node JS™

Agenda

- ➢ About Node
- ➢ Globals
- ➢ Events
- ➢ Modules
- ➢ Node Architecture
- ➢ NPM commands
- ➢ Server creation

get ready to node, this session is hands on!

Agenda

- ➢ Web Application
- ➢ File System
- ➢ Routing
- ➢ Tcp/udp
- ➢ Process
- ➢ REST
- ➢ Express

get ready to node, this session is hands on!

If it's
Open
it
grows

About Node

- ➢ Open Source, cross-platform runtime environment for server-side and networking applications
- ➢ Initial Release May 27, **2009**
- ➢ Created by Ryan Dahl
- ➢ Stable Release 5**.3.x**
- ➢ http://nodejs.org/, https://github.com/joyent/node

About Node

- Applications are written in **JavaScript**
- Runs on Google’s V8 Engine (same as Chrome)
- Installation

  node-v5.3.0-x64.msi

Is Node JS a serious option for enterprise applications?

YES!

In fact, it's our **best** option for typical (heavy data IO) web applications!

Let's see why…

What is unique about Node.js?

1. JavaScript used in client-side but node.js puts the JavaScript on server-side thus making communication between client and server will happen in same language

2. Servers are normally thread based but Node.JS is “Event” based. Node.JS serves each request in a Evented loop that can able to handle simultaneous requests.

How does it differ?

"Node is similar in design to and influenced by systems like Ruby's event machine or Python's twisted. Node takes the event model a bit further—it presents the event loop as a language construct instead of as a library."

What can you do with Node ?

➢It is a command line tool. You download a tarball, compile and install the source.

➢It lets you Layered on top of the TCP library is a HTTP and HTTPS client/server.

➢The JS executed by the V8 javascript engine (the ting that makes Google Chrome so fast)

➢Node provides a JavaScript API to access the network and file system.

What can’t do with Node?

➢Node is a platform for writing JavaScript applications outside web browsers. This is not the JavaScript we are familiar with in web browsers. There is no DOM built into Node, nor any other browser capability.

➢Node can’t run on GUI, but run on terminal

Threads VS Event-driven

Threads	Asynchronous Event-driven
Lock application / request with listener-workers threads	only one thread, which repeatedly fetches an event
Using incoming-request model	Using queue and then processes it
multithreaded server might block the request which might involve multiple events	manually saves state and then goes on to process the next event
Using context switching	no contention and no context switches
Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock	Using asynchronous I/O facilities (callbacks, not poll/select or O_NONBLOCK) environments

Why node.js use event-based?

In a normal process cycle the webserver while processing the request will have to wait for the IO operations and thus blocking the next request to be processed.

Node.JS process each request as events, The server doesn’t wait for the IO operation to complete while it can handle other request at the same time.

When the IO operation of first request is completed it will call-back the server to complete the request.

Node.js VS Apache

1. It's fast
2. It can handle tons of concurrent requests
3. It's written in JavaScript (which means you can use the same code server side and client side)

Platform	Number of request per second
PHP (via Apache)	3187,27
Static (via Apache)	2966,51
Node.js	5569,30

Why Node?

Build Fast!

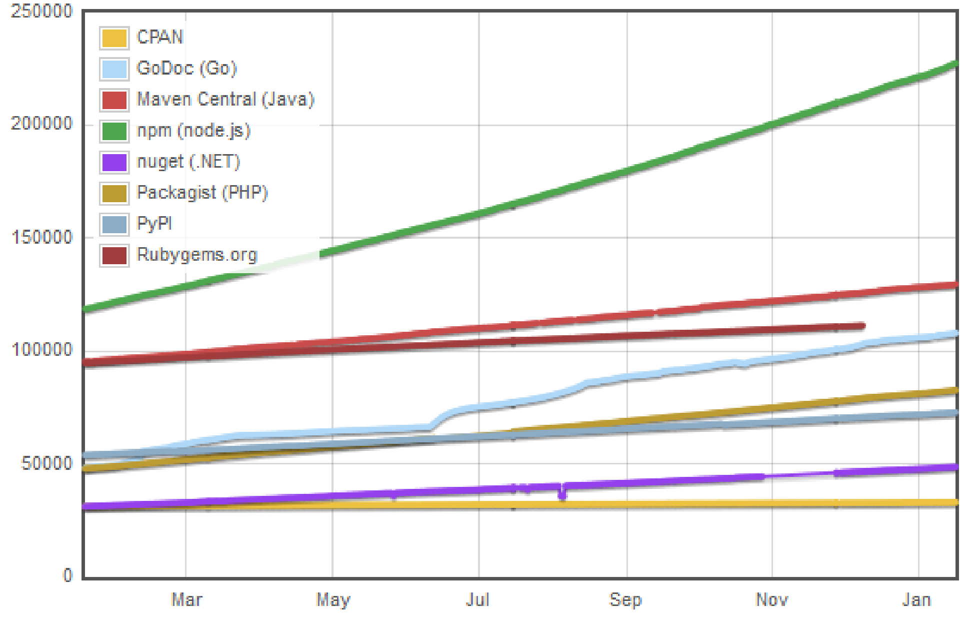
- ➢Less keystrokes with JS
- ➢JS on server and client allows for more code reuse
- ➢A lite stack (quick create-test cycle)
- ➢Large number of offerings for web app creation

Why Node?

- ➢Fast V8 Engine
- ➢Great I/O performance with event loop!
- ➢Small number of layers
- ➢Horizontal Scaling

Why Node?

Module Counts



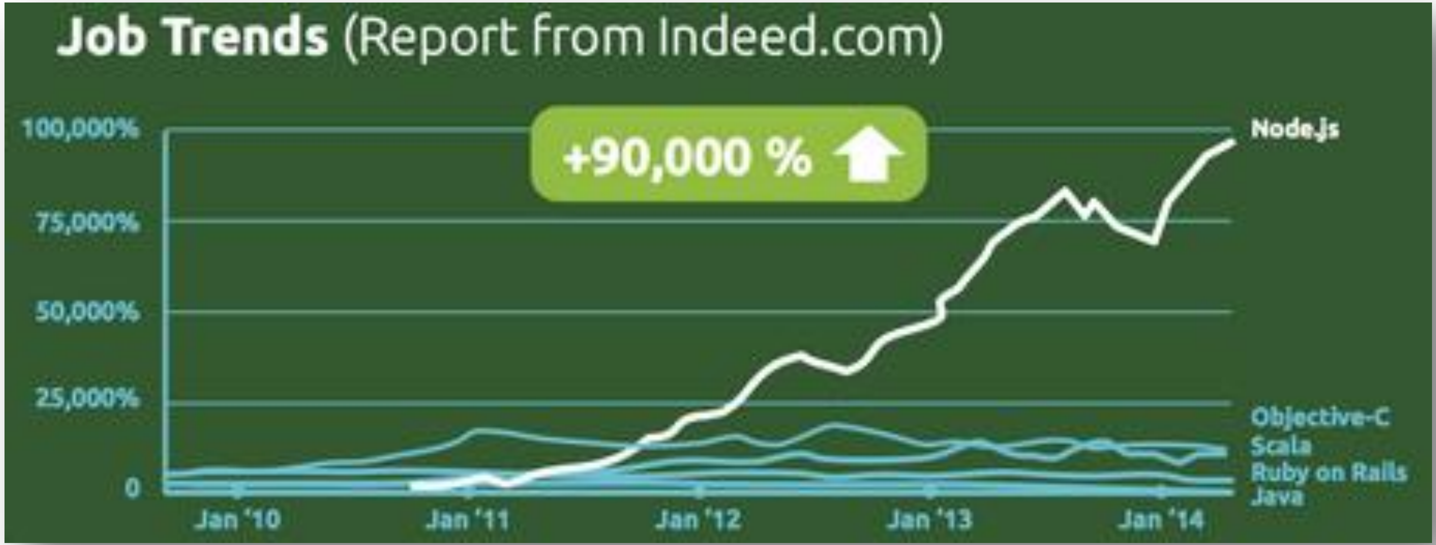
Include

- ☐ Bower (JS)
- ☐ Clojars (Clojure)
- ☑ CPAN
- ☐ CPAN (search)
- ☐ CRAN (R)
- ☐ Crates.io (Rust)
- ☐ Drupal (php)
- ☑ GoDoc (Go)
- ☐ Hackage (Haskell)
- ☐ Hex.pm (Elixir/Erlang)
- ☐ LuaRocks (Lua)
- ☑ Maven Central (Java)
- ☐ MELPA (Emacs)
- ☑ npm (node.js)
- ☑ nuget (.NET)
- ☑ Packagist (PHP)
- ☐ Pear (PHP)
- ☐ Perl 6 Ecosystem (perl 6)
- ☑ PyPI
- ☑ Rubygems.org

Community

- ➢ Over **2 million** downloads per month
- ➢ NPM averages **2000** new modules per day
- ➢ Total Number of npm Packages: **2,27,405**
- ➢ Second most stared project on Github
- ➢ Plus, corporate backing from Joyent

Community


Job Trends (Report from Indeed.com)
+90,000 %
100,000%
75,000%
50,000%
25,000%
0
Jan '10
Jan '11
Jan '12
Jan '13
Jan '14
Node.js
Objective-C
Scala
Ruby on Rails
Java

Hands On **#1** – **‘Welcome Noder!’**

Running Node Apps

Node Globals

- **process** – object providing information and methods for the current process
  - process.stdout, process.stderr, process.stdin, process.argv, process.env
- **console** – allows printing to stdout and stderr
- **require()** – function to load a module
- **module** – refers to the current module
- _filename , _dirname

Modules

- Modules allow Node to be extended (act as libaries)
- We can include a module with the global require function, **require(‘module’)**
- Node provides core modules that can be included by their name:
  - File System – **require(‘fs’)**
  - Http – **require(‘http’)**
  - Utilities – **require(‘util’)**

Modules

- ➢ We can also break our application up into modules and require them using a file path:
  - ➢ **‘/’** (absolute path), **‘./’** and **‘../’** (relative to calling file)
- ➢ Any valid type can be exported from a module by assigning it to **module.exports**

Modules

bar.js

```
module.exports = ‘I am a string’
```

foo.js

```
var msg = require(‘./bar.js’)
console.log(msg) // prints ‘I am a string’
```

Modules

bar.js

```
  module.exports.hello = function() {return ‘hello’}
  module.exports.bye = function() {return ‘bye’}
```

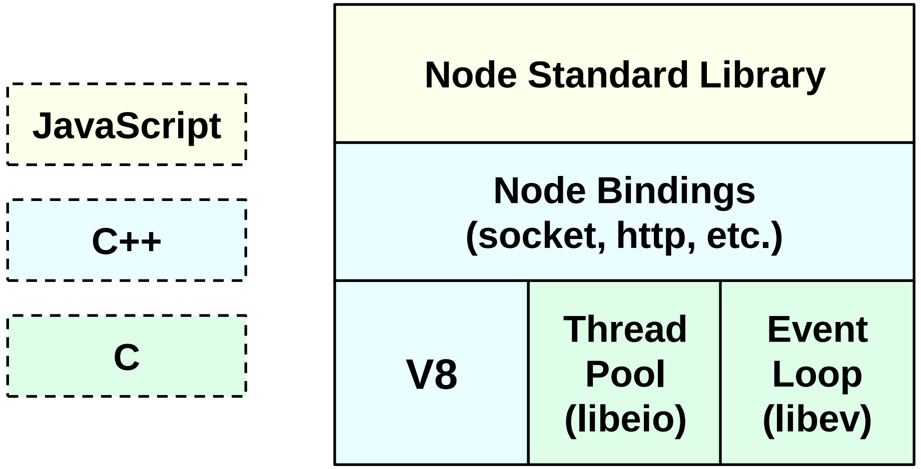
foo.js

```
  var bar = require(‘./bar.js’)
  console.log(bar.hello()) // prints ‘hello’
  console.log(bar.bye()) // prints ‘bye’
```

Hands On #**2** – **‘Require It’**

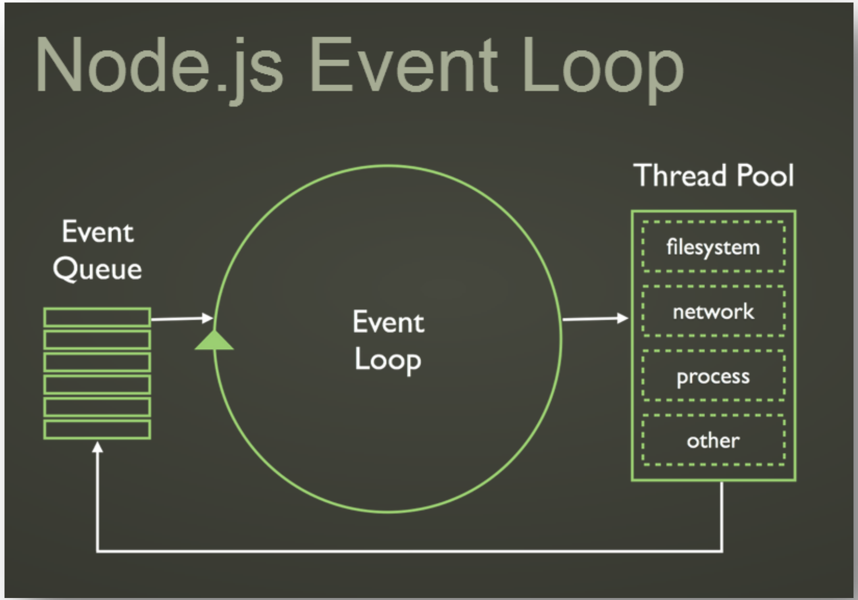
Requiring Modules, File System, Args

Node Architecture



Node Architecture

➢ Node handles requests with a **single** thread (the event loop)!

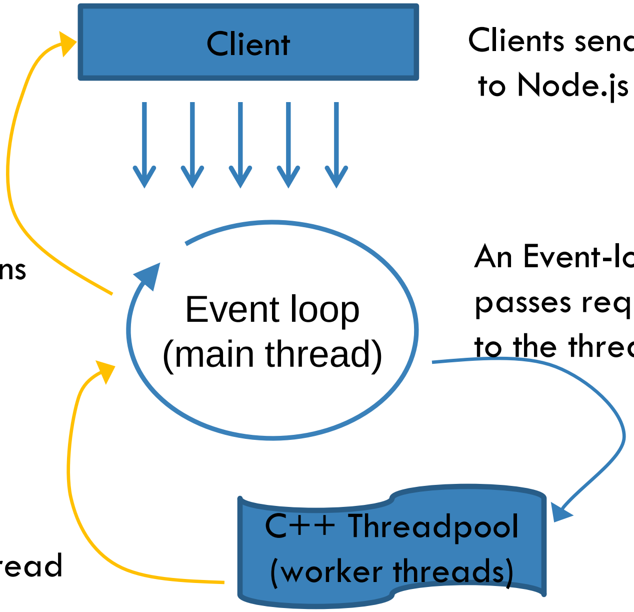


Node Architecture

- ➢ How can this be faster?
  - ➢ Expensive I/O is moved off request thread (in traditional multi-threaded environments, each thread handles the complete request)
  - ➢ Threads are limited by RAM and are expensive to create
  - ➢ Avoids context switching
- ➢ Allows us to easily write asynchronous code without heavy thread management (synchronization, message passing, etc.)

Some Theory: Event-loops

➢Event-loops are the core of event-driven programming, almost all the UI programs use event-loops to track the user event, for example: Clicks, Ajax Requests etc.



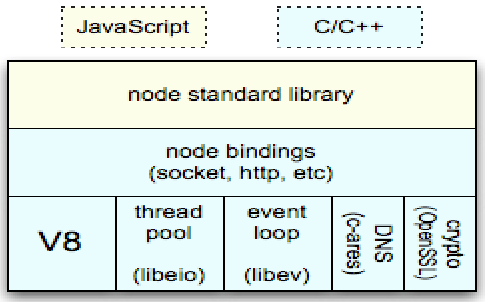
Clients send HTTP requests to Node.js server

Event loop returns result to client

An Event-loop is woken up by OS, passes request and response objects to the thread-pool

Long-running jobs run on worker threads

Response is sent back to main thread via callback



Some Theory: Non-Blocking I/O

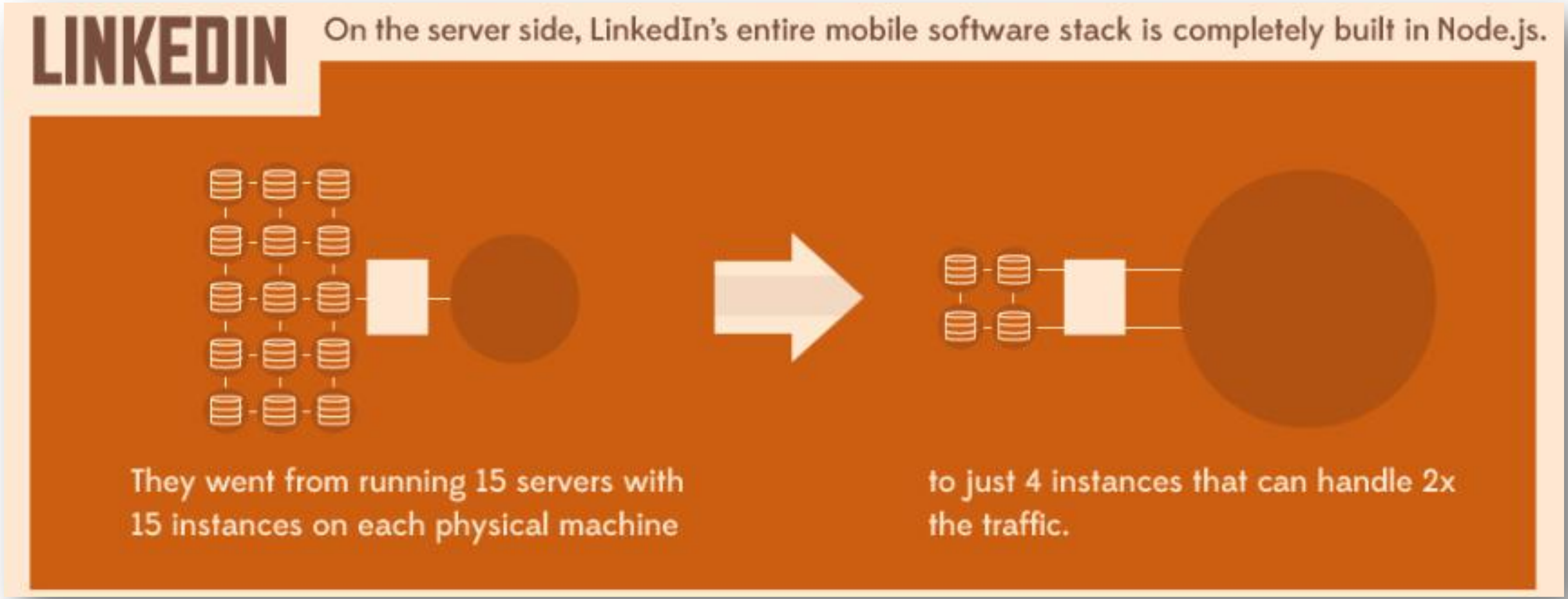
➢Traditional I/O

```
var result = db.query(“select x from table_Y”);
doSomethingWith(result); //wait for result!
doSomethingWithOutResult(); //execution is blocked!
```

➢Non-traditional, Non-blocking I/O

```
db.query(“select x from table_Y”,function (result){
       doSomethingWith(result); //wait for result!
});
doSomethingWithOutResult(); //executes without any delay!
```

Node Architecture



On the server side, LinkedIn's entire mobile software stack is completely built in Node.js.

They went from running 15 servers with 15 instances on each physical machine

to just 4 instances that can handle 2x the traffic.

Node Architecture

Warning! Be careful to keep CPU intensive operations off the event loop.

Hands On #**3** – **‘Real Time Data Flow’**

Http and Streams

NPM

- ➢ Modules can be shared by packaging
- ➢ Node Package Manager (NPM) is a package manager for node
  - ➢ Command line interface
  - ➢ Public registry www.npmjs.org
  - ➢ Allows us to install packages from repo, and publish our own

NPM

- ➢ First, we need to create a **package.json** file for our app
- ➢ Contains metadata for our app and lists the dependencies
- ➢ Package.json Interactive Guide

➢ Online repositories for node.js packages/modules which are searchable on search.nodejs.org

➢ Command line utility to install Node.js packages, do version management and dependency management of Node.js packages.

NPM

- Common npm commands:
  - **npm init** initialize a package.json file
  - **npm install <package name> -g** install a package, if –g option is given package will be installed as a global, **--save** and **--save-dev** will add package to your dependencies
  - **npm install** install packages listed in package.json
  - **npm ls –g** listed local packages (without –g) or global packages (with –g)
  - **npm update <package name>** update a package

NPM

- ➢ SemVar Versions (version [Major].[Minor].[Patch]):
  - ➢ **= (default), >, <, >=, <=**
  - ➢ ***** most recent version
  - ➢ **1.2.3 – 2.3.4** version greater than 1.2.3 and less than 2.3.4
  - ➢ **~1.2.3** most recent patch version greater than or equal to 1.2.3 (>=1.2.3 <1.3.0)
  - ➢ **^1.2.3** most recent minor version greater than or equal to 1.2.3 (>=1.2.3 <2.0.0)

Hands On #4 – **‘Services’**

Http and Express

HTTP Server

Following code creates an **HTTP** Server and prints ‘*Hello World*’ on the browser:

```
var http = require('http');
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/plain'});
res.end('Hello World\n'); }).listen(5000, "127.0.0.1");
```

Events

1. addListener()
2. removeListener()
3. removeAllListeners
4. once()
5. on()
6. emit()

Events

```
var data = '';
req
  .on('data', function(chunk) {
    data += chunk;
  })
  .on('end', function() {
    console.log('POST data: %s', data);
  })
```

Events

```
var onData = function(chunk) {
  console.log(chunk);
  req.removeListener(onData);
}

req.on('data', onData);
```

File System

```
fs.readFile('/etc/passwd', function (err, data) {
  if (err) throw err;
  console.log(data);
});
```

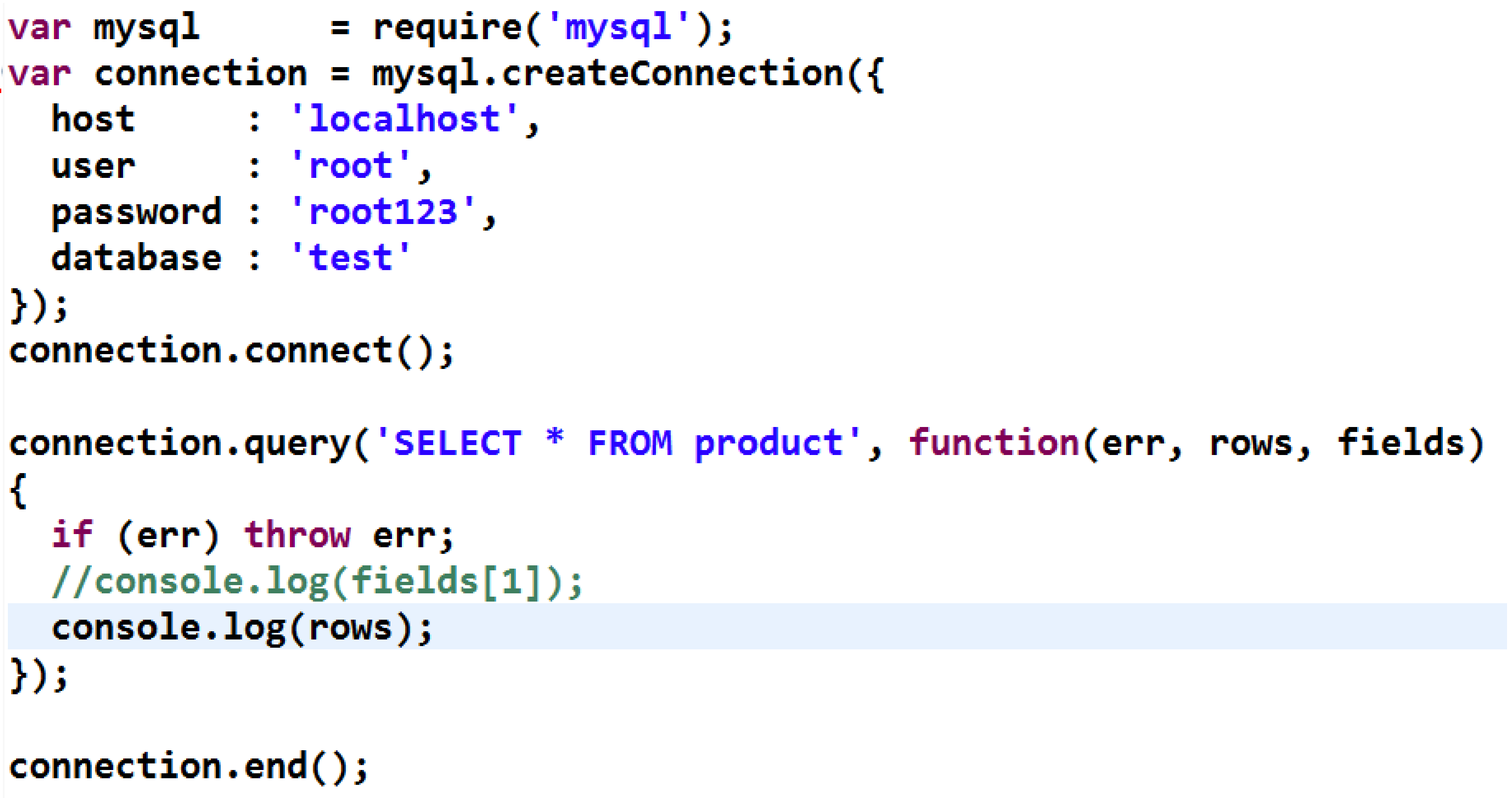
File System

```
fs.writeFile('message.txt', 'Hello Node.js', function (err) {
  if (err) throw err;
  console.log('It\'s saved!');
});
```

File System

```
fs.appendFile('message.txt', 'data to append', function (err) {
  if (err) throw err;
  console.log('The "data to append" was appended to file!');
});
```

NodeJS for DB support



```
var mysql      = require('mysql');
var connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'root',
  password : 'root123',
  database : 'test'
});
connection.connect();

connection.query('SELECT * FROM product', function(err, rows, fields)
{
  if (err) throw err;
  //console.log(fields[1]);
  console.log(rows);
});

connection.end();
```

NodeJS for REST support

```
var express = require('express');
var app = express();
var fs = require("fs");
app.get('/listUsers', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
       console.log( data );
       res.end( data );
   });
})
```

NodeJS process global

```
process.on('exit', function(code) {

  // Following code will never execute.
  setTimeout(function() {
    console.log("This will not run");
  }, 0);

  console.log('About to exit with code:', code);
});
console.log("Program Ended");
```

NodeJS process global

```
// Printing to console
process.stdout.write("Hello World!" + "\n");

// Reading passed parameter
process.argv.forEach(function(val, index, array) {
   console.log(index + ': ' + val);
});

// Getting executable path
console.log(process.execPath);


// Platform Information
console.log(process.platform);
```

NodeJS process global

```
// Print the current directory
console.log('Current directory: ' + process.cwd());

// Print the process version
console.log('Current version: ' + process.version);

// Print the memory usage
console.log(process.memoryUsage());
```

NodeJS process global

```
process.stdin.resume();

process.stdin.pipe(process.stdout, { end:false});

process.stdin.on("end", function() {
  process.stdout.write("Goodbye\n");
});
process.stdin.emit("end");
```

NodeJS process global –scaling App

```
const fs = require('fs');
const child_process = require('child_process');

for(var i=0; i<3; i++) {
   var workerProcess = child_process.spawn('node', ['support.js', i]);

   workerProcess.stdout.on('data', function (data) {
      console.log('stdout: ' + data);
   });

   workerProcess.stderr.on('data', function (data) {
      console.log('stderr: ' + data);
   });

   workerProcess.on('close', function (code) {
      console.log('child process exited with code ' + code);
   });
}
```

NodeJS process global –scaling App

Support.js->

```
console.log("Child Process " + process.argv[2] + " executed." );
```

Output->

```
stdout: Child Process 0 executed.

child process exited with code 0
stdout: Child Process 1 executed.

stdout: Child Process 2 executed.

child process exited with code 0
child process exited with code 0
```

NodeJS process global –scaling App -fork

```
const fs = require('fs');
const child_process = require('child_process');

for(var i=0; i<3; i++) {
   var worker_process = child_process.fork("support.js", [i]);

   worker_process.on('close', function (code) {
      console.log('child process exited with code ' + code);
   });
}
```

Output->

```
Child Process 0 executed.
Child Process 1 executed.
Child Process 2 executed.
child process exited with code 0
child process exited with code 0
child process exited with code 0
```

NodeJS TCP Server

```
var net = require('net');

var HOST = '127.0.0.1';
var PORT = 6969;

net.createServer(function(sock) {
console.log('CONNECTED: ' + sock.remoteAddress +':'+ sock.remotePort);
sock.on('data', function(data) {
    console.log('DATA ' + sock.remoteAddress + ': ' + data);
sock.write('You said "' + data + '"');
  });
sock.on('close', function(data) {
    console.log('CLOSED: ' + sock.remoteAddress +' '+ sock.remotePort);
  });
}).listen(PORT, HOST);
console.log('Server listening on ' + HOST +':'+ PORT);
```

NodeJS Routing

```
function route(handle, pathname, response, postData) {
console.log("About to route a request for " + pathname);
if (typeof handle[pathname] === 'function') {
handle[pathname](response, postData);
} else {
console.log("No request handler found for " + pathname);
response.writeHead(404, {"Content-Type": "text/plain"});
response.write("404 Not found");
response.end();
}
}
```

NodeJS Routing

```
var handle = {}
handle["/"] = requestHandlers.start;
handle["/start"] = requestHandlers.start;

handle["/upload"] = requestHandlers.upload;
server.start(router.route, handle);
```

NodeJS TCP Client

```
var net = require('net');

var HOST = '127.0.0.1';
var PORT = 6969;

var client = new net.Socket();
client.connect(PORT, HOST, function() {
    console.log('CONNECTED TO: ' + HOST + ':' + PORT);
client.write('I am Habib!');
});
client.on('data', function(data) {
    console.log('DATA: ' + data);
client.destroy();
});
client.on('close', function() {
    console.log('Connection closed');
});
```

NodeJS with JQuery

```
var $ = require('jquery');
var http = require('http');
var options = {
host: 'jquery.com',
port: 80,
path: '/'
};
var html = '';
http.get(options, function(res) {
res.on('data', function(data) {
// collect the data chunks to the variable named "html"
html += data;
}).on('end', function() {
// the whole of webpage data has been collected. parsing time!
var title = $(html).find('title').text();
console.log(title);
});
});
```

Who is using Node.js in production?

➢**Yahoo!** : iPad App **Livestand** uses Yahoo! Manhattan framework which is based on Node.js.

➢**LinkedIn** : LinkedIn uses a combination of Node.js and MongoDB for its mobile platform. iOS and Android apps are based on it.

➢ **eBay** : Uses Node.js along with ql.io to help application developers in improving eBay's end user experience.

➢**Dow Jones** : The WSJ Social front-end is written completely in Node.js, using Express.js, and many other modules.

➢Complete list can be found at: https://github.com/joyent/node/wiki/Projects,-Applications,-and-Companies-Using-Node

Some Good Modules

- ➢**Express** – to make things simpler e.g. syntax, DB connections.
- ➢**Jade** – HTML template system (hbs)
- ➢**Socket.IO** – to create real-time apps
- ➢**Nodemon** – to monitor Node.js and push change automatically
- ➢**CoffeeScript** – for easier JavaScript development
- ➢Find out more about some widely used Node.js modules at: http://blog.nodejitsu.com/top-node-module-creators

THANKS
A
TON

Thanks!

Learn More at

http://nodejs.org/