Node.js Introduction

What is Node.js?

- Node.js is an open source server environment
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server

Why Node.js?

Node.js uses asynchronous programming!

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

What is a Node.js File?

- Node.js files contain tasks that will be executed on certain events
- A typical event is someone trying to access a port on the server
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js

Node.js Get Started

Download Node.js

The official Node.js website has installation instructions for Node.js: https://nodejs.org

---

Getting Started

Once you have downloaded and installed Node.js on your computer, let's try to display "Hello World" in a web browser.

Create a Node.js file named "myfirst.js", and add the following code:

myfirst.js

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



http.createServer(function (req, res) {

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

  res.end('Hello World!');

}).listen(8080);
```

Save the file on your computer: C:\Users\Your Name\myfirst.js

The code tells the computer to write "Hello World!" if anyone (e.g. a web browser) tries to access your computer on port 8080.

For now, you do not have to understand the code. It will be explained later.

Command Line Interface

Node.js files must be initiated in the "Command Line Interface" program of your computer.

How to open the command line interface on your computer depends on the operating system. For Windows users, press the start button and look for "Command Prompt", or simply write "cmd" in the search field.

Navigate to the folder that contains the file "myfirst.js", the command line interface window should look something like this:

```
C:\Users\Your Name>_
```

Initiate the Node.js File

The file you have just created must be initiated by Node.js before any action can take place.

Start your command line interface, write `node myfirst.js` and hit enter:

Initiate "myfirst.js":

```
C:\Users\Your Name>node myfirst.js
```

Now, your computer works as a server!

If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!

Start your internet browser, and type in the address: http://localhost:8080

What is a Module in Node.js?

Consider modules to be the same as JavaScript libraries.

A set of functions you want to include in your application.

Built-in Modules

Node.js has a set of built-in modules which you can use without any further installation.

Look at our Built-in Modules Reference for a complete list of modules.

Include Modules

To include a module, use the `require()` function with the name of the module:

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

Now your application has access to the HTTP module, and is able to create a server:

```
http.createServer(function (req, res) {

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

  res.end('Hello World!');
}).listen(8080);
```

Create Your Own Modules

You can create your own modules, and easily include them in your applications.

The following example creates a module that returns a date and time object:

```
exports.myDateTime = function () {

  return Date();
};
```

Use the `exports` keyword to make properties and methods available outside the module file.

Save the code above in a file called "myfirstmodule.js"

ADVERTISEMENT

Include Your Own Module

Now you can include and use the module in any of your Node.js files.

Example

Use the module "myfirstmodule" in a Node.js file:

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

var dt = require('./myfirstmodule');
```

```
http.createServer(function (req, res) {

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

  res.write("The date and time are currently: " + dt.myDateTime());

  res.end();

}).listen(8080);
```

Run example »

Notice that we use `./` to locate the module, that means that the module is located in the same folder as the Node.js file.

Save the code above in a file called "demo_module.js", and initiate the file:

Initiate demo_module.js:

```
C:\Users\Your Name>node demo_module.js
```

If you have followed the same steps on your computer, you will see the same result as the example: http://localhost:8080