Chapter 3

The Node.js API

Business Training

1

Outline

- Introduction to the Node.js API
- Utility functions (console,...)
- The Node.js module system
- IO : Paths, Files, and Folders
- Node.js Buffers and Streams
- HTTP/HTTPS

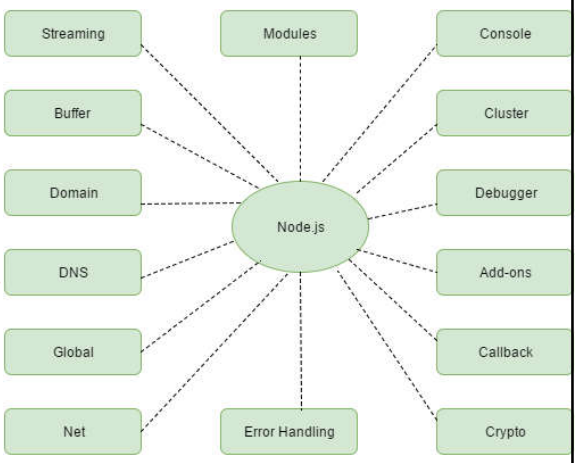
2

Introduction to the Node.js API

3

The Node.js standard API

- **The standard API of Node.js is organized in modules. It covers many domains:**
  - Utilities
  - Networking
  - IO and Streams
  - Streaming
  - …
- **It is installed with Node.js**
- **The documentation is provided here:** https://nodejs.org/dist/latest-v14.x/docs/api/



4

Node.js Utilities

5

Global objects

- **Object `process`**
  - The process object is a global that provides information about current process, and provides way to have control over it.
  - There are plenty of useful methods and event listeners as part of Process object.
    - process.pid : Get Current Process id.
    - process.argv: line arguments when the Node.js process was launched.
    - process.exit(): instructs Node.js to terminate the process synchronously
- **Object `console`**
  - print different levels of messages to stdout and stderr.

```
console.info("Program Started");
var counter = 10;
console.log("Counter: %d", counter);
console.time("Getting data");
// Do some processing here...
console.timeEnd('Getting data');
console.info("Program Ended")
```


6

Global objects

- **Object `__dirname`**
  - The directory name of the current module.

```
console.log(__dirname); // Note: 2 underscores as
prefix.
// The above line prints the full path
// to the directory of the current module.
```

- **Object `__filename`**
  - The file name of the current module.

```
console.log(__filename); // Note: 2 underscores as prefix.
// The above line prints the current module file's absolute path
// with symlinks resolved.
```


7

Global objects

- **Object `exports`**
  - **`exports`** or **`module.exports`** is used for defining what a module exports and makes available for other modules to import and use.

- **Function `require()`**
  - It is used to import the module(non-global) and make use of what has been exported from that module. require takes an id as an argument which is usually a module name or path.
  - It follows the CommonJS Module Pattern

```
const path = require("path");
```


8

The Node.js module system

9

The Module System Architecture

- **In Node every file is considered a module and before each file (module) is executed, it's wrapped within a Module Wrapper function which exposes the following variables/arguments module, exports, require, filename, dirname and looks something like;**

```
(function(exports, require, module, __filename, __dirname) {
  // module code goes in here
});
```

  - The `exports` and `module` object exposed by the wrapper function enable the module to expose functions/objects to be used by other modules.
  - The `require` object allows for the module to import other module(s)

- **The Node.js module system is an implementation of the CommonJS specification**

10

The Module System Architecture

- **Avoiding Naming Collision**
  - The variables exposed by the wrapper function are not globally scoped. Instead, they are locally scoped to the module
  - Variables declared within the module (in the global scope of the module) are also not directly accessible by other modules when the module is imported into another module except these variable are explicitly exported by the module
- **The module object**
  - The module object represents the file in which it exits.
  - To investigate this objects :
    - console.log(module);
- **The module.exports**
  - module.exports property exposes values from the module which can be imported into other modules by require('/path/to/module') and reused.

11

How Module loading works ?

- **The order of resolution starts from the most local towards the globally available modules**

```
'/Users/Max/repl/node_modules', <-- your node_modules in the app
folder
     '/Users/Max/node_modules', <-- node_modules in the home folder
     '/Users/node_modules', <-- Shared node modules
     '/node_modules',
     '/Users/Max/.node_modules',
     '/Users/Max/.node_libraries',
     '/usr/local/lib/node_modules/' <-- where all global modules
reside
```

- **If the module path to load begins with a relative path (“./”, “../”,or “/” it is load from the location without requiring a resolution procedure**
- **The loading module is a synchronous process**

12

CommonJS vs AMD vs ES Modules

- **Asynchronous Module Definition (AMD)**
  - AMD was born as CommonJS wasn't suited for the browsers early on. As the name implies, it supports asynchronous module loading.
  - It is designed to be used in browsers for better startup times and these modules can be objects, functions, constructors, strings, JSON, etc.

```
define(['module1', ',module2'], function(module1, module2) {
  console.log(module1.setName());
});
```

- **RequireJS**
  - RequireJS implements the AMD API. It loads the plain JavaScript files as well as modules by using plain script tags. It includes an optimizing tool which can be run while deploying our code for better performance.

13

CommonJS vs AMD vs ES Modules

- **ES Modules**
  - ECMAScript 6 a.k.a., ES6 a.ka., ES2015 offers possibilities for importing and exporting modules compatible with both synchronous and asynchronous modes of operation.
  - ECMAScript modules **are the official standard format** to package JavaScript code for reuse. Modules are defined using a variety of import and export statements.

```
//------ lib.js ------
export const sqrt = Math.sqrt;
export function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}

//------ main.js ------

import { square, diag } from 'lib';
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```


14

Node.js with ES 6 modules

- **Syntax change from CommonsJSto ES6 Modules**


```
const
  Http = require('http')
  ,Fs = require('fs')
  ,Path = require('path')
```

```
import * as Http from 'http'
import * as Fs from 'fs'
import * as Path from 'path'
```

- **To use ES 6 modules in Node.js apps:**
  - Node.js >= v13 : You need to either:
    - Save the file with .mjs extension, or
    - Add { "type": "module" } in the nearest package.json.
  - Node.js <= v12 : save the file with ES6 modules with .mjs extension and run it like:
    - node --experimental-modules my-app.mjs

15

I/O

16

Path

- **The path module provides a lot of very useful functionality to access and interact with the file system.**
  - Useful Path methods
    - path.basename(): Return the last portion of a path. A second parameter can filter out the file extension:
    - path.dirname() : Return the directory part of a path
    - path.join(): Joins two or more parts of a path
    - path.resolve() : You can get the absolute path calculation of a relative path using path.resolve()

Files and Directories

- **The fs module provides a lot of very useful functionality to access and interact with the file system.**
- **Reading a file**

```
// Open a Directory and list its files
fs.readdir(path.join(__dirname, 'lib'), (err, files) => {
    if (err) {
        console.error(`Err: ${err.message}`)
    } else {
        files.forEach(file => {
            console.log(`Open dir, File:  ${file}`);
        })
    }
})
```

- **Listing the content of a Directory**

```
// Reading a text file asynchrously
fs.readFile(filePath, 'utf8', (err, data) => {
    if (err) {
        console.error(err); return;
    }
    console.log(`\nReading a text file:`)
    console.log(`Contents of the data.json file : ${data}`)
})
```

Buffers and Streams

19

Buffers

- **The Buffer class in Node.js is designed to handle raw binary data.**
- **Creating Buffers:**
  - There are a few ways to create new buffers:
    - var buffer = Buffer.alloc(8); // This will print out 8 bytes of zero
    - var buffer = Buffer.from([8, 6, 7, 5, 3, 0, 9]);
- **Writing to Buffers**
  - > buffer.write("Hello", "utf-8")
- **Getting a Slice of Buffer**
  - >buffer.slice(0,2)

```
const buf = Buffer.from('Hey!')
buf.slice(0).toString() //Hey!
const slice = buf.slice(0, 2)
console.log(slice.toString()) //He
buf[1] = 111 //o
console.log(slice.toString()) //Ho
```


20

Streams

- **Instead of a program reading a file into memory all at once like in the traditional way, streams read chunks of data piece by piece, processing its content without keeping it all in memory.**
  - Streams are a way to handle reading/writing files, network communications, or any kind of end-to-end information exchange in an efficient way.
- **A stream pipeline example**

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

// Compression
function compress (source, target)  {
    let readableStream = fs.createReadStream(source, { encoding: 'utf8' });
    let compressedStream = zlib.createGzip();
    let writeableStream = fs.createWriteStream(target, { encoding: 'utf8' });
    readableStream.pipe(compressedStream).pipe(writeableStream);
}

compress('./demos/simpleText.txt', './demos/simpleText.txt.gz');
console.log("See the demos folder, and check that the gzip file has been generated");
```


21

HTTP/HTTPS

22

HTTP Server

- **Use http module to develop a HTTP Server**

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

const port = process.env.PORT

const server = http.createServer((req, res) => {
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/html')
  res.end('<h1>Hello, World!</h1>')
})

server.listen(port, () => {
  console.log(`Server running at port ${port}`)
})
```


23

How to pass the PORT from the Command line ?

- **From the command line**
  - MS DOS > SET PORT=3002 && node node-04-http-server.js
  - Powershell > $env:PORT=3338 ; node node-04-http-server.js
  - Bash SHELL > PORT=3333 node node-04-http-server.js
- **From env file :**
  - npm install dotenv --save
  - add this require : require('dotenv').config();

24

HTTP Server with routes

```
const http = require('http')
require('dotenv').config();

const PORT = process.env.PORT || 3000;

const server = http.createServer((req, res) => {
    if (req.method !== "GET") return error(res, 405);
    if (req.url === "/todo") return todo(res);
    if (req.url === "/") return index(res);
    error(res, 404);
});

function error(res, code) {
    res.statusCode = code;
    res.end(`{"error": "${http.STATUS_CODES[code]}"}`);
}
function todo(res) {
    res.end('[{"task_id": 1, "description": "walk dog"}]}');
}
function index(res) {
    res.end('{"name": "todo-server"}');
}
server.listen(PORT, () => {
    console.log(`Server listening on port ${server.address().port}`);
});
```


HTTPS

- **Generating a self signed X509 Certificate**
  - openssl genrsa -out privatekey.pem 1024
  - openssl req -new -key privatekey.pem -out certrequest.csr
  - openssl x509 -req -in certrequest.csr -signkey privatekey.pem -out certificate.pem

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


const options = {
  key: fs.readFileSync('certificates/privatekey.pem'),
  cert: fs.readFileSync('certificates/certificate.pem')
};

const port = process.env.PORT_HTTPS || 3443

const server = https.createServer(options, (req, res) => {

  res.statusCode = 200
  res.setHeader('Content-Type', 'text/html')

  res.end('<h1>From Secured Server Hello, World!</h1>')
})
server.listen(port, () => {
  console.log(`The Secure Server Server running at port ${port}`)
})
```