụng tính kết quả biểu thức cho trước.

Sử dụng chương trình notepad hoặc công cụ soạn thảo lập trình nào đó để tạo 2 file sau. Server.js:

Tạo một tập tin server.js với nội dung sau:

*var io = require('socket.io');*

*var socket - io.listen (BOBO); socket.sockets.on('connection', function (socket) {socket.on('message', function (expr) console.log('Received expression from client ', expr);*

*4/ Bắt lỗi đối với biểu thức xấu*

*tryi*

*socket.send(eval (expr));*

*} catch (err)*

*socket.emit ("error", err.message);*

*socket.on('disconnect', function() {*

*// console.log('Disconnected');*

*});*

*Client.html:*

*<html>*

*<head> <title>WebSocket Client</title> <script src="http://localhost:8080/socket.io/socket.io.js"></script> <script> window.onload = function() {*

*var input - document.getElementById('input'); var output = document.getElementById('output');*

2.4.7 Ứng dụng webchat:

* 1. **Xây dựng một webchat server cơ bản.**

Trước tiên, ta sẽ nhập các mô-đun Socket.IO.

*var exp = require('express');*

*var app = exp.createServer();*

Sau đó, ta sẽ tạo ra một biến toàn cục duy nhất của Socket.IO được sử dụng để chia sẻ trên nhiều ứng dụng.

*global.socket = require('socket.io').listen (app); global.socket.set('log level', 1);*

*global.socket.set('transports', ['websocket', 'flashsocket', 'htmlfile', 'xhr-polling', 'jsonp-polling']);*

Ta sẽ tải tập tin cấu hình, router, và một mô-đun chat-socket đã có sẵn trong cùng một thời điểm.

*require('./app/config') (app, exp);*

*require('./app/server/router') (app); require('./app/server/modules/chat-socket');*

Và sau đó khởi động

*server. app.listen (8080, function() {*

*console.log("Express server listening on port $d in %s mode", app.address() .port, app.settings.env);*

* 1. **Tạo module Server-Side chat socket.**

Điều đầu tiên ta cần làm là xác định không gian tên cho ứng dụng này.Ta làm điều này với câu lệnh socket.of ('namespace').

*module.exports - function()*

*global.socket.of('/chat').on('connection', function (socket) [*

*console.log("a user has connected to the 'chat' namespace");*

*}); }();*

Bây giờ ta có thể thêm một số code để theo dõi người sử dụng kết nối, phát tin nhắn của họ và lắng nghe khi họ ngắt kết nối.

Nhưng trước tiên hãy tạo một mảng các màu mà ta ngẫu nhiên sẽ gán cho người dùng khi họ kết nối để phân biệt nhau trong bảng chat.

*var colors = ['#AE331F', '#D68434', '#116A9F', '#360B95', '#5F209E'];*

*Tiếp theo ta sẽ tạo ra một " connections object" để theo dõi người dùng kết nối.*

*var connections - {};*

Kịch bản phía client sẽ phát ra một sự kiện "user-ready" sau khi một người kết nối thành công và được giao một tên. Server sẽ lưu trữ giá trị đó trong socket server để sử dụng trong tương lai. Server cũng sẽ giao cho người sử dụng mới được kết nối một màu ngẫu nhiên từ mảng màu sắc có sẵn.

*global.socket.of("/chat').on('connection', function (socket) {*

*socket.on('user-ready', function (data) socket.name = data.name;*

*socket.color - data.color - colors [Math.floor(Math.random() \* colors.length)); // let everyone else in the room know a new user has just connected //*

*brodcastMessage('user-ready', data);*

Ta sẽ lắng nghe cho đến khi họ ngắt kết nối, do đó ta có thể loại bỏ chúng từ các đối tượng kết nối và thông báo cho người sử dụng.

*socket.on('disconnect', function() {*

* 1. **Khởi tạo kết nối trên Client:**

Khi một người dùng kết nối với ứng dụng webchat, ta sẽ cung cấp cho họ một tên mặc định. Khi gửi tin nhắn, ta sẽ lấy giá trị này và thêm nó vào tin nhắn được gửi đi.

*$('#name').val(Math.random().toFixed(8).toString().substr(2)); socket = io.connect('/chat');*

*Đó là những không gian tên riêng biệt cho phép các ứng dụng chatcó thể duy trì quyền tự chủ trong khi chạy như tên miền phụ của ứng dụng Node.*

*$('#btn-send').click(function () { sendMessage(); })*

*var sendMessage = function() {*

2.5 Mối liên hệ giữa JavaScript và NodeJS

Nền tảng NodeJS là Javascript, khi bắt đầu với lập trình NodeJS chúng ta cần nắm rõ kiến thức về JavaScript để việc bắt đầu với NodeJS dễ dàng hơn. Khi lập trình với NodeJS bạn sử dụng hoàn toàn cú pháp của Javascript, code JavaScript hoạt động được trên Node, nhưng điều ngược lại thì chưa chắc, NodeJS vẫn có tích hợp một số module riêng nên chỉ sử dụng được trong NodeJS chứ trong Javascript không sử dụng được.

CHƯƠNG 3 – CƠ SỞ LÝ THUYẾT VÀ THỰC NGHIỆM

3.1

3.2

CHƯƠNG 4 – TỔNG KẾT

4.1 Kết quả đạt được

4.2 Những mặt hạn chế

4.3 Hướng phát triển trong tương lai