Desarrollo de servicios web con Node.js, Socket.io y MongoDB

Desarrollo de Sistemas Distribuidos

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada

2 de mayo de 2024

Indice

Sección 1 | Introducción a Node.js

Conceptos generales

- Node.js es una plataforma que permite desarrollar servicios web en JavaScript
- La programación sobre Node.js se hace de manera asíncrona:
  - Las funciones son no bloqueantes
  - Utiliza mecanismos como “callbacks” y “promises”

Callbacks

- Función que se pasa como argumento a otra función
- Se ejecuta cuando la función receptora ha concluido su ejecución
- Complicadas de gestionar cuando existen anidaciones múltiples
- Gestión de errores puede resultar complicada

```
function getData(callback) {
    setTimeout(() => {
      const data = 'Some data';
      callback(data);
    }, 1000);
  }

  getData((data) => {
    console.log(data);
  });
```

Promises

- Representan un valor que puede estar disponible ahora o en el futuro
- Pueden resolverse de forma satisfactoria o generar un error
- Admiten sintaxis encadadenada que permite manejar ambos estados con facilidad (then/catch)

```
function getData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const data = 'Some data';
      resolve(data);
    }, 1000);
  });
}

getData()
  .then((data) => {
    console.log(data);
  })
  .catch((error) => {
    console.log(error);
  });
```

Ejemplo Hola Mundo

```
import http from 'node:http';

http.createServer((request, response) => {
        console.log(request.headers);
        response.writeHead(200, {'Content-Type': 'text/plain'});
        response.write('Hola mundo');
        response.end();
    })
    .listen(8080);

console.log('Servicio HTTP iniciado');
```

Ejemplo Hola Mundo

- Instala Node.js y descomprime los ejemplos en el directorio de trabajo
- Lanza el ejemplo con: **$ node helloworld.js**
- Abre el navegador e introduce la url: `http://localhost:8080/`

Ejemplo Calculadora REST

```
import http from 'node:http';

function calcular(operacion, val1, val2) {
    if (operacion == 'sumar') return val1+val2;
    else if (operacion == 'restar') return val1-val2;
    else if (operacion == 'producto') return val1*val2;
    else if (operacion == 'dividir') return val1/val2;
    else return 'Error: Parámetros no válidos';
}

http.createServer((request, response) => {
        let {url} = request;
        url = url.slice(1);
        const params = url.split('/');
        let output='';
        if (params.length >= 3) {
            const val1 = parseFloat(params[1]);
            const val2 = parseFloat(params[2]);
            const result = calcular(params[0], val1, val2);
            output = result.toString();
        }
        else output = 'Error: El número de parámetros no es válido';

        response.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
        response.write(output);
        response.end();
    })
    .listen(8080);
console.log('Servicio HTTP iniciado');
```

Ejemplo Calculadora REST

- Lanza el ejemplo con: **$ node calculadora.js**
- Abre el navegador e introduce una url con el siguiente patrón: `http://localhost:8080/operacion/operando1/operando2`
- Ejemplo: `http://localhost:8080/sumar/5/7`

Ejemplo Calculadora Web

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="uft-8">
        <title>Calculadora</title>
    </head>
    <body>
        <form id="calculadora">
            Valor1: <input type="label" id="val1"><br>
            Valor2: <input type="label" id="val2"><br>
            Operación:
            <select id="operacion">
                <option value="sumar">Sumar</option>
                <option value="restar">Restar</option>
                <option value="producto">Producto</
         option>
                <option value="dividir">Dividir</option>
            </select><br>
            <input type="submit" value="Calcular">
        </form>
        <span id="resultado"></span>
    </body>
```

```
    <script type="text/javascript">
        const calc = document.getElementById('calculadora');

        calc.addEventListener('submit', (e) => {
            e.preventDefault();

            const serviceURL = document.URL;
            const val1 = document.getElementById('val1').value;
            const val2 = document.getElementById('val2').value;
            const oper = document.getElementById('operacion').value;

            const url  = serviceURL + oper + '/' + val1 + '/' +val2;

            fetch(url)
            .then(response => response.text())
            .then(response => {
                let resultado = document.getElementById('resultado');
                resultado.textContent = response;
            });
        });
    </script>
</html>
```

Ejemplo Calculadora Web

```
import http         from 'node:http';
import {join}       from 'node:path';
import {readFile}   from 'node:fs';

// function calcular: idéntica a calculadora rest

http.createServer((request, response) => {
        let {url} = request;
        if(url == '/') {
            url = '/calc.html';
            const filename = join(process.cwd(), url);

            readFile(filename, (err, data) => {
                if(!err) {
                    response.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
                    response.write(data);
                } else {
                    response.writeHead(500, {"Content-Type": "text/plain"});
                    response.write(`Error en la lectura del fichero: ${url}`);
                }
                response.end();
            });
        } else {
            // Caso para atender petición REST
            // Implementación idéntica al ejemplo anterior
        }
    })
    .listen(8080);
console.log('Servicio HTTP iniciado');
```

Ejemplo Calculadora Web

- Lanza el ejemplo con: **$ node calculadora-web.js**
- Al introducir la url `http://localhost:8080` se accede al cliente web. El formulario construye una petición REST idéntica al ejemplo anterior.
- Continúa admitiendo urls con el siguiente patrón: `http://localhost:8080/operacion/operando1/operando2`

Sección 2 | Introducción a Socket.io

Conceptos generales

- Socket.io (módulo de node.js) permite enviar notificaciones a los clientes de un servicio
- Los clientes mantienen una conexión ("websocket") con el servicio
- Cuando el servicio tiene datos nuevos, los notifica al cliente

Publish-subscribe (eventos)

- **'connect'**: conexión realizada correctamente.
- **'disconnect'**: desconexión entre cliente y servicio.
- **'connection_error'**: cierre de conexión inesperado.
- **'error'**
- **'reconnect'**: cliente se reconecta a un servicio con éxito.
- **'reconnect_attempt'**: cliente intenta conectarse de nuevo a un servicio.
- **'reconnect_failed'**: error de reconexión durante 'reconnect_attempt'.

Ejemplo Clientes conectados - Servidor (1)

```
import http         from 'node:http';
import {join}       from 'node:path';
import {readFile}   from 'node:fs';
import {Server}     from 'socket.io';

const httpServer = http
    .createServer((request, response) => {
        let {url} = request;
        if(url == '/') {
            url = '/connections.html';
            const filename = join(process.cwd(), url);

            readFile(filename, (err, data) => {
                if(!err) {
                    response.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
                    response.write(data);
                } else {
                    response.writeHead(500, {"Content-Type": "text/plain"});
                    response.write(`Error en la lectura del fichero: ${url}`);
                }
                response.end();
            });
        } else {
            console.log('Peticion invalida: ' + url);
            response.writeHead(404, {'Content-Type': 'text/plain'});
            response.write('404 Not Found\n');
            response.end();
        }
    });
```

Ejemplo Clientes conectados - Servidor (2)

```
let allClients = new Array();
const io = new Server(httpServer);
io.sockets.on('connection', (client) => {
        const cAddress = client.request.socket.remoteAddress;
        const cPort = client.request.socket.remotePort;
        allClients.push({address:cAddress, port:cPort});
        console.log(`Nueva conexión de ${cAddress}:${cPort}`);

        io.sockets.emit('all-connections', allClients);
        client.on('output-evt', (data) => {
            client.emit('output-evt', 'Hola Cliente!');
        });

        client.on('disconnect', () => {
            console.log(`El usuario ${cAddress}:${cPort} se va a desconectar`);
            const index = allClients.findIndex(cli => cli.address == cAddress && cli.port == cPort);
            if (index != -1) {
                allClients.splice(index, 1);
                io.sockets.emit('all-connections', allClients);
            } else {
                console.log('¡No se ha encontrado al usuario!')
            }
            console.log(`El usuario ${cAddress}:${cPort} se ha desconectado`);
        });
    }
);
httpServer.listen(8080);
console.log('Servicio Socket.io iniciado');
```

Ejemplo Clientes conectados - Cliente web (1)

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="uft-8">
        <title>Connections</title>
    </head>
    <body>
        <span id="mensaje_servicio"></span>
        <div id="lista_usuarios"></div>
    </body>
    <script src="/socket.io/socket.io.js"></script>
    <script type="text/javascript">
        function mostrar_mensaje(msg) {
            const span_msg = document.getElementById('mensaje_servicio');
            span_msg.textContent = msg;
        }

        function actualizarLista(usuarios) {
            const listCont = document.getElementById('lista_usuarios');
            while(listCont.firstChild && listCont.removeChild(listCont.firstChild));
            const listElement = document.createElement('ul');
            listCont.appendChild(listElement);
            const num = usuarios.length;
            for(var i=0; i<num; i++) {
                const listItem = document.createElement('li');
                listItem.textContent = usuarios[i].address + ':' + usuarios[i].port;
                listElement.appendChild(listItem);
            }
        }
```

Ejemplo Clientes conectados - Cliente web (2)

```
        const serviceURL = document.URL;
        const socket = io(serviceURL);
        socket.on('connect', () => {
            socket.emit('output-evt', 'Hola Servicio!');
        });
        socket.on('output-evt', (data) => {
            mostrar_mensaje('Mensaje de servicio: ' + data);
        });
        socket.on('all-connections', (data) => {
            actualizarLista(data);
        });
        socket.on('disconnect', () => {
            mostrar_mensaje('El servicio ha dejado de funcionar!!');
        });
    </script>
</html>
```

Ejemplo Clientes conectados

- El cliente web se limita a mostrar una lista de usuarios conectados
- La lista se actualiza en tiempo real gracias a las suscripciones mantenidas con el servidor
- El servidor notifica a sus suscriptores de la conexión y desconexión de usuarios a través del evento 'all-connections'

Sección 3 | Introducción a MongoDB

Conceptos generales

- MongoDB es un sistema de gestión de bases de datos de tipo NoSQL
- Utiliza colecciones de entradas de JSON en lugar de tablas relacionales
- Cada entrada puede tener un conjunto de claves y valores arbitrario
- Se pueden realizar “consultas” similares a las utilizadas en SQL

Órdenes y consultas básicas

- Para acceder al cliente en línea: **$ mongosh**
- Consulta para listar las bbdd: **$ show dbs**
- Sentencia para acceder a una bd: **$ use nombreBD**
- Consulta para listar las colecciones de una bd: **$ show collections**
- Consulta para listar los registros: **$ db.nombreColeccion.find()**

Ejemplo MongoDB + Socket.io - Servidor (1)

```
import http          from 'node:http';
import {join}        from 'node:path';
import {readFile}    from 'node:fs';
import {Server}      from 'socket.io';
import {MongoClient} from 'mongodb';

const httpServer = http
    .createServer((request, response) => {
        let {url} = request;
        if(url == '/') {
            url = '/mongo-test.html';
            const filename = join(process.cwd(), url);
            readFile(filename, (err, data) => {
                if(!err) {
                    response.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
                    response.write(data);
                } else {
                    response.writeHead(500, {"Content-Type": "text/plain"});
                    response.write(`Error en la lectura del fichero: ${url}`);
                }
                response.end();
            });
        } else {
            console.log('Peticion invalida: ' + url);
            response.writeHead(404, {'Content-Type': 'text/plain'});
            response.write('404 Not Found\n');
            response.end();
        }
    });
```

Ejemplo MongoDB + Socket.io - Servidor (2)

```
MongoClient.connect("mongodb://localhost:27017/").then((db) => {
    const dbo = db.db("baseDatosTest");
    const collection = dbo.collection("test");

    const io = new Server(httpServer);
    io.sockets.on('connection', (client) => {
        client.emit('my-address', {host:client.request.socket.remoteAddress, port:client.request.socket.remotePort});
        client.on('poner', (data) => {
            collection.insertOne(data, {safe:true}).then((result) => {});
        });
        client.on('obtener', (data) => {
            collection.find(data).toArray().then((results) => {
                client.emit('obtener', results);
            });
        });
    });
    httpServer.listen(8080);
}).catch((err) => {console.error(err);});

console.log('Servicio MongoDB iniciado');
```

Ejemplo MongoDB + Socket.io - Cliente web (1)

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="uft-8">
        <title>MongoDB Test</title>
    </head>
    <body>
        <div id="resultados"></div>
    </body>
    <script src="/socket.io/socket.io.js"></script>
    <script type="text/javascript">
        function actualizarLista(usuarios) {
            const listCont = document.getElementById('resultados');
            while(listCont.firstChild && listCont.removeChild(listCont.firstChild));

            const listElement = document.createElement('ul');
            listCont.appendChild(listElement);

            const num = usuarios.length;
            for(var i=0; i<num; i++) {
                const listItem = document.createElement('li');
                listItem.textContent = JSON.stringify(usuarios[i]);
                listElement.appendChild(listItem);
            }
        }
```

Ejemplo MongoDB + Socket.io - Servidor (2)

```
        const serviceURL = document.URL;
        const socket = io(serviceURL);

        socket.on('my-address', (data) => {
            var d = new Date();
            socket.emit('poner', {host:data.host, port:data.port, time:d});
            socket.emit('obtener', {host:data.host});
        });

        socket.on('obtener', (data) => {actualizarLista(data);});
        socket.on('disconnect', ()  => {actualizarLista({});});

    </script>
</html>
```

Ejemplo MongoDB + Socket.io

- El cliente web muestra una lista que se va actualizando conforme se conectan nuevos usuarios al sistema
- A diferencia del ejemplo anterior, la lista de usuarios conectados se almacena en una colección de una base de datos NoSQL
- La conexión de un cliente crea una nueva entrada en la colección y se recupera la lista completa mediante una operación de consulta.