Chương 1: Khái niệm và cài đặt NodeJS

**1.NodeJS là gì:**

Node.js là nền tảng phía máy chủ được xây dựng trên Bộ công cụ JavaScript của Google Chrome (Công cụ V8). Node.js được phát triển bởi Ryan Dahl trong năm 2009 và phiên bản mới nhất của nó là 9.11.0.

Node.js là một nền tảng được xây dựng trên thời gian chạy JavaScript của Chrome để dễ dàng xây dựng các ứng dụng mạng nhanh và có thể mở rộng. Node.js sử dụng mô hình I / O theo định hướng theo sự kiện(event-driven), không khóa(non-blocking I/O model) và làm cho nó nhẹ và hiệu quả, hoàn hảo cho các ứng dụng thời gian thực đòi hỏi nhiều dữ liệu chạy trên các thiết bị phân tán.

Node.js là một môi trường chạy đa nền tảng nguồn mở, để phát triển các ứng dụng máy chủ và mạng. Các ứng dụng Node.js được viết bằng JavaScript và có thể chạy Node.js trên OS X, Microsoft Windows và Linux.

Node.js cũng cung cấp một thư viện phong phú các mô-đun JavaScript khác nhau giúp đơn giản hóa sự phát triển của các ứng dụng web sử dụng Node.js tới một mức độ lớn.

Node.js = Runtime Environment + JavaScript Library

**2.Các tính chất của NodeJS:**

-Bất đồng bộ và sự kiện(Asynchronous and Event Driven )

-Rất nhanh(very fast)

-Chạy đơn threaded nhưng mở rộng cao(Single Threaded but Highly Scalable )

-Không dùng bộ nhớ đệm (no buffering).

-Giấy phép(License)  sử dụng giấy phép the [MIT license](https://raw.githubusercontent.com/joyent/node/v0.12.0/LICENSE).

**3. Những ứng dụng sử dụng nodeJS**

* I/O bound Applications
* Data Streaming Applications
* Data Intensive Real-time Applications (DIRT)
* JSON APIs based Applications
* Single Page Applications

**4. Các thành phần của NodeJS**



**5.Công cụ soạn thảo NodeJS:**

-Nodepad, nodepad++ trên windows

-vim hoặc vi sử dụng trên Linux or UNIX.

Hoặc một số tools khác: Visual Studio Code, [Cloud 9](https://c9.io/), [Intellij IDEA](https://www.jetbrains.com/idea/), [WebStorm](https://www.jetbrains.com/webstorm/), [Komodo IDE](https://www.activestate.com/komodo-ide), [Koding](https://www.koding.com/), [Eclipse](https://eclipse.org/ide/), [WebMatrix](https://www.microsoft.com/Web/webmatrix/node.aspx).

**6.Tên file**

Tất cả các file của NodeJS có đuôi mở rộng là .js

**7. Cài đặt NodeJS**

Tùy hệ điều hành mà chọn cài đặt cho phù hợp

[www.nodejs.org](http://www.nodejs.org) để cập nhật version mới nhất

|  |  |
| --- | --- |
| **OS** | **Archive name** |
| Windows | node-v6.3.1-x64.msi |
| Linux | node-v6.3.1-linux-x86.tar.gz |
| Mac | node-v6.3.1-darwin-x86.tar.gz |
| SunOS | node-v6.3.1-sunos-x86.tar.gz |

Cài đặt trên UNIX/Linux/Mac OS X, và SunOS

Dựa trên kiến trúc hệ điều hành, tải về và giải nén vào thư mục tmp sau đó di chuyển các file đó vào trong thư mục /usr/local/nodejs

Ví dụ: trên linux

$ cd /tmp

$ wget http://nodejs.org/dist/v6.3.1/node-v6.3.1-linux-x64.tar.gz

$ tar xvfz node-v6.3.1-linux-x64.tar.gz

$ mkdir -p /usr/local/nodejs

$ mv node-v6.3.1-linux-x64/\* /usr/local/nodejs

Cấu hình /usr/local/nodejs/bin trong biến môi trường Path

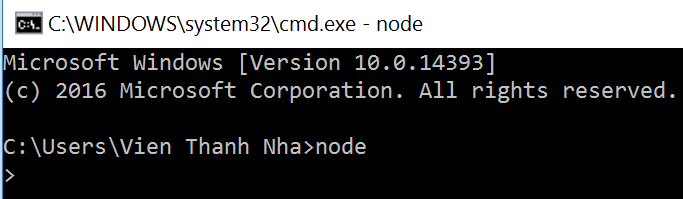
|  |  |
| --- | --- |
| **OS** | **Output** |
| Linux | export PATH=$PATH:/usr/local/nodejs/bin |
| Mac | export PATH=$PATH:/usr/local/nodejs/bin |
| FreeBSD | export PATH=$PATH:/usr/local/nodejs/bin |

### Cài đặt trên Windows:

Dùng file MSI và cài đặt theo hướng dẫn của Node.js. Thư mục cài đặt mặc định của Node.js C:\Program Files\nodejs.

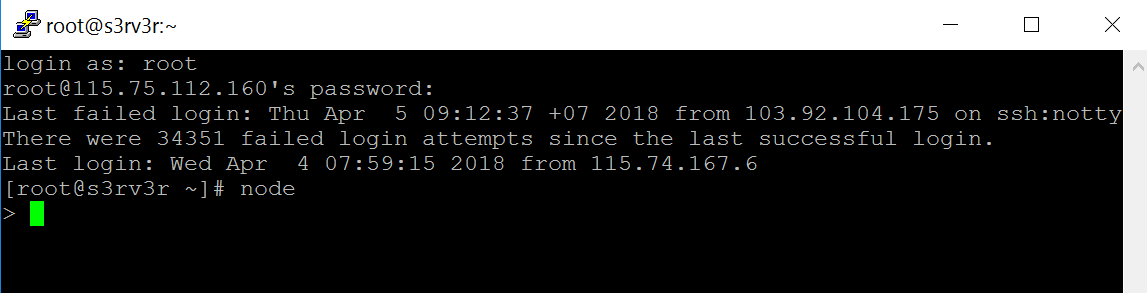
Sau đó tiến hành Test nodejs đã cài đặt hoàn thành chưa

Trên windows-> run->cmd->node



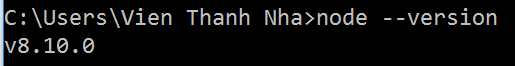
Trên linux:đăng nhập từ tài khoản root

Gõ node

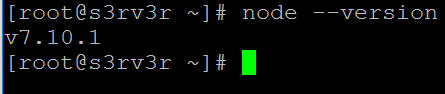


Kiểm tra version của node đã cài đặt:

Node –version trên windows



Node –version trên ubuntu hoặc centos..



### **8. Một ví dụ về NodeJS và cách chạy file Nodejs**

*Main.js*

var http = require("http");

http.createServer(function (request, response) {

// Send the HTTP header

// HTTP Status: 200 : OK

// Content Type: text/plain

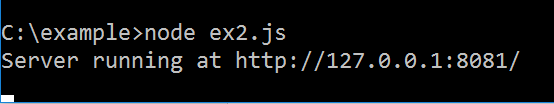
response.writeHead(200, {'Content-Type': 'text/plain'});

// Send the response body as "Hello World"

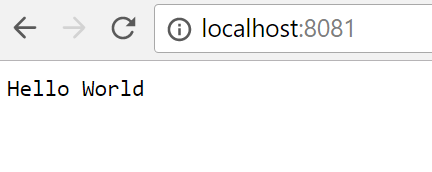
response.end('Hello World\n');

}).listen(8081);

console.log('Server running at http://127.0.0.1:8081/');

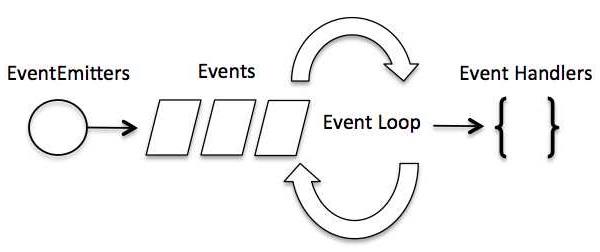
*Kết quả:*

*Kết quả chạy bằng trình duyệt(Client)*



9. Even-Driven programming: Lập trình dựa trên sự kiện

Node.js sử dụng sự kiện rất nhiều và nó cũng là một trong những lý do tại sao Node.js là khá nhanh so với các công nghệ tương tự khác. Ngay khi Node bắt đầu máy chủ của nó, nó chỉ đơn giản là khởi tạo các biến của nó, tuyên bố các hàm và sau đó chỉ cần đợi cho sự kiện xảy ra. Trong một ứng dụng điều khiển sự kiện, thường có một vòng lặp chính nghe các sự kiện, và sau đó kích hoạt một chức năng gọi lại khi một trong những sự kiện được phát hiện.



Sơ đồ các bước để tạo ra sự kiện

Các bước để tạo sự kiện

Import sự kiện module

var events = require('events');

Tạo đối tượng eventEmitter

var eventEmitter = new events.EventEmitter();

Sau đó rán cho sự kiện

// Bind event and event handler as follows

eventEmitter.on('eventName', eventHandler);

Gọi sự kiện ra

eventEmitter.emit('eventName');

Ví dụ:

var events = require("events");

var eventEmitter = new events.EventEmitter();

var onoff=false;

var ringBell = function ringBellss(){

if(onoff==true){

onoff=false;

console.log("Light"+onoff);

}else{

onoff=true;

console.log("Light"+onoff);

}

eventEmitter.emit("received");

}

eventEmitter.on("received", function(){

console.log("Light"+onoff);

}

);

eventEmitter.on('doorOpen', ringBell);

eventEmitter.emit('doorOpen');

Kết quả:

Lighttrue

Lighttrue

Ví dụ:

Giả sử có 2 hàm giải phương trình bậc 1 và 2 như sau:

var ptb1=function (a,b)

{

var x;

if(a==0){

if(b==0){

x="Phương trình vô số nghiệm";

}else{

x="Phương trình vô nghiệm";

}

}

else{

x=-b/a;

}

console.log(x);

}

var ptb2=function (a,b,c){

var x,x1,x2;

if(a==0){

x=ptb1(b,c);

}else{

var d=b\*b-4\*a\*c;

if(d==0){

x="x="+(-b/(2\*a));

}else if(d<0){

x="Phương trình bậc 2 vô nghiệm";

}else{

x1=-b+Math.sqrt(d)/(2\*a);

x2=-b-Math.sqrt(d)/(2\*a);

x="x1="+x1+"x2="+x2;

}

}

console.log(x);

}

var events = require("events");

var eventEmitter = new events.EventEmitter();

//Đăng ký sự kiện

eventEmitter.on('giaipt', ptb1);

eventEmitter.on('giaipt', ptb2);

//Gọi sự kiện

eventEmitter.emit('giaipt',2,9,6);

Kết quả:

-4.5

x1=-7.563859338365493x2=-10.436140661634507

## 10. Blocking Code

Tạo file input.txt:

Cám ơn bạn đã đọc giáo trình này

Create a js file named main.js with the following code −

var fs = require("fs");

var data = fs.readFileSync('input.txt');

console.log(data.toString());

console.log("Kết thúc chương trình");

chạy main.js

$ node main.js

Kết quả.

Cám ơn bạn đã đọc giáo trình này

Kết thúc chương trình

Ví dụ này cho thấy rằng chương trình sẽ chặn cho đến khi nó đọc tập tin và sau đó nó chỉ tiến hành để kết thúc chương trình.

11.Non-block

var fs = require("fs");

fs.readFile('input.txt', function (err, data) {

if (err) return console.error(err);

console.log(data.toString());

});

console.log("Program Ended");

Chạy main.js

$ node main.js

Kết quả.

Kết thúc chương trình

Cám ơn bạn đã đọc giáo trình này

Ví dụ thứ hai cho thấy chương trình không chờ đọc tệp và tiến hành in "Chương trình đã kết thúc" và đồng thời, chương trình không chặn tiếp tục đọc tệp.

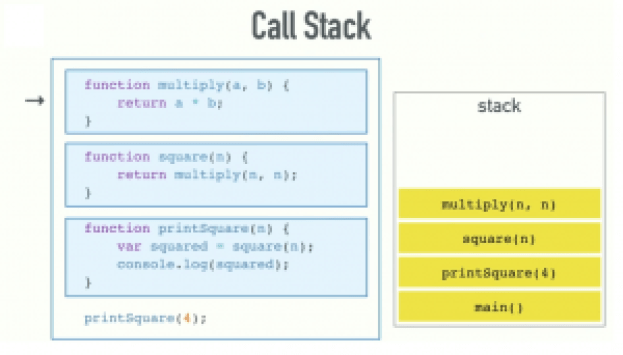
Do đó, một chương trình chặn thực hiện rất nhiều theo trình tự. Từ quan điểm lập trình, nó dễ dàng hơn để thực hiện logic nhưng các chương trình không chặn không thực hiện theo thứ tự.

Trong trường hợp một chương trình cần sử dụng bất kỳ dữ liệu nào để xử lý, nó sẽ được giữ trong cùng một khối để thực thi tuần tự.

12. Callback:

Các hàm được truyền vào một hàm khác được gọi là hàm callback, thường chúng ta sẽ gọi các hàm này cuối cùng sau khi đã thực hiện các công việc khác. Đầu tiên chúng ta sẽ tìm hiểu cách Node chạy các hàm callback.

Trong vùng bộ nhớ của Javascript có một khu vực gọi là *Callstack*, các hàm được gọi sẽ được sắp xếp lần lượt trong stack này, và được gọi lần lượt theo thứ tự *“vào trước ra sau” (FILO – First In Last Out).*Bạn xem trong hình dưới đây:

[](https://i2.wp.com/phocode.com/wp-content/uploads/2016/10/Untitled.png)

Trong hình trên, chúng ta có đoạn code, các hàm được gọi lần lượt theo thứ tự từ main()→ printSquare() → square() → multiply(), sau đó hàm multiply() trả về trước, rồi đến hàm square() trả về, rồi hàm printSquare() gọi console.log() và kết thúc, đây là các lời gọi bình thường, không phải callback.

Node cho phép chúng ta truyền tham số vào một hàm là một hàm khác chứ không chỉ có các giá trị thô như số nguyên, chuỗi, ví dụ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | var click = function(str, callback) {      console.log("Click() called: " + str);      callback();  }    click("Something", function() {      console.log("Callback() called");  });    var callback2 = function() {      console.log("callback2() called");  }    click("Something", callback2); |

Trong đoạn code trên, chúng ta có hàm click() nhận vào một tham số là str và một tham số là một hàm khác, khi chúng ta gọi hàm click(), chúng ta có thể tryền vào một hàm đã được định nghĩa trước hoặc định nghĩa ngay trong lời gọi hàm luôn cũng được.

|  |  |
| --- | --- |
| Output | |
| 1  2  3  4 | Click() called: Something  Callback() called  Click() called: Something  callback2() called |

Ví dụ:

var myCallback = function(data) {

console.log('got data: '+data);

};

var usingItNow = function(callback) {

callback('get it?');

};

//Gọi hàm sử dụng callback

usingItNow(myCallback);

Kết quả:

Got data: get it?

Ví dụ:Sử dụng callback có kiểm tra lỗi

var myCallback = function(err, data) {

if (err) throw err; // Check for the error and throw if it exists.

console.log('got data: '+data); // Otherwise proceed as usual.

};

var usingItNow = function(callback) {

callback(null, 'get it?'); // I dont want to throw an error, so I pass null for the error argument

};

Hoặc muốn điều khiển lỗi thì

var usingItNow = function(callback) {

var myError = new Error('My custom error!');

callback(myError, 'get it?'); // I send my error as the first argument.

};

//Gọi hàm

usingItNow(myCallback);

### **Chương II. Lập trình NodeJS**

**1.Kiểu dữ liệu**

Javascript có nhiều kiểu dữ liệu như: numbers, strings, objects

Ví dụ:

### var length = 16;                               // Number var lastName = "Thanh Nhã";                      // String var x = {firstName:"Thanh", lastName:"Nhã"};    // Object

### Kiểu dữ liệu là rất quan trọng trong lập trình nhằm xác định để thực hiện các phép toán trên biến

Ví dụ:

var x = 16 + "Nhã";

var x = "16" + "Nhã";

Trong Javacript khi số number + chuỗi string thì số sẽ được chuyển sang chuỗi

Kết quả: x=”16Nhã”

Trong Javacript giá trị biểu thức sẽ thực hiện từ trái sang phải:

var x = 16 + 4 + "Nhã";

Kết quả:20Nhã

Nhưng var x = "Nhã" + 16 + 4;

Kết quả:

Nhã164

### Trong Javascript cho phép khai báo biến kiểu động(dynamic)

var x;           // x is undefined  
x = 5;           // x is a Number  
x = "John";      // x is a String

**Kiểu chuỗi:**Chuỗi là hàng loạt các kí tự

Trong Javascript cho phép khai báo kiểu chuỗi trong cặp một nháy hoặc 2 nháy ‘Nhã’ hoặc “Nhã”

Bạn có thể sử dụng dấu ngoặc kép bên trong một chuỗi, miễn là chúng không khớp với dấu ngoặc kép xung quanh chuỗi

var answer = "It's alright";             // Single quote inside double quotes  
var answer = "He is called 'Johnny'";    // Single quotes inside double quotes  
var answer = 'He is called "Johnny"';

Kết quả:

It's alright  
He is called 'Johnny'  
He is called "Johnny"

## Javacript chỉ có một loại kiểu số:

## var x1 = 34.00;

## var x2 = 34;

## var x3 = 3.14;

Kết quả:

34  
34  
3.14

Số cực nhỏ hoặc cực lớn có thể được viết bằng ký hiệu khoa học (hàm mũ):

var y = 123e5;      // 12300000  
var z = 123e-5;     // 0.00123

Kết quả:

12300000  
0.00123

Kiểu boolean: có 2 giá trị đúng sai

var a=true;

var b=false;

A==b ->false

**Kiểu mảng:**

Được sử dụng bởi dấu [ ]

Ví dụ:

var color=[“Xanh”,”đỏ”,”vàng”];

Truy xuất mảng color[0]->Xanh

**Mảng đối tượng:**

Các đối tượng JavaScript được viết bằng dấu ngoặc nhọn.

Thuộc tính đối tượng được viết dưới dạng cặp tên: giá trị, được phân tách bằng dấu phẩy.

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

Truy xuất:

person.firstName + " is " + person.age + " years old.";

Kết quả:

John is 50 years old.

**Phép toán typeof:**

Bạn có thể sử dụng toán tử typeof của JavaScript để tìm kiểu biến JavaScript.

Toán tử typeof trả về kiểu của một biến hoặc một biểu thức:

typeof ""                  // Returns "string"  
typeof "John"              // Returns "string"  
typeof "John Doe"          // Returns "string"

typeof 0                   // Returns "number"  
typeof 314                 // Returns "number"  
typeof 3.14                // Returns "number"  
typeof (3)                 // Returns "number"  
typeof (3 + 4)             // Returns "number"

Undefined:

Trong JavaScript, một biến không có giá trị, có giá trị undefined. Kiểu type cũng không xác định.

var car;                // Value is undefined, type is undefined

typeof car; //undefined

Hoặc chúng ta có thể định nghĩa cho nó:

var car=undefined;

**Giá trị rỗng:**

Một giá trị rỗng không liên quan gì đến undefined.

Một chuỗi rỗng có cả giá trị rỗng và một kiểu.

var car = "";              // The value is "", the typeof is "string"

**Null:**

Trong JavaScript null là "không có gì". Nó được cho là một cái gì đó không tồn tại.

Thật không may, trong JavaScript, kiểu dữ liệu của null là một đối tượng.

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
person = null;        // Now value is null, but type is still an object

typeof person; //object

Ta cũng có thể sử dụng undefined :

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
person = undefined;   // Now both value and type is undefined

**Kiểu sơ khai (cơ bản):**

Giá trị dữ liệu sơ khai là một giá trị dữ liệu duy nhất đơn giản mà không có thuộc tính và phương pháp bổ sung.

* string
* number
* boolean
* undefined

typeof "John"              // Returns "string"   
typeof 3.14                // Returns "number"  
typeof true                // Returns "boolean"  
typeof false               // Returns "boolean"  
typeof x                   // Returns "undefined" (if x has no value)

**Kiểu phức tạp:**

**Toán tử typeof có thể trả về một trong hai loại phức tạp:**

**Toán tử typeof trả về đối tượng cho cả đối tượng, mảng và null.**

**Toán tử typeof không trả về đối tượng cho các hàm.**

typeof {name:'John', age:34} // Returns "object"  
typeof [1,2,3,4]             // Returns "object" (not "array", see note below)  
typeof null                  // Returns "object"  
typeof function myFunc(){}   // Returns "function"

**2. Hàm (Funtion)**

Một hàm JavaScript là một khối mã được thiết kế để thực hiện một tác vụ cụ thể.

Một hàm JavaScript được thực hiện khi "cái gì đó" gọi nó (gọi nó).

Ví dụ:

function myFunction(p1, p2) {  
    return p1 \* p2;       // Tích của 2 số p1, p2  
}

Gọi hàm:

Console.log(myFunction(3,4));//12

Cú pháp:

function name(parameter1, parameter2, parameter3) {  
    code to be executed  
}

Các dấu ngoặc đơn có thể bao gồm các tên tham số được phân cách bằng dấu phẩy:

(parameter1, parameter2, ...)

Các paramerter là các tham số hình thức có nghĩa chỉ đại diện trong hàm đó còn khi gọi hàm thì tham số đó là tham số thực

Ví dụ:

var a=10;

var b=12;

var kq= myFunction(a,b);//tham số thực a,b không cần phải là p1, hoặc p2

console.log(kq);//120

Hàm return :

-Được dùng khi muốn nhận kết quả của hàm được gọi hoặc sử dụng cho mục đích input cho một hàm khác.

-Dùng để tái sử dụng cho những lần gọi sau

Một hình ảnh sau cho ta biết nên dùng hàm return hay ?

Blackbox

Số input nhiều hơn output thì nên sử dụng hàm

Ví dụ:

Tìm ước số chung của 2 số nguyên a và b

fuction uscln(a,b)

{

while(a!=b){

if(a>b){

a=a-b;

}else{

b=b-a;

}

}

return a;

}

Gọi hàm:

var us= uscln(4,8);

console.log(us);->4

USCLN

b

a

a

Ta dùng kết quả của hàm này để tính bộ số chung nhỏ nhất:

function bscnn(a,b){

var aa=a;

var bb=b;

var us=uscln(a,b);

return (aa\*bb)/us;

}

Gọi hàm:

Console.log(bscnn(4,8));//8

**Không dùng return:**

**Khi muốn kết quả hiển thị sau khi gọi hàm**

fuction uscln(a,b)

{

while(a!=b){

if(a>b){

a=a-b;

}else{

b=b-a;

}

}

console.log(“Ước số chung lớn nhất”+a);

}

Gọi hàm:

Uscln(4,8);//4

Một cách viết khác gọi là hàm vô danh

**var** x = function (a, b) {

return a + b;

}

console.log(x(10, 20));

Kết quả: 30

**var** x = function tong(a, b) {

return a + b;

}

console.log(x(10, 20));

Kết quả: 30

**Một dạng hàm khác:**

(function myData() {

console.log(**'myData was executed!'**);

})();

Chúng ta có thể thực hiện một hàm ngay sau khi bạn xác định nó.

Đơn giản chỉ cần bỏ các chức năng trong dấu ngoặc đơn () và gọi nó.

3.Phạm vi biến:

var a="cuong";//Toàn cục

(function myData() {

var a="nhã";//cục bộ hàm

console.log('myData was executed!'+a); //a này chịu ảnh hưởng của hàm biến cục bộ hàm

})();

Kết quả: myData was executed!nhã

var a="cuong";//Toàn cục

(function myData() {

console.log('myData was executed!'+a);

//a này chịu ảnh hưởng của hàm biến cục bộ hàm

})();

Kết quả:

myData was executed!cường

**3. Một số hàm có sẵn thông dụng:**

\*Chuỗi:

**Hàm indexOf(x)**

Ví dụ:

**var** i = **"this is a test"**.indexOf(**"is"**);

console.log(i);->2

Hàm này trả về vị trí đầu tiên khi tìm thấy x trong chuỗi.

Nếu không tìm thấy trả về -1

**Hàm substr(vị trí cần lấy,số kí tự cần lấy):** hàm lấy chuỗi con trong chuỗi

var s = "this is a test string.".substr(19, 3);

console.log(s);

Kết quả: ng.

**Hàm slice(vị trí bắt đầu, vị trí kết thúc chuỗi): dùng để cắt chuỗi**

var s1 = "this is a test string.".slice(19, 22);

console.log(s1);

Kết quả: ng.

**Hàm Split(express): Để tách chuỗi thành các chuỗi con, Kết quả nhận được là 1 mảng**:

Ví dụ:

**var** s = **"a|b|c|d|e|f|g|h"**.split(**"|"**);

console.log(s);

Kết quả:  
http://www.java2s.com/Tutorials/JavascriptImage/myResult/S/SPLIT__07796568995FFDBD2133.PNG

**Trim():Loại bỏ khoảng trắng**

Ví dụ:

var s = ' cat '. trim();

console.log(s);

**Length: Dùng để tính chiều dài chuỗi**

Ví dụ:

Var s=”abs”;

s.length=> 3

**+: nối chuỗi:**

**Ví dụ:**

Var s=”thanh nhã”+” viên”-> thanh nhã viên

**4. Cấu trúc điều kiện**

**Phép toán ?:**

**Ví dụ:**

**var** isCat = true;

**var** pet = isCat ? **"cat"** : **"dog"**;

console.log(pet);

**Kết quả:**

**Cat**

if (diều kiện )

{

Lệnh 1;

Lệnh 2;

…

}

Else{

Lệnh 1;

Lệnh 2;

….

}

Ví dụ:

var a=5;

var b=10;

If(a>b)

console.log(a+” > ”+b);

Else{

console.log(a+” < ”+b);

}

Hoặc

If(điều kiện){

Lệnh 1;

Lệnh 2;

…

}else if(diều kiện){

Lệnh 1;

Lệnh 2;

…

}

Ví dụ:

var a=10;

var b=3;

var c=9;

var d=b\*b-4\*a\*c

If(d<0){

console.log(“Phương trình bậc 2 vô nghiệm”);

}else if(d==0){

console.log(“Phương trình bậc 2 có nghiệm kép x1=x2”+(-b/(2\*a)));

}else{

console.log(“x1=”+(-b+Math.sqrt(d)/(2\*a)))+”x2=”+ (-b+Math.sqrt(d)/(2\*a))));

}

Hoặc

If(Điều kiện){

If(điều kiện){

Lệnh 1;

Lệnh 2;

…

}

else{

Lệnh 1;

Lệnh 2;

…

}

}else{

}

Ví dụ:

Giải phương trình bậc 1

ax+b=0;

if(a==0)

if(b==0)

{

console.log(“Phương trình vô số nghiệm”);

}else{

console.log(“Phương trình vô nghiệm”);

}

else{

console.log(“Phương trình có nghiệm x=”+(-b/a));

}

Vòng lặp: là quá trình lặp đi lặp lại một số lệnh có tính chất tương tự theo một số lần

Khởi tạo biến chạy

Kiểm tra điều kiện lặp

Tập hợp lệnh

Thay đổi biến điều kiện

True

False

Kết quả

**Vòng lặp For:** Kiểm tra rồi thực hiện lệnh

Ví dụ thay gì ghi như sau:

var cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

var text = "";

text += cars[0] + "\n";   
text += cars[1] + "\n";   
text += cars[2] + "\n";   
text += cars[3] + "\n";   
text += cars[4] + "\n";   
text += cars[5] + "\n";

Ta có thể dụng dòng lặp như sau:

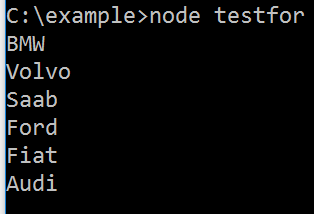
var cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

var text = "";

var i;  
for (i = 0; i < cars.length; i++) {   
    text += cars[i] + \n";  
}

console.log(text);

Kết quả



**Vòng lặp while:** Kiểm tra rồi thực hiện lệnh

var cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

var text = "";

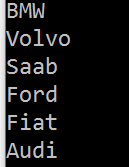
var i=0;

while(i<cars.length){

text += cars[i] + \n";  
i++;

}  
console.log(text);

Kết quả như nhau

****

**Vòng lặp do … while:** thực hiện ít nhất 1 lệnh rồi kiểm tra

var cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

var text = "";

var i=0;

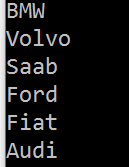
do{

text += cars[i] + \n";  
i++;

}while(i<cars.length);

console.log(text);

Kết quả:



Module là các thành phần cơ bản để xây dựng nên một ứng dụng Node. Một module chứa các hàm API, tức là chúng ta chỉ biết chính xác hàm đó có tên gì, có tham số gì, dữ liệu trả về là gì chứ không biết chi tiết chúng hoạt động như thế nào.

Trong các bài trước chúng ta đã làm việc với module, bản thân mỗi file JavaScript đã là một module rồi. Trong phần này chúng ta sẽ tìm hiểu sâu hơn.

## Định nghĩa một module

Trong [bài trước](https://phocode.com/javascript/nodejs/nodejs-su-dung-node/) chúng ta có viết một file có tên ls.js, trong file đó có dòng code như sau:

|  |  |
| --- | --- |
| 1 | var fs = require('fs'); |

Trong dòng code trên chúng ta dùng hàm require(), hàm này có chức năng tìm module fs và chép vào trình thông dịch, và chúng ta có thể sử dụng các hàm có trong module đó. Đối tượng fs (biến var fs) chứa code được định nghĩa trong module fs.

Tiếp theo chúng ta xem xét ví dụ sau:

|  |  |
| --- | --- |
| simple.js | |
| 1  2  3  4 | var count = 0;  exports.next = function() {      return count++;  } |

Chúng ta tạo một file có tên simple.js với đoạn code trên, sau đó mở terminal lên và gõ lệnh node, chỉ gõ node để mở trình thông dịch Node lên thôi chứ không chạy file simple.js. Lưu ý nếu thư mục hiện tại trên terminal không phải là thư mục chứa file simple.js thì bạn phải chuyển đường dẫn đến thư mục đó (bằng lệnh cd trên cả Windows và Linux, nếu không biết bạn có thể tìm trên Google cách dùng). Sau đó chúng ta chạy một số lệnh như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | C:\Users\PhoCode>node  > var s = require('./simple');  > s.next();  0  > s.next();  1  > s.next();  2  > s.next();  3  > |

Trong file simple.js, đối tượng exports được trả về từ hàm require('./simple'). Cũng có nghĩa là cứ mỗi lần chúng ta gọi s.next() thì hàm next() được định nghĩa trong file simple.js sẽ được gọi, ở đây hàm next() trả về giá trị của biến count và tăng biến đó lên 1 đơn vị, do đó các lần gọi tiếp theo giá trị sẽ tăng dần.

Quy tắc ở đây là bất cứ thứ gì từ hàm cho tới đối tượng nào mà được gán vào làm một trường của đối tượng exports thì đều có thể gọi được ở ngoài, nếu không phải là một trường của đối tượng exports thì sẽ không thể gọi tới được, cho dù có require() hay không.

Lưu ý là các đối tượng global được định nghĩa trong một file .js cũng chỉ có thể nhìn thấy ở trong file đó thôi, chứ không thể gọi từ file khác được, nếu muốn gọi từ nơi khác thì chúng ta cũng phải gán vào đối tượng exports.

## Các module trong Node

Các module có trong Node thường được viết theo cách viết của module CommonJS (chúng ta sẽ tìm hiểu về CommonJS sau) nhưng không giống hoàn toàn. Trong Node, mỗi module được lưu trong một file, cách lưu các file này cũng rất linh hoạt.

Tên một module chính là tên file .js nhưng không có phần đuôi, tức là chúng ta viết require('./simple') thì Node sẽ tự hiểu là tìm file simple.js.

Ngoài các module là code thì Node cũng hỗ trợ các thư viện đã được dịch ra thành file nhị phân, ở đây các file này có đuôi là .node. Tuy nhiên chúng ta sẽ không đi sâu vào tìm hiểu các module kiểu này, bạn chỉ cần nhớ là các file có đuôi .node là các file module bình thường như các file .jsthôi, có điều là chúng đã được dịch thành mã nhị phân.

Ngoài ra còn có các module cốt lõi đã được biên dịch thành file khả thi (có đuôi .exe trong Windows).

Đường dẫn để tìm module của Node có 3 loại là đường dẫn tương đối, đường dẫn tuyệt đối hoặc top-level.

Ví dụ:

* Đường dẫn tương đối : ./simple.js, ../simple.js
* Đường dẫn tuyệt đối : C:/simple.js
* Đường dẫn top-level: simple.js (module nằm chung thư mục)

## Module nằm trong một app

Thường thì các module sẽ nằm trong một ứng dụng nào đó hoặc không.

Một ứng dụng thường là một thư mục chứa các file và thư mục con trong đó, các thư mục con này có thể chứa file code, file tài nguyên, hoặc chứa các module khác. Các module biết chúng được đặt ở đâu và khi cần thì có thể gọi đến nhau một cách dễ dàng.

Ví dụ dưới đây là thư mục của một gói rất phổ biến trong Node là gói Express, gói này chứa các file và thư mục con bên trong nó theo cấu trúc cây.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | express/      bin/      History.md      index.js      lib/          express.js          http.js          https.js          request.js          response.js          /router              collection.js              index.js              methods.js              route.js          utils.js          view/              partial.js              view.js          view.js      LICENSE      Makefile      node\_modules\          qs\      package.json      Readme.md |

Trong gói Express thì module thường được sử dụng nhiều nhất là utils.js. Tùy vào vị trí được đặt ở đâu mà các module khác có câu lệnh gọi khác nhau, ví dụ:

|  |  |
| --- | --- |
| 1  2  3 | var utils = requrire('./lib/utils');  var utils = requrire('./utils');  var utils = requrire('../utils'); |

## Cách tìm module của Node

Ngoài thư mục của ứng dụng thì các module thường còn được đặt trong thư mục node\_modules,khi gọi thì chúng ta chỉ cần dùng tên module là đủ (tức là đường dẫn kiểu top-level), ví dụ:

|  |  |
| --- | --- |
| 1 | var express = require('express'); |

Node sẽ tự động tìm module trong các thư mục node\_modules, thư mục này có rất nhiều, Node sẽ tìm thư mục node\_modules trong thư mục của file gọi đến module đó, nếu không thấy thì tiếp tục tiềm trong thư mục cha của thư mục chứa file gọi đến module đó và nếu không tìm thấy nữa thì Node sẽ tiếp tục đi dần dần lên các thư mục cha tiếp theo cho đến chạm đến thư mục gốc thì thôi.

Trong ví dụ trước, gói Express có một thư mục tên là node\_modules, trong thư mục này chứa một thư mục khác có tên là qs, tức là module qs và module này có thể được gọi tới bởi bất kì file nào trong thư mục express.

|  |  |
| --- | --- |
| 1 | var qs = require('qs'); |

Như vậy tóm lại cách dễ nhất để gọi các module là tạo thư mục node\_modules và đặt chúng ta trong thư mục này, thư mục node\_modules nên được đặt ở vị trí cao nhất để các module khác có thể thấy được. Ví dụ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | projects/      drawapp/          index.js          lib/              draw.js              node\_modules/              svg.js          node\_modules/              express/              bin/              History.md              index.js              lib/              LICENSE              Makefile              node\_modules/                  qs/              package.json              Readme.md          package.json      node\_modules/ |

Trong ví dụ trên, những module nằm trong thư mục drawapp có thể thấy được các module nằm trong thư mục projects/node\_modules, và ngược lại các module trong thư mục node\_modules này không thể thấy được module qs vì qs nằm ở trong cùng trong khi Node chỉ tìm các module bằng cách đi ngược về các thư mục cha.

Tương tự, những module nằm trong thư mục lib/node\_modules có thể thấy được từ draw.js và svg.js nhưng không thấy được từ index.js.

Lý do tại sao lại có thể có nhiều thư mục node\_modules là vì làm như thế thì Node cho phép chúng ta có thể dùng một module với nhiều phiên bản khác nhau thay vì chỉ dùng một thư mục duy nhất rồi chép tất cả vào gây lộn xộn.

## Tìm trong biến NODE\_PATH

Ngoài các thư mục node\_modules thì Node còn cho phép chúng ta đặt module trong các thư mục được định nghĩa trong biến môi trường NODE\_PATH, biến này lưu danh sách các đường dẫn thư mục mà Node có thể tìm trong đó, mỗi thư mục cách nhau bằng dấu chấm phẩy (trong Windows) hoặc dấu 2 chấm (trong Linux).