Google | طويق 1000

Introduction to Node.js

أكاديمية طويق
TUWAIQ ACADEMY

وزارة الاتصالات وتقنية المعلومات
MINISTRY OF COMMUNICATIONS AND INFORMATION TECHNOLOGY

الاتحاد السعودي للأمن السيبراني والبرمجة والدرونز
SAUDI FEDERATION FOR CYBERSECURITY, PROGRAMMING & DRONES

مهارات المستقبل

stc

Objective

- Install Node.js.
- Learn what Node.js is
- The advantages of using Node.js
- Use existing Node.js modules
- Create Node.js modules

What is Node.js?

- ❖ Node.js was designed so that JavaScript can be used, not just in a browser on the client-side, but also on web servers
- ❖ Node.js is an
  - open-source (free)
  - cross-platform (Windows, Linux, Unix, Mac OS X, etc.)
  - runtime environment
- ❖ It can run JavaScript files without a browser
- ❖ Node.js makes full stack web development using JavaScript possible!

What is Node.js?

Advantages of using Node.js:

- ❖ Great performance
- ❖ The code is written in JavaScript
- ❖ The node package manager (NPM) provides access to hundreds of thousands of reusable packages (similar to react)
- ❖ It is portable and compatible with most operating systems
- ❖ It has a very active developer community
- ❖ Node.js runs single-threaded, asynchronous programming

What is Node.js?

To write applications or utilities with Node.js, we must:

- Install nodeJS.
- Access modules. To be able to run JavaScript on the server, we need some extra functionality (beyond the functionality that we use when we write code to be run in a browser).

Install Node.js.

You probably already have Node installed. Check this by typing `‘node -v’` into your command line interface. If a version number is displayed, Node is already installed!

Download Node.js and then install it. On Windows and Mac OS X:

- Go to https://nodejs.org/en/ to download the required installer.
- Click on the downloaded file and follow the installation prompts.

Access Modules

- Module: a unit of code that performs a specific task/achieves a specific goal.
- Node.js runs directly on a computer or server operating system so the environment omits browser-specific JavaScript APIs and adds support for more traditional OS APIs
- Browser APIs are tools like SessionStorage
- OS APIs are tools for things like file handling
- To include a module, use the `require()` method as shown with the following code:

```
const name_of_variable = require('name_of_module');
```

-

The HTTP Module

Step 1: Create a “hello.js” file

Step 2: Include required modules. To include the HTTP module, enter the following: const http = require('http’);

Step 3: Create a server object (i.e. createServer())

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

http.createServer(function(request, response) {
    response.write('Hello World!');
    response.end();
}).listen(3000);
```

The HTTP Module

Step 4: Save your Node.js file

Step 5: Initiate the Node.js file.

- Open your command line interface
- If necessary, change directory so that you are in the same directory as the one in which your Node.js file is stored. e.g: “cd NodeExamples”
- Type: node hello.js into the command line

The HTTP Module

Step 6: Start your browser and navigate to http://localhost:3000/. Voila! You should see your first node.js app in action! To quit the server, you can simply enter CTRL-BREAK (CTRL + C) or exit the command window.

The HTTP Module

Let’s summarise what you need to do to create a web server using Node.js. You will always have to:

1. Include all required modules.
2. Create a server object.
3. Save your Node.js file and
4. Initiate your Node.js file using the command line

The file system Module

Another built-in Node.js module, is the file system (fs) module. This module allows you to manipulate files on your computer. Some of the methods in this module are listed below:

- `fs.open()`
- `fs.readFile()`
- `fs.writeFile()`
- `fs.appendFile()`
- `fs.rename()`
- `fs.unlink()`

To use the file system module you will have to include it as you did with the HTTP module above. e.g.

```
      let fs = require(‘fs’);
```

The file system Module

It is important to remember that methods are called asynchronously. This could lead to errors:

```
fs.rename('/tmp/hello', '/tmp/world', (err) => {
  if (err) throw err;
  console.log('renamed complete');
});
fs.stat('/tmp/world', (err, stats) => {
  if (err) throw err;
  console.log(`stats: ${JSON.stringify(stats)}`);
});
```

The file system Module

To rectify this, use chained callbacks:

```
fs.rename('/tmp/hello', '/tmp/world', (err) => {
  if (err) throw err;
  fs.stat('/tmp/world', (err, stats) => {
    if (err) throw err;
    console.log(`stats: ${JSON.stringify(stats)}`);
  });
});
```

The Reading a file

Here is an example of reading a file on node:

```
let http = require('http');
let fs = require('fs');
http.createServer(function (req, res) {
  //Open a file on the server and return its content:
  fs.readFile('file.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

The Reading a file

Here is an example of writing to a file on node:

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

//Replace the file with a new one:
fs.writeFile('mynewfile3.txt', 'This is my text.', function
(err) {
  if (err) throw err;
  console.log('Replaced!');
});
```

The Reading a file

Here is an example of deleting a file on node:

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

//Delete the file.txt:
fs.unlink(‘file.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

Creating your own module

Create a JavaScript file that contains functions, but use the exports keyword to make the code available outside the module or JavaScript file.

❖ Create a module using the following code:

```
exports.myDateTime = function () {
    return Date();
};
```

❖ Save the file with the extension .js e.g.

`‘myFunction.js’`

Creating your own module

- ❖ Include the module you created where you want to use it as you would any other module, e.g.:

```
const dateTest = require('./myFunction.js’);
        console.log(dateTest.myDateTime());
```

- ❖ Save the code above in a file called ‘`test.js`’.
- ❖ Test the code from the command line by typing node test where ‘test’ is the name of the calling code. Make sure you are in the correct directory.

NPM

- There are many other libraries of code that you could access
- Use NPM, a package manager for Node.js
- See packages available:
  - Go to www.npmjs.com and create an account
  - You should see a number of free packages. You can ‘Discover packages’ by type, e.g. packages that deal with Math, CSS, robotics etc.
  - See what back-end packages are available.

NPM

- Step 1: Download the package using the CLI. To do this, type: npm install name_of_package, e.g. npm install chalk
- Step 2: Include the package. You do this just as you would include any other module, e.g. const chalk = require('chalk’);
- Step 3: Use the package.

```
const chalk = require('chalk');
const log = console.log;

//   Combine   styled   and   normal
strings
log(chalk.blue('Hello') + ' World'
+ chalk.red('!'));

//From
```

https://www.npmjs.com/package/chalk

Summary

- Install Node.js.
- Learn what Node.js is
- The advantages of using Node.js
- Use existing Node.js modules
- Create Node.js modules

Resources

- https://www.w3resource.com/javascript-exercises/javascript-functions-exercises.php
- https://www.w3resource.com/javascript-exercises/javascript-dom-exercises.php
- https://cs.lmu.edu/~ray/notes/javascriptfunctions/