Introduction

To start using Node.js, you must first understand the differences between Node.js and traditional server-side scripting environments (eg: PHP, Python, Ruby, etc).

Asynchronous Programming

> *Node.js uses a module architecture to simplify the creation of complex applications.*

Chances are good that you are familiar with asynchronous programming; it is, after all, the "A" in Ajax. Every function in Node.js is asynchronous. Therefore, everything that would normally block the thread is instead executed in the background. This is the most important thing to remember about Node.js. For example, if you are reading a file on the file system, you have to specify a callback function that is executed when the read operation has completed.

You are Doing Everything!

Node.js is only an environment - meaning that you have to do everything yourself. There is not a default HTTP server, or any server for that matter. This can be overwhelming for new users, but the payoff is a high performing web app. One script handles all communication with the clients. This considerably reduces the number of resources used by the application. For example, here is the code for a simple Node.js application:

```
var i, a, b, c, max;

max = 1000000000;

var d = Date.now();

for (i = 0; i < max; i++) {
    a = 1234 + 5678 + i;
    b = 1234 * 5678 + i;
    c = 1234 / 2 + i;
}

console.log(Date.now() - d);
```

And here is the equivalent written in PHP:

```
$a = null;
$b = null;
$c = null;
$i = null;
```

```
$max = 1000000000;

$start = microtime(true);

for ($i = 0; $i < $max; $i++) {
    $a = 1234 + 5678 + $i;
    $b = 1234 * 5678 + $i;
    $c = 1234 / 2 + $i;
}

var_dump(microtime(true) - $start);
```

Now let's look at the benchmark numbers. The following table lists the response times, in milliseconds, for these two simple applications:

Number of iterations	Node.js	PHP
100	2.00	0.14
10'000	3.00	10.53
1'000'000	15.00	1119.24
10'000'000	143.00	10621.46
1'000'000'000	11118.00	1036272.19

I executed the two apps from the command line so that no server would delay the apps' execution. I ran each test ten times and averaged the results. PHP is notably faster with a smaller amount of iterations, but that advantage quickly dissolves as the number of iterations increases. When all is said and done, PHP is 93% slower than Node.js!

Node.js is fast, but you will need to learn a few things in order to use it properly.

Modules

Node.js uses a module architecture to simplify the creation of complex applications. Modules are akin to libraries in C, or units in Pascal. Each module contains a set of functions related to the "subject" of the module. For example, the *http*module contains functions specific to HTTP. Node.js provides a few core modules out of the box to help you access files on the file system, create HTTP and TCP/UDP servers, and perform other useful functions.

Including a module is easy; simply call the `require()` function, like this:

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

> *Node.js is only an environment; you have to do everything yourself.*

The `require()` function returns the reference to the specified module. In the case of this code, a reference to the *http*module is stored in the `http` variable.

In the above code, we passed the name of a module to the `require()` function. This causes Node to search for a *node_modules* folder in our application's directory, and search for the *http* module in that folder. If Node does not find the *node_modules* folder (or the *http* module within it), it then looks through the global module cache. You can also specify an actual file by passing a relative or absolute path, like so:

```
var myModule = require('./myModule.js');
```

Modules are encapsulated pieces of code. The code within a module is mostly private - meaning that the functions and variables defined within them are only accessible from the inside of the module. You can, however, expose functions and/or variables to be used outside of the module. To do so, use the `exports` object and populate its properties and methods with the pieces of code that you want to expose. Consider the following module as an example:

```
var PI = Math.PI;

exports.area = function (r) {
  return PI * r * r;
};

exports.circumference = function (r) {
  return 2 * PI * r;
};
```

This code creates a `PI` variable that can only be accessed by code contained within the module; it is not accessible outside of the module. Next, two functions are created on the `exports`object. These functions are accessible outside of the module because they are defined on the `exports` object. As a result, `PI` is completely protected from outside interference. Therefore, you can rest assured that `area()` and `circumference()` will always behave as they should (as long as a value is supplied for the `r` parameter).

Global Scope

Node is a JavaScript environment running in Google's V8 JavaScript engine. As such, we should follow the best practices that we use for client-side development. For example, we should avoid putting anything into the global scope. That, however, is not always possible. The global scope in Node is `GLOBAL` (as opposed to `window` in the browser), and you can easily create a global variable of function by omitting the `var` keyword, like this:

```
globalVariable = 1;
globalFunction = function () { ... };
```

Once again, globals should be avoided whenever possible. So be careful and remember to use `var` when declaring a variable.

Installation

Naturally, we need to install Node before we can write and execute an app. Installation is straight forward, if you use Windows or OS X; the nodejs.org website offers installers for those operating systems. For Linux, use any package manager. Open up your terminal and type:

```
sudo apt-get update
sudo apt-get install node
```

or:

```
sudo aptitude update
sudo aptitude install node
```

Node.js is in sid repositories; you may need to add them to your sources list:

```
sudo echo deb http://ftp.us.debian.org/debian/ sid main > /etc/apt/sour
```

But be aware that installing sid packages on older systems may break your system. Be careful, and remove `/etc/apt/sources.list.d/sid.list` after you finish installing Node.

Installing New Modules

Node.js has a package manager, called Node Package Manager (NPM). It is automatically installed with Node.js, and you use NPM to install new modules. To install a module, open your terminal/command line, navigate to the desired folder, and execute the following command:

```
npm install module_name
```

It doesn't matter what OS you have; the above command will install the module you specify in place of *module_name*.

The Hello World App

Naturally, our first Node.js script will print the text `'Hello World!'` to the console. Create a file, called *hello.js*, and type the following code:

```
console.log('Hello World!');
```

Now let's execute the script. Open the terminal/command line, navigate to the folder that contains *hello.js*, and execute the following command:

```
node hello.js
```

You should see `'Hello World!'` displayed in the console.

HTTP Server

Let's move on to a more advanced application; it's not as complicated as you may think. Lets start with the following code. Read the comments and then the explanation below:

```
// Include http module.
var http = require("http");

// Create the server. Function passed as parameter is called on every r
// request variable holds all request parameters
// response variable allows you to do anything with response sent to th
http.createServer(function (request, response) {
	// Attach listener on end event.
	// This event is called when client sent all data and is waitin
	request.on("end", function () {
		// Write headers to the response.
		// 200 is HTTP status code (this one means success)
		// Second parameter holds header fields in object
		// We are sending plain text, so Content-Type should be
		response.writeHead(200, {
			'Content-Type': 'text/plain'
		});
		// Send data and end response.
		response.end('Hello HTTP!');
	});
// Listen on the 8080 port.
}).listen(8080);
```

This code is very simple. You can send more data to the client by using the `response.write()` method, but you have to call it before calling `response.end()`. Save this code as *http.js* and type this into your console:

```
node http.js
```

Open up your browser and navigate to `http://localhost:8080`. You should see the text "Hello HTTP!" in the page.

Handling URL Parameters

As I mentioned earlier, we have to do everything ourselves in Node, including parsing request arguments. This is, however, fairly simple. Take a look at the following code:

```
// Include http module,
var http = require("http"),
// And url module, which is very helpful in parsing request parameters.
        url = require("url");

// Create the server.
http.createServer(function (request, response) {
        // Attach listener on end event.
        request.on('end', function () {
                // Parse the request for arguments and store them in _g
                // This function parses the url from request and return
                var _get = url.parse(request.url, true).query;
                // Write headers to the response.
                response.writeHead(200, {
                        'Content-Type': 'text/plain'
                });
                // Send data and end response.
                response.end('Here is your data: ' + _get['data']);
        });
// Listen on the 8080 port.
}).listen(8080);
```

This code uses the `parse()` method of the *url* module, a core Node.js module, to convert the request's URL to an object. The returned object has a `query` property, which retrieves the URL's parameters. Save this file as *get.js* and execute it with the following command:

```
node get.js
```

Then, navigate to `http://localhost:8080/?data=put_some_text_here` in your browser. Naturally, changing the value of the `data` parameter will not break the script.

Reading and Writing Files

To manage files in Node, we use the *fs* module (a core module). We read and write files using the `fs.readFile()` and `fs.writeFile()` methods, respectively. I will explain the arguments after the following code:

```
// Include http module,
var http = require("http"),
// And mysql module you've just installed.
        fs = require("fs");

// Create the http server.
http.createServer(function (request, response) {
        // Attach listener on end event.
        request.on("end", function () {
                // Read the file.
                fs.readFile("test.txt", 'utf-8', function (error, data)
                        // Write headers.
                        response.writeHead(200, {
                                'Content-Type': 'text/plain'
                        });
                        // Increment the number obtained from file.
                        data = parseInt(data) + 1;
                        // Write incremented number to file.
                        fs.writeFile('test.txt', data);
                        // End response with some nice message.
                        response.end('This page was refreshed ' + data
                });
        });
// Listen on the 8080 port.
}).listen(8080);
```

> *Node.js has a package manager, called Node Package Manager (NPM). It is automatically installed with Node.js*

Save this as *files.js*. Before you run this script, create a file named *test.txt* in the same directory as *files.js*.

This code demonstrates the `fs.readFile()` and `fs.writeFile()` methods. Every time the server receives a request, the script reads a number from the file, increments the number, and writes the new number to the file. The `fs.readFile()` method accepts three arguments: the name of file to read, the expected encoding, and the callback function.

Writing to the file, at least in this case, is much more simple. We don't need to wait for any results, although you would check for errors in a real application.
The `fs.writeFile()` method accepts the file name and data as arguments. It also accepts third and fourth arguments (both are optional) to specify the encoding and callback function, respectively.

Now, let's run this script with the following command:

```
node files.js
```

Open it in browser (`http://localhost:8080`) and refresh it a few times. Now, you may think that there is an error in the code because it seems to increment by two. This isn't an error. Every time you request this URL, two requests are sent to the server. The first request is automatically made by the browser, which requests *favicon.ico*, and of course, the second request is for the URL (`http://localhost:8080`).

Even though this behavior is technically not an error, it is behavior that we do not want. We can fix this easily by checking the request URL. Here is the revised code:

```
// Include http module,
var http = require("http"),
// And mysql module you've just installed.
        fs = require("fs");

// Create the http server.
http.createServer(function (request, response) {
        // Attach listener on end event.
        request.on('end', function () {
                // Check if user requests /
                if (request.url == '/') {
                        // Read the file.
                        fs.readFile('test.txt', 'utf-8', function (erro
                                // Write headers.
                                response.writeHead(200, {
                                        'Content-Type': 'text/plain'
                                });
                                // Increment the number obtained from f
                                data = parseInt(data) + 1;
                                // Write incremented number to file.
                                fs.writeFile('test.txt', data);
                                // End response with some nice message.
                                response.end('This page was refreshed '
                        });
                } else {
                        // Indicate that requested file was not found.
                        response.writeHead(404);
                        // And end request without sending any data.
                        response.end();
                }
        });
// Listen on the 8080 port.
}).listen(8080);
```

Test it now; it should work as expected.

Accessing MySQL Databases

Most traditional server-side technologies have a built-in means of connecting to and querying a database. With Node.js, you have to install a library. For this tutorial, I've picked the stable and easy to use *node-mysql*. The full name of this module is *mysql@2.0.0-alpha2* (everything after the @ is the version number). Open your console, navigate to the directory where you've stored your scripts, and execute the following command:

```
npm install mysql@2.0.0-alpha2
```

This downloads and installs the module, and it also creates the *node_modules* folder in the current directory. Now let's look at how we can use this in our code; see the following example:

```
// Include http module,
var http = require('http'),
// And mysql module you've just installed.
	mysql = require("mysql");

// Create the connection.
// Data is default to new mysql installation and should be changed acco
var connection = mysql.createConnection({
	user: "root",
	password: "",
	database: "db_name"
});

// Create the http server.
http.createServer(function (request, response) {
	// Attach listener on end event.
	request.on('end', function () {
		// Query the database.
		connection.query('SELECT * FROM your_table;', function
			response.writeHead(200, {
				'Content-Type': 'x-application/json'
			});
			// Send data as JSON string.
			// Rows variable holds the result of the query.
			response.end(JSON.stringify(rows));
		});
	});
// Listen on the 8080 port.
}).listen(8080);
```

Querying the database with this library is easy; simply enter the query string and callback function. In a real application, you should check if there were errors (the `error` parameter will not be `undefined` if errors occurred) and send response codes dependent upon the success or failure of the query. Also note that we have set the `Content-Type` to `x-application/json`, which is the valid MIME type for JSON.

The `rows` parameter contains the result of the query, and we simply convert the data in `rows` to a JSON structure using the `JSON.stringify()` method.

Save this file as *mysql.js*, and execute it (if you have MySQL installed, that is):

```
node mysql.js
```

Navigate to `http://localhost:8080` in your browser, and you should be prompted to download the JSON-formatted file.

Conclusion

> *Every function in Node.js is asynchronous.*

Node.js requires extra work, but the payoff of a fast and robust application is worth it. If you don't want to do everything on the lowest level, you can always pick some framework, such as >Express, to make it easier to develop applications.

Node.js is a promising technology and an excellent choice for a high load application. It has been proven by corporations, like Microsoft, eBay, and Yahoo. If you're unsure about hosting your website/application, you can always use a cheap VPS solution or various cloud-based services, such as Microsoft Azure and Amazon EC2. Both of these services provide scalable environments at a reasonable price.