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

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 Modules

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:

Example:

Create a module that returns the current date and time:

```
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"

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);
```

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
```

•Node.js HTTP Module

The Built-in HTTP Module

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

To include the HTTP module, use the `require()` method:

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

Node.js as a Web Server

The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

Use the `createServer()` method to create an HTTP server:

Example:

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

//create a server object:
http.createServer(function (req, res) {
  res.write('Hello World!'); //write a response to the client
  res.end(); //end the response
}).listen(8080); //the server object listens on port 8080
```

The function passed into the `http.createServer()` method, will be executed when someone tries to access the computer on port 8080.

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

Initiate demo_http.js:

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

Add an HTTP Header

If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

Example:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

The first argument of the `res.writeHead()` method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

Read the Query String

The function passed into the `http.createServer()` has a `req` argument that represents the request from the client, as an object (http.IncomingMessage object).

This object has a property called "url" which holds the part of the url that comes after the domain name:

demo_http_url.js

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

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

Initiate demo_http_url.js:

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

If you have followed the same steps on your computer, you should see two different results when opening these two addresses:

http://localhost:8080/summer

Will produce this result:

```
/summer
```

http://localhost:8080/winter

Will produce this result:

```
/winter
```

Split the Query String

There are built-in modules to easily split the query string into readable parts, such as the URL module.

Example:

Split the query string into readable parts:

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

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

Initiate demo_querystring.js:

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

The address:

http://localhost:8080/?year=2017&month=July

Will produce this result:

```
2017 July
```

•Node.js File System Module

Node.js as a File Server

The Node.js file system module allows you to work with the file system on your computer.

To include the File System module, use the `require()` method:

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

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

Read Files

The `fs.readFile()` method is used to read files on your computer.

Assume we have the following HTML file (located in the same folder as Node.js):

demofile1.html

```
<html>
<body>
<h1>My Header</h1>
<p>My paragraph.</p>
</body>
</html>
```

Create a Node.js file that reads the HTML file, and return the content:

Example:

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('demofile1.html', function(err, data) {
```

```
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

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

Initiate demo_readfile.js:

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

Create Files

The File System module has methods for creating new files:

- `fs.appendFile()`
- `fs.open()`
- `fs.writeFile()`

The `fs.appendFile()` method appends specified content to a file. If the file does not exist, the file will be created:

Example:

Create a new file using the appendFile() method:

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

fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

The `fs.open()` method takes a "flag" as the second argument, if the flag is "w" for "writing", the specified file is opened for writing. If the file does not exist, an empty file is created:

Example:

Create a new, empty file using the open() method:

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

```
fs.open('mynewfile2.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('Saved!');
});
```

The `fs.writeFile()` method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created:

Example:

Create a new file using the writeFile() method:

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

fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Update Files

The File System module has methods for updating files:

- `fs.appendFile()`
- `fs.writeFile()`

The `fs.appendFile()` method appends the specified content at the end of the specified file:

Example:

Append "This is my text." to the end of the file "mynewfile1.txt":

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

fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {
  if (err) throw err;
  console.log('Updated!');
});
```

The `fs.writeFile()` method replaces the specified file and content:

Example:

Replace the content of the file "mynewfile3.txt":

```
var fs = require('fs');
fs.writeFile('mynewfile3.txt', 'This is my text', function (err) {
  if (err) throw err;
  console.log('Replaced!');
});
```

Delete Files

To delete a file with the File System module, use the `fs.unlink()` method.

The `fs.unlink()` method deletes the specified file:

Example:

Delete "mynewfile2.txt":

```
var fs = require('fs');
fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

Rename Files

To rename a file with the File System module, use the `fs.rename()` method.

The `fs.rename()` method renames the specified file:

Example

Rename "mynewfile1.txt" to "myrenamedfile.txt":

```
var fs = require('fs');
fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {
  if (err) throw err;
  console.log('File Renamed!');
});
```

•Node.js URL Module

The Built-in URL Module

The URL module splits up a web address into readable parts.

To include the URL module, use the `require()` method:

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

Parse an address with the `url.parse()` method, and it will return a URL object with each part of the address as properties:

Example:

Split a web address into readable parts:

```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2017&month=february'

var qdata = q.query; //returns an object: { year: 2017, month: 'february'
}
console.log(qdata.month); //returns 'february'
```

Node.js File Server

Now we know how to parse the query string, and in the previous chapter we learned how to make Node.js behave as a file server. Let us combine the two, and serve the file requested by the client.

Create two html files and save them in the same folder as your node.js files.

summer.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Summer</h1>
<p>I love the sun!</p>
</body>
</html>
```

winter.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Winter</h1>
<p>I love the snow!</p>
</body>
</html>
```

Create a Node.js file that opens the requested file and returns the content to the client. If anything goes wrong, throw a 404 error:

demo_fileserver.js:

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

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

Remember to initiate the file:

Initiate demo_fileserver.js:

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

If you have followed the same steps on your computer, you should see two different results when opening these two addresses:

http://localhost:8080/summer.html

Will produce this result:

Summer

```
I love the sun!
```

http://localhost:8080/winter.html

Will produce this result:

Winter

```
I love the snow!
```

•Node.js NPM

What is NPM?

NPM is a package manager for Node.js packages, or modules if you like.

www.npmjs.com hosts thousands of free packages to download and use.

The NPM program is installed on your computer when you install Node.js

What is a Package?

A package in Node.js contains all the files you need for a module.

Modules are JavaScript libraries you can include in your project.

Download a Package

Downloading a package is very easy.

Open the command line interface and tell NPM to download the package you want.

I want to download a package called "upper-case":

Download "upper-case":

```
C:\Users\Your Name>npm install upper-case
```

Using a Package

Once the package is installed, it is ready to use.

Include the "upper-case" package the same way you include any other module:

```
var uc = require('upper-case');
```

Create a Node.js file that will convert the output "Hello World!" into upper-case letters:

Example:

```
var http = require('http');
var uc = require('upper-case');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(uc.upperCase("Hello World!"));
  res.end();
}).listen(8080);
```

•Node.js Upload Files

The Formidable Module

There is a very good module for working with file uploads, called "Formidable".

The Formidable module can be downloaded and installed using NPM:

```
C:\User\Your Name>npm install formidable
```

After you have downloaded the Formidable module, you can include the module in any application:

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

Upload Files

Now you are ready to make a web page in Node.js that lets the user upload files to your computer:

Step 1: Create an Upload Form

Create a Node.js file that writes an HTML form, with an upload field:

Example:

This code will produce an HTML form:

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('<form action="fileupload" method="post"
enctype="multipart/form-data">');
  res.write('<input type="file" name="filetoupload"><br>');
  res.write('<input type="submit">');
  res.write('</form>');
  return res.end();
}).listen(8080);
```

Step 2: Parse the Uploaded File

Include the Formidable module to be able to parse the uploaded file once it reaches the server.

When the file is uploaded and parsed, it gets placed on a temporary folder on your computer.

Example:

The file will be uploaded, and placed on a temporary folder:

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

http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      res.write('File uploaded');
      res.end();
    });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="fileupload" method="post"
enctype="multipart/form-data">');
    res.write('<input type="file" name="filetoupload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8080);
```

Step 3: Save the File

When a file is successfully uploaded to the server, it is placed on a temporary folder.

The path to this directory can be found in the "files" object, passed as the third argument in the `parse()` method's callback function.

To move the file to the folder of your choice, use the File System module, and rename the file:

Example:

Include the fs module, and move the file to the current folder:

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

http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      var oldpath = files.filetoupload.filepath;
      var newpath = 'C:/Users/Your Name/' +
files.filetoupload.originalFilename;
      fs.rename(oldpath, newpath, function (err) {
        if (err) throw err;
        res.write('File uploaded and moved!');
        res.end();
      });
 });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="fileupload" method="post"
enctype="multipart/form-data">');
    res.write('<input type="file" name="filetoupload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8080);
```