NodeJS. Características y principales módulos

TSR 2020-2021, Juansa Sendra, Grupo B

NodeJS.- ¿Qué es?

Entorno orientado a la creación de aplicaciones distribuidas. Ejemplos

- Aplicaciones escalables no críticas parte servidor (toleran retardos ocasionales)
- Aplicaciones con interfaces ligeras REST/JSON o interacción AJAX
- Mensajería instantánea, juegos multijugador, ..
- Datos para streaming

NodeJS.- Define

- Interfaces (módulos) http://nodejs.org/api
  - Estudiamos **events**, **stream**, **net**, **http**
- E/S asincrónica (dirigida por eventos)
  - Los métodos de los módulos permiten interacción asincrónica
  - El método retorna immediatamente el control, y el resultado se pasa a un callback -> el invocante NO se bloquea
  - Un único hilo de ejecución: NO objectos compartirs, NO secciones críticas
  - Cola de turnos, orden temporal (contextos de funciones preparadas para la ejecución)
  - El soporte en ejecución extrae y ejecuta en orden de cola
- Intérprete JavaScript
- Gestor de paquetes (npm)
- ...

NodeJS.- Módulo events

- Permite crear emisores de eventos (clase EventEmitter)
  - `emisor.emit(event, arg, arg, ..) // >=0 args`
- Podemos registrar acciones (>=0) asociadas a cada evento
  - `emisor.on(event, accion)`
  - `emisor.once(event, accion)`

```
var ev = require('events')
var emisor = new ev.EventEmitter
emisor.on  ('event', () => {console.log('accion')}) // registra accion
setInterval(() => {emisor.emit('event')}, 1000)    // genera evento
```

NodeJS.- Módulo stream (canales de datos)

- Readable.- únicamente lectura. Eventos: readable,data,end,close,error
  - Ex. process.stdin, ficheros, peticiones HTTP en servidor, ...
- Writable.- únicamente escriptura. Eventos: drain,finish,pipe,unpipe.
  - Ex. process.stdout, process.stderr, ficheros, respuestas HTTP en servidor, ...
- Duplex.- Readable + Writable
  - Ex. sockets TCP, fitxers, ...
- Transform.- Duplex + las escrituras dependen de la información leída

```
console.log("Escribe radio circulo: ")
process.stdin.resume() // inicializa teclado
process.stdin.setEncoding('utf8')
process.stdin.on('data', function(s) {  // Intro
  let r = parseFloat(s.slice(0,-1))
  console.log(`Radi ${r} circumferència ${2*r*Math.PI}`)
  console.log("Escribe radio circulo: ")
})
process.stdin.on('close',function() {console.log("Adios")}) // Ctrl-D
```

NodeJS.- Módulo net (clases para trabajar con sockets TCP)

- net.Server -> servidor TCP
  - `net.createServer([options,][connectionListener])`
    - `connectionListener` recibe como argumento un Socket con conexión establecida
  - eventos listening, connection, close, error
- net.Socket -> socket TCP. Implanta Duplex Stream
  - `new net.Socket()`
  - `net.connect(options[,listener])`, o `net.connect(port[,host][,listener])`
  - eventos connect, data, end, timeout, drain, error, close

NodeJS.- Módulo net (ejemplo client/server, misma máquina)

client

```
const net=require('net')
let client=net.connect({port:9000}, () => {
   console.log('client connected')
   client.write('world!\r\n')
})
client.on('data', (data) => {console.log(data.toString()); client.end()})
client.on('end', ()=>{console.log('Bye')})
```

server

```
const net=require('net')
let server=net.createServer(connectListener)
function connectListener(c) {
   console.log('server connected')
   c.on('end', () => {console.log('server disconnected')})
   c.write('Hello\r\n') // send to client
   c.pipe(c) // write to socket c what reads from c
}
server.listen(9000, () => {console.log('server bound')})
```

NodeJS.- Módulo net. Client/Server (misma màquina). Client

```
const net=require('net')
let args = process.argv.slice(2) // args en linia comandament
if (args.length != 2) {
   console.log('Use: node cs2.js funcio argEntrada')
   process.exit()
}
let fun = args[0], num = Math.abs(parseInt(args[1]))
let client=net.connect({port:9000}, () => {
   console.log('client connected')
   client.write(JSON.stringify({"fun":fun, "num":num}))
})
client.on('data',  (data) => {console.log(data.toString()); client.end()})
client.on('end',   () => {console.log('Bye')})
client.on('error', () => {console.log('connection error')})
```

NodeJS.- Módulo net. Client/Server (misma màquina). Server

```
const net=require('net')
function fact(n) {return n<=1 ? 1: n*fact(n-1)}
function fibo(n) {return n<=2 ? 1: fibo(n-1)+fibo(n-2)}
function connectListener(c) {
   c.on('end',   () => {console.log('server disconnected')})
   c.on('error', () => {console.log('connect error')})
   c.on('data',  calcula)
}
function calcula(c) {
   return function(data) {
     let p = JSON.parse(data), q = NaN
     if (typeof(p.num) == 'number')
       switch (p.fun) {
         case 'fibo': q = fibo(p.num); break
         case 'fact': q = fact(p.num)
       }
     c.write(`${p.fun}(${p.num}) = ${q}`) // response
   }
}
let server=net.createServer(connectListener)
server.listen(9000, () => {console.log('server bound')})
```

NodeJS.- Módulo http

Clases para implantar servidores web

- http.Server -> EventEmitter que modela al servidor web
- http.ClientRequest -> modela petición al servidor. És un Stream Writable i un EventEmitter. Eventos: response, socket, connect, upgrade, continue
- http.ServerResponse -> WritableStream y EventEmitter que modela resposta del servidor. eventos: close
- http.IncomingMessage -> ReadableStream que modela las peticiones que llegan al servidor. eventos: close

NodeJS.- Módulo http: servidor web mínimo

```
const http = require('http')
const args = process.argv.slice(2) // args en linia comandament
const host = args[0]             || '127.0.0.1'
const port = parseInt(args[1]) || 3000
const server = http.createServer(procesa)

server.listen(port, host, () => {
  console.log(`server running at http://${host}:${port}/`)
})

function procesa(req, res) { // res is a ServerResponse
  res.statusCode = 200                          // ok
  res.setHeader('Content-type', 'text/plain') // sets response header
  res.end('Hola a tots')                        // response completed
}
```