**Node.Js**: Asynchronous non-blocking I/O API

Sommaire

[1/How to load module in my js file? 2](#_Toc482864003)

[2/What can I do with the loaded module in my js file? 3](#_Toc482864004)

[3/How to route with Express? 4](#_Toc482864005)

[4/What are middlewares ? 6](#_Toc482864006)

# 1/How to load module in my js file?

**Module:** a javascript library (a .js file), to be used in another js file.

/app

mymodule4.js

/config

/myconfig.js

/project

/server.js

/mymodule1.js

/node\_modules

/http.js

/mymodule2.js

/other\_modules

/mymodule3.js

Examples to load a module in /project/server.js:

|  |  |
| --- | --- |
| file.js in /project/node\_modules | var http = require(‘http’)  var mm2 = require(‘mymodule2’) |
| file.js in /project | var mm1 = require(‘./mymodule1’) |
| file.js in /project/other\_modules | var mm3 = require(‘./other\_modules/mymodule3’) |
| file.js in /app | var mm4 = require(‘../mymodule4’) |
| file.js in /app/config | var cfg = require(‘../config/myconfig’) |

Absolute path begin with a **‘/’:**

**/**app/project/other\_modules

Relative path is relative to the file using require().

**./**mymodule2 🡪 the required file is in the same directory

**../**mymodule4 🡪 the required file is in the parent directory

var myvar = required(xxx) means ‘myvar’ will be an instance of the module.

# 2/What can I do with the loaded module in my js file?

The loaded modules in server.js can return anything: a boolean, an instance, a class, an array, a string, a function: we can even then pass argument when using require().

The loaded modules return **module.exports**. **By default**, module.exportsis an **empty object {}.**

module.exports as a container of properties:

Within the code of the module, use **module.exports.xxx** or **(**or shorthand**: exports.xxx)** to add properties to the default empty object. Those properties will then be available to the caller. ‘xxx’ could be anything (object, string, integer, function)

|  |
| --- |
| *mymodule1.js*  **module.exports**.myvariable = 3; //add ‘myvar’ property to the module.exports object  **exports**.saywow = function() {console.log(‘wow’);}; //add ‘saywow’ property to the module.exports object  // module.exports will look like this {myvariable:3, saywow: function() {console.log(‘wow’);}} |

|  |
| --- |
| *server.js*  var mm1 = require(‘./mymodule1’);  console.log(‘’+ mm1.myvariable); // will show ‘3’  mm1.saywow(); // will show ‘wow’ |

module.exports as a specific type object ( function, an object, a string, etc.) :

Within the code of the module, assign **module.exports** directly to anything: in that case, don’t mix it up with **exports.xxx**:any added properties using exports.xxx would be ignored and overwritten by the assignment of module.exports.

This behavior is the normal behavior of object assignment in javascript, since module.exports will now reference something else than the default object, while exports.xxx still refer to the default object.

|  |
| --- |
| *mymodule1.js*  **module.exports** = function(mystring) {console.log(‘’+ mystring);}; //the module will return a function |

|  |
| --- |
| *server.js*  var mm1 = require(‘./mymodule1’)(‘I pass an argument’); // will show ‘I pass an argument’  mm1(‘another argument’) // will show ‘another argument’ |

# 3/How to route with Express?

Express is a light framework to create API easily.

Routing is defining the application end points (URIs) and how they respond to client requests.

Express methods help to build routes and respond to those requests.

|  |
| --- |
| myserver.js  var express = require('express');// express is an instance of the module ‘express’  var app = express(); // express() is an instance of Express (a JS function actually)  var http = require(‘http’); // http is an instance of the module ‘http’, a built-in Node module  app.get(); // detailed later on  app.use(); // detailed later on  const server = http.createServer(app); /\* server is a Server instance. By passing app when creating server, we provide a callback that will handle requests. x in createServer(x) doesn’t have to be an Express application at all : it could just be any function (callback) with (req,res) parameters.\*/  server.listen(3000); // listen to the port 3000 |

The general syntax to define the app end points and how to respond is :

**app.METHOD(PATH, HANDLER) :**

* app is an instance of express()
* METHOD is a HTTP request method (get, post, etc.)
* PATH is the path on the server
* HANDLER is the function executed when the route is matched

Examples :

app.get('/', function (req, res) {

res.send('Hello World!')

})

app.post('/something', function (req, res) {

res.send('Got a POST request')

})

app.get('/example/b', function (req, res, next) {

console.log('the response will be sent by the next function ...')

next()

}, function (req, res) {

res.send('Hello from B!')

})

The methods on the response object (res) in the following table can send a response to the client, and terminate the request-response cycle. If none of these methods are called from a route handler, the client request will be left hanging.

| **Method** | **Description** |
| --- | --- |
| [res.download()](http://expressjs.com/en/4x/api.html#res.download) | Prompt a file to be downloaded. |
| [res.end()](http://expressjs.com/en/4x/api.html#res.end) | End the response process. |
| [res.json()](http://expressjs.com/en/4x/api.html#res.json) | Send a JSON response. |
| [res.jsonp()](http://expressjs.com/en/4x/api.html#res.jsonp) | Send a JSON response with JSONP support. |
| [res.redirect()](http://expressjs.com/en/4x/api.html#res.redirect) | Redirect a request. |
| [res.render()](http://expressjs.com/en/4x/api.html#res.render) | Render a view template. |
| [res.send()](http://expressjs.com/en/4x/api.html#res.send) | Send a response of various types. |
| [res.sendFile()](http://expressjs.com/en/4x/api.html#res.sendFile) | Send a file as an octet stream. |
| [res.sendStatus()](http://expressjs.com/en/4x/api.html#res.sendStatus) | Set the response status code and send its string representation as the response body. |

**app.route()**

app.route() is used when one path is used by different HTTP request methods :

app.route('/book')

.get(function (req, res) {

res.send('Get a random book')

})

.post(function (req, res) {

res.send('Add a book')

})

.put(function (req, res) {

res.send('Update the book')

})

**express.Router**

Use the express.Router class to create modular, mountable route handlers. A Router instance is a complete middleware and routing system; for this reason, it is often referred to as a “mini-app”.

The following example creates a router as a module, loads a middleware function in it, defines some routes, and mounts the router module on a path in the main app.

|  |
| --- |
| birds.js  var express = require('express')  var router = express.Router()  // middleware that is specific to this router  router.use(function timeLog (req, res, next) {  console.log('Time: ', Date.now())  next()  })  // define the home page route  router.get('/', function (req, res) {  res.send('Birds home page')  })  // define the about route  router.get('/about', function (req, res) {  res.send('About birds')  })  module.exports = router |

|  |
| --- |
| server.js  var birds = require('./birds')  // ...  app.use('/birds', birds) |

# 

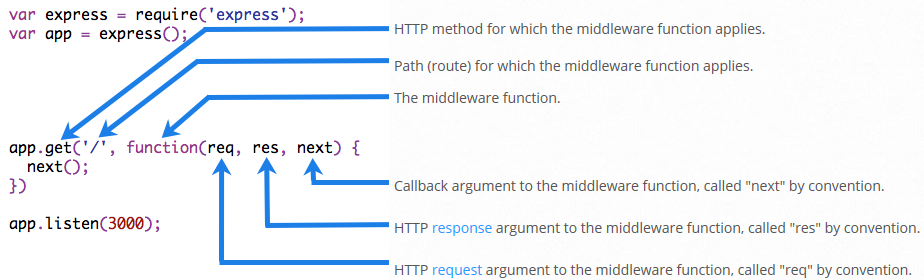
# 4/What are middlewares ?

Middleware are functions that have access to the request object and response object (req, res).

Middleware functions can perform the following tasks:

* Execute any code.
* Make changes to the request and the response objects.
* End the request-response cycle.
* Call the next middleware in the stack.

If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.



An Express application is essentially a series of middleware function calls.

The order of middleware loading is important: middleware functions that are loaded first are also executed first.

## Application-level middleware: middleware that bind to the app object

|  |
| --- |
| var express = require('express')  var app = express()  var requestTime = function (req, res, next) {  req.requestTime = Date.now()  next()  }  app.use(requestTime) //with no mount path, the middleware is executed for every request  app.get('/', function (req, res) {  var responseText = 'Hello World!<br>'  responseText += '<small>Requested at: ' + req.requestTime + '</small>'  res.send(responseText)  })  app.listen(3000) |

To skip the rest of the middleware functions from a middleware stack, call next('route') to pass control to the next route.

**NOTE**: next('route') will work only in middleware functions that were loaded by using the app.METHOD() or router.METHOD() functions.

|  |
| --- |
| app.get('/user/:id', function (req, res, next) {  // if the user ID is 0, skip to the next route  if (req.params.id === '0') next('route')  // otherwise pass the control to the next middleware function in this stack  else next()  }, function (req, res, next) {  // render a regular page  res.render('regular')  })  // handler for the /user/:id path, which renders a special page  app.get('/user/:id', function (req, res, next) {  res.render('special')  }) |

## Router-level middleware: middleware that bind to an instance of express.Router() – router.use() e.g.

Router middleware are pretty much the same as app middlewares.

To back-out from the router’s middlewares, use next(‘route**r**’) (do not confuse with next(‘route’)).

## Built-in middleware: express.static is the only built-in middle in Express

express.static is used to serve static assests like HTML files, images and so on.

const path = require('path');

app.use(express.static(path.join(\_\_dirname, 'dist')));

path.join joins all the given strings, separating them with a delimiter and normalizes the path. It will return a string.

\_\_dirname: The directory name of the current module

In the example above, path.join(\_\_dirname, 'dist') if use in server.js will return :

‘/app/project/dist’

If app.listen(3000), then images in ‘dist’ will be available like that : http://localhost:3000/kitten.jpg