

JavaScript en el Servidor: Node.js

Desarrollo de Aplicaciones en Entornos Web
Curso 2016/2017

Contenido

- Introducción a Node.js
- Módulos en Node.js
- Módulo HTTP
- Servidor web en Node.js
- Procesamiento de una petición
- Generación de una respuesta
- Uso de 'npm'
- Acceso a MySQL
- Framework Express

Introducción a Node.js

- Plataforma para el desarrollo de aplicaciones Javascript.
 - Antes: Javascript era soportado solo por los navegadores
 - Aplicaciones de cliente y **servidor** (servidor web).
 - e.j.: Ejecutar aplicación en Node.s *'node miApliacion.js'*
- 2009
- Basado en el motor de **Javascript V8** desarrollado por **Google**.
 - Soporta **ECMAScript 2015 (ES6)**
- Arquitectura Node.js:
 - Lenguaje Javascript
 - Módulos Node.js (servidores web, administración de archivos, protocolos de comunicaciones, etc..)

-
- <https://nodejs.org/>
 - Motor de ejecución Javascript
 - Dirigido por Eventos
 - Modelo de Entrada/Salida asíncrono (no bloqueante)
 - Ligero y eficiente

 - Última versión sujeta al LTS (Long-term Support):
6.10 (Boron)

 - Versión actual: 7.9

-
- Gestión de módulos y paquetes a través del *npm*
 - *npm* no es acrónimo de 'Node.js Package Manager' → bacrónimo (retroacrónimo)
 - <https://www.npmjs.com/>
 - instala, comparte y distribuye librerías/módulos JS
 - gestión de dependencias en proyectos JS

 - Módulos disponibles en la propia distribución de Node.js + módulos instalados desde *npm*.
 - e.j.: 'npm install mime' (instala el módulo .js MIME)

Módulos en Node.js

- Node.js utiliza el concepto de módulo en su desarrollo
 - Creación de librerías/módulos Javascript para componer una aplicación
- Un módulo contiene funciones, objetos y variables Javascript
 - Indicar qué será exportado para ser utilizado por otros programas.

Ejemplo módulo: matematica.js

```
var PI=3.14;
function sumar(x1,x2){return x1+x2;
function restar(x1,x2){return x1-x2;
function dividir(x1,x2){
    if (x2==0 {
        mostrarErrorDivision();
    }else{return x1/x2;}
}
function mostrarErrorDivision() {
    console.log('No se puede dividir por cero');
}
exports.sumar=sumar; exports.restar=restar;
exports.dividir=dividir; exports.PI=PI;
```

-
- variables, funciones, objetos accesibles desde otros archivos son exportados agregándolos al objeto ***exports***

```
exports.sumar=sumar;
```

- Para usar un módulo: ***require***

```
var mat=require('./matematica');  
console.log('La suma de 2+2=' + mat.sumar(2,2));  
console.log('El valor de PI=' + mat.PI);
```

- Módulos pueden ser una carpeta que contiene un conjunto de archivos y de subcarpetas.

❑ Módulo Operative System:

```
var os = require('os');  
console.log('Sistema operativo:'+os.platform());  
console.log('Versión del OS:'+os.release());  
console.log('Memoria total:'+os.totalmem()+ ' bytes');  
console.log('Memoria libre:'+os.freemem()+ ' bytes');
```

❑ Módulo File System:

‘permite acceder al sistema de archivos para poder leer sus contenidos y crear otros archivos o carpetas’

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

- Node.js basado en el uso de **programación asíncrona** en tareas de larga duración
 - no detener la ejecución del programa ante actividades que requieren mucho tiempo
- Módulo 'fs' implementa programación asincrónica para el manejo de ficheros: creación, lectura, etc.

Ejemplo módulo 'fs'

```
var fs = require('fs');
fs.writeFile('./archivo1.txt',
  'línea 1\nLínea 2',
  function(error){
    if (error) console.log(error);
    else console.log('El archivo
      fue creado');
  });
console.log('última línea del programa');
```

2°

1°

□ **writeFile:**

- *primer* parámetro: nombre del archivo de texto a crear (path absoluto o relativo)
- *segundo* parámetro: contenido (string) a escribir en el archivo
- *tercer* parámetro: función anónima que será llamada por la función `writeFile` cuando haya terminado de crear el archivo. **Asíncrona.**
 - La función recibe como parámetro un objeto con el **error** en caso de no poderse crear el archivo (en caso contrario el valor es *null*).

```
var fs = require('fs');  
function leer(error, datos) {  
    if (error) {  
        console.log(error);  
    } else {  
        console.log(datos.toString());  
    }  
}  
  
fs.readFile('./archivol.txt', leer);  
console.log('última línea del programa');
```

2°

1°

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

```
fs.exists(ruta_archivo,  
          function(existe) {  
            if (existe) { ... }  
            else { ... }  
          });
```

Módulo HTTP

- Módulo para implementar un servidor web (soporte HTTP)
 - Permite crear **aplicaciones web**

- En Node.js hay que implementar un servidor web
 - Diferente a las plataformas Java, PHP, Microsoft, etc. donde los servidores son provistos por terceros (Apache, IIS, etc..)

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

Servidor web en Node.js

- El módulo **'http'** tiene una función llamada **createServer** para *crear un servidor web basado en el protocolo HTTP*.
- *createServer* requiere una función con dos parámetros: **petición** y **respuesta**.
- Los objetos petición y respuesta son creados por la función *createServer* para dar acceso a la **petición y respuesta HTTP**.

Ejemplo servidor 'Hola Mundo'

```
var http = require('http');  
var servidor=http.createServer(  
  function (peticion, respuesta) {  
    respuesta.writeHead(200,  
      {'Content-Type': 'text/html'});  
    respuesta.write(  
      '<!doctype html><html><head></head>' +  
      '<body><h1>Hola Mundo</h1></body></html>');  
    respuesta.end();  
  });  
servidor.listen(8080);  
console.log('Servidor web iniciado');
```

-
- `createServer()` se ejecuta en forma asíncrona

 - `servidor.listen(8080)` también es asíncrona
 - se queda esperando a recibir peticiones.

 - *función* pasada al servidor es la encargada de actuar como *manejador de peticiones HTTP*.
 - En el ejemplo anterior, cualquier URL (petición) invocará a la función manejadora

Generación de una respuesta

- `respuesta.writeHead`
Escribir cabeceras en la respuesta HTTP

- `respuesta.write`
Escribir el contenido de la respuesta HTTP

- `respuesta.end`
Indicar que se ha finalizado de escribir el contenido
 - se puede invocar varias veces `respuesta.write` antes de `respuesta.end`

Procesamiento de una petición

- En cada petición de recurso se ejecuta la función pasada a *createServer*.
- El primer parámetro de la función manejadora permite el acceso a la petición HTTP.
- Acceso a la URL de la petición:
`petition.url`
 - Usar módulo 'url' para analizar la URL de la petición
`var url = require('url');`

```
http://localhost:8080/carpeta1/pagina1.html?param1=10&param2=20
```

```
var objetourl = url.parse(pedido.url);
```

```
console.log('path completo del recurso y  
parámetros: '+objetourl.path);
```

```
carpeta1/pagina1.html?param1=10&param2=20
```

```
console.log('solo el path y nombre del  
recurso : '+objetourl.pathname)
```

```
carpeta1/pagina1.html
```

```
console.log('parámetros del recurso :' +  
objetourl.query)
```

```
param1=10&param2=20
```

Acceso a *index.html* por defecto

```
var servidor = http.createServer(  
    function(peticion, respuesta){  
        var objetourl = url.parse(peticion.url);  
        var ruta='public'+objetourl.pathname;  
        if (ruta=='public/')  
            ruta='public/index.html';  
        procesarPeticion(peticion, respuesta,  
            ruta);  
    }  
);
```

discriminar por URL

...

Procesamiento GET

index.html

```
<html>
<head> <title>Formulario</title></head>
<body>
  <form action="recuperardatos" method="get">
    Nombre:<input type="text" name="nombre" size="30"><br>
    Clave: <input type="password" name="clave" size="30"><br>
  <input type="submit" value="Enviar"></form>
</body>
</html>
```

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

```
...
```

```
function procesarFormulario(peticion, respuesta) {
  var nombre = url.parse(peticion.url, true).query.nombre;
  var clave = url.parse(peticion.url, true).query.clave;
}
```

false: devuelve string
true: devuelve objeto parseado

Procesamiento POST

index.html

```
<html>
<head> <title>Formulario</title></head>
<body>
    <form action="recuperardatos" method="post">
        Nombre:<input type="text" name="nombre" size="30"><br>
        Clave: <input type="password" name="clave" size="30"><br>
    <input type="submit" value="Enviar"></form>
</body>
</html>
```

```
function procesarPeticion (peticion, respuesta,
    ruta) {
    switch (ruta) {
        case 'public/recuperardatos':{
            procesarFormulario(peticion, respuesta);
            break;
        }
    }
}
```

Procesamiento POST

```
var querystring = require('querystring');  
  
...  
function procesarFormulario(peticion, respuesta) {  
    var info = '';  
    peticion.on('data', function(datosparciales){  
        info += datosparciales;  
    });  
    peticion.on('end', function(){  
        var formulario = querystring.parse(info);  
        respuesta.writeHead(200, {'Content-Type': 'text/html'});  
        var pagina='...nombre:' + formulario['nombre'] + '<br>' +  
            'clave:' + formulario['clave'] + '<br> ...';  
        respuesta.end(pagina);  
    });  
}
```

```
{  
  nombre: 'javier',  
  clave: '1234'  
}
```

Uso de 'npm'

- Gestor de módulos/paquetes en Javascript para Node.js
 - Instalada con la distribución de Node.js

- Node.js viene por defecto con un conjunto limitado de módulos de uso general.

- Repositorio de módulos/paquetes en: <https://www.npmjs.com/>
 - 2015 : mas de 172.000 paquetes JS

Uso de 'npm'

□ Uso:

```
npm install nombre_modulo
```

- desde la carpeta de proyecto
- se crea una carpeta llamada ***node_modules/***
- en dicha carpeta encontraremos el módulo (nueva carpeta ***nombre_modulo/***)
- incluye documentación de uso

```
var modulo = require('nombre_modulo');
```

Acceso a MySQL

□ `npm install mysql`

```
var mysql = require('mysql');  
var conexion = mysql.createConnection(  
  {  
    host: 'localhost:3306',  
    user: 'root',  
    password: '',  
    database: 'daweb'  
  });
```

Acceso a MySQL

```
conexion.connect(function (error){
    if (error)
        console.log('Problemas de conexion con mysql');
});

var datosregistro = { nombre: formulario['nombre'],
    clave: formulario['clave'] };

conexion.query('insert into usuarios set ?',
    datosregistro,
    function (error, resultado){
        if (error){ console.log(error);
            return; }
    });
```

Acceso a MySQL

```
conexion.query(
    'select nombre, clave from usuarios',
    function (error, filas) {
        if (error) { console.log(error); return; }
        for (var i = 0; i < filas.length; i++) {
            ... filas[i].nombre ...
            ... filas[i].clave ...
        }
    });
...
var dato = [formulario['nombre']];
conexion.query('select clave from usuarios where nombre
    = ?', dato, function( ...
```

Framework Express

- Framework sobre Node.js para implementar sitios web.
 - Motivación: facilitar y organizar proyectos web en Node.js
 - Permite crear servidores web y manejar peticiones/respuestas con código más simple.

- Instalado desde *npm*:
`npm install express`

- Usar módulo Express

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

- Crear Aplicación servidora

```
var app = express();
```

- Definir manejadores de peticiones

```
app.get(ruta, funcion_manejadora);
```

```
app.put(ruta, funcion_manejadora);
```

- ruta: URL para la captura
- funcion_manejadora: manejador de la petición con los parámetros petición y respuesta. Asíncrona.

- Arrancar el servidor

```
var server = app.listen(puerto, funcion);
```

Ejemplo servidor Express : Hola Mundo

```
var express = require('express');
var app = express();

app.get('/',
  function (peticion, respuesta){
    respuesta.send('...</head><body><h1>' +
      'Hola Mundo</h1></body>...');
  }
);

var server = app.listen(8080, function(){
  console.log('Servidor web iniciado');
});
```

Procesamiento peticiones Express

index.html

```
<html>
<head> <title>Formulario</title></head>
<body>
  <form action="recuperardatos" method="post ó get">
    Nombre:<input type="text" name="nombre" size="30"><br>
    Clave: <input type="password" name="clave" size="30"><br>
  <input type="submit" value="Enviar"></form>
</body>
</html>
```

```
var bodyParser = require('body-parser');
//extended: false para parsear solo parámetros de tipo string
app.use(bodyParser.urlencoded({ extended: false }));
app.post('/recuperardatos', function (req, res) {
  var nombre = req.body.nombre;
  ...
app.get('/recuperardatos', function (req, res) {
  var nombre = req.query.nombre;
  ...
```

Generador de código en Express

- 'express-generator' : instalar desde la consola del sistema operativo:

```
npm install express-generator -g
```

- crear aplicación Node.js utilizando Express y su generador de código:

```
express ejemplo_aplicacion --hbs
```

- --hbs : referencia al sistema de plantillas usado para las vistas.
Handlebars <http://handlebarsjs.com>

Generador de código en Express

ejemplo_aplicacion/

app.js

package.json

bin/

www/

public/

images/, javascripts/, stylesheets/

routes/

index.js

users.js

views/

error.hbs

index.hbs

layout.hbs

package.json

```
{  
  "name": "ejemplo_aplicacion",  
  "version": "0.0.0", "private": true,  
  "scripts": {  
    "start": "node ./bin/www"  
  },  
  "dependencies": {  
    "body-parser": "~1.13.2",  
    "cookie-parser": "~1.3.5",  
    "debug": "~2.2.0", "express": "~4.13.1",  
    "hbs": "~3.1.0", "morgan": "~1.6.1",  
    "serve-favicon": "~2.3.0"  
  }  
}
```

Generador de código en Express

```
npm install
```

- ejecuta install sobre todas las dependencias (al estilo Maven)

```
npm start
```

- arranca el servidor en el puerto 3000
- toma como punto de arranque el script indicado:

```
"scripts": {  
  "start": "node ./bin/www"  
},
```

Generador de código en Express

app.js

```
var express = require('express');
var bodyParser = require('body-parser');
...
var routes = require('./routes/index');
var app = express();
...
app.use('/', routes);
...
```

Manejadores de petición

```
index.js
```

```
var express = require('express');  
var router = express.Router();  
  
router.get('/', function(req, res, next) {  
  res.render('index', { title: 'Express' });  
});  
  
module.exports = router;  
...
```

```
index.js*
```

```
...
```

```
var datos={  
  titulo:'Articulos disponibles a la fecha',  
  articulos: [  
    { codigo: 1,precio:12,descripcion: 'peras' },  
    { codigo: 2,precio:132,descripcion: 'manzanas' },  
    { codigo: 3,precio:23,descripcion: 'naranjas' },  
  ],  
  descuento:{lunes:'5%',martes:'10%'}  
};  
res.render('index', datos);
```

```
...
```

Plantillas de las vistas

```
index.hbs*
```

```
<h1>{{titulo}}</h1>
<table border="1">
  <tr><td>Codigo</td><td>Descripcion</td><td>Precio</td>
  </tr>
  {{#each articulos}}
  <tr>
    <td>{{codigo}} </td> <td>{{descripcion}}</td>
    <td>{{precio}}</td>
  </tr>
  {{/each}}
</table>
<p>Descuentos el día lunes:{{descuento.lunes}}</p>
<p>Descuentos el día martes:{{descuento.martes}}</p>
```

layout.hbs

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{titulo}}</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    {{{body}}}
  </body>
</html>
```