



UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Programador Web Avanzado

Clase N° 2: NodeJS

Profesor: Ing. Leandro Rodolfo Gil Carrano

Email: leangilutn@gmail.com

Node JS - Definición

NodeJS, conocido habitualmente también con la palabra "node" a secas, surge en 2009 como respuesta a algunas necesidades encontradas a la hora de desarrollar sitios web, específicamente el caso de la concurrencia y la velocidad.

NodeJS es un plataforma súper-rápida, especialmente diseñada para realizar operaciones de entrada / salida (Input / Output o simplemente I/O en inglés) en redes informáticas por medio de distintos protocolos, apegada a la filosofía UNIX. Es además uno de los actores que ha provocado, junto con HTML5, que Javascript gane gran relevancia en los últimos tiempos, pues ha conseguido llevar al lenguaje a nuevas fronteras como es el trabajo del lado del servidor.

Node JS - Definición

Es un *framework* para implementar operaciones de entrada y salida, como decíamos anteriormente. Está basado en eventos, *streams* y construido encima del motor de Javascript V8, que es con el que funciona el Javascript de Google Chrome.

NodeJS se programa del lado del servidor, lo que indica que los procesos para el desarrollo de software en "Node" se realizan de una manera muy diferente que los de Javascript del lado del cliente.

Node JS - Ejemplos

Existen varios ejemplos de sitios y empresas que ya están usando Node en sitios en producción y algunos casos de éxito que son realmente representativos. El más conocido es el de LinkedIn. Al pasar a NodeJS, LinkedIn ha reducido sensiblemente el número de servidores que tenían en funcionamiento para dar servicio a sus usuarios, específicamente de 30 servidores a 3.

NodeJS tiene un *footprint* de memoria menor. Es decir, los procesos de NodeJS ocupan niveles de memoria sensiblemente menores que los de otros lenguajes, por lo que los requisitos de servidor para atender al mismo número de usuarios son menores. Podríamos llegar a tener 1.000 usuarios conectados a la vez y el proceso de NodeJS ocuparía solamente 5 MB de memoria.

Frameworks Basados en Node js

Node JS - Ejemplos

Meteor JS: Un framework Open Source para crear aplicaciones web rápidamente, basado en programación con "Javascript puro" que se ejecuta sobre el motor de Node.JS.

Grunt: Un conjunto de herramientas que te ayudan como desarrollador web Javascript. Minifica archivos, los verifica, los organiza, etc. Todo basado en línea de comandos.

Yeoman: Otra herramienta, esta vez basada en Grunt, que todavía ofrece más utilidades que ayudan a simplificar diversas tareas en la creación de proyectos, basados en muchas otras librerías y frameworks habituales como Bootstrap, BackboneJS

Ventajas

Node JS - Ventajas

Javascript sin restricciones:

Con NodeJS tenemos un "Javascript sin restricciones" ya que todo se ejecuta en el servidor y no tenemos que preocuparnos de si nuestro código será compatible o no con distintos clientes. Todo lo que escribas en Node JS y te funcione en tu servidor, estarás seguro que funcionará bien, sea cual sea el sistema que se conecte, porque toda la ejecución de código del servidor se queda aislada en el servidor.

Node JS - Ventajas

Programación asíncrona

La filosofía detrás de Node.JS es hacer programas que no bloqueen la línea de ejecución de código con respecto a entradas y salidas, de modo que los ciclos de procesamiento se queden disponibles durante la espera. Por eso todas las APIs de NodeJS usan callbacks de manera intensiva para definir las acciones a ejecutar después de cada operación I/O, que se procesan cuando las entradas o salidas se han completado.

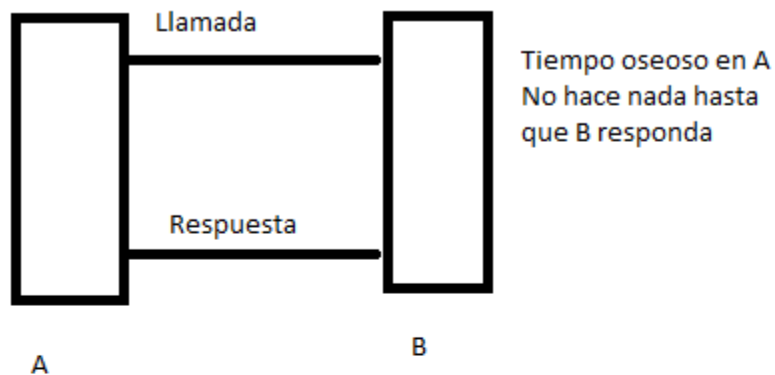


UTN.BA

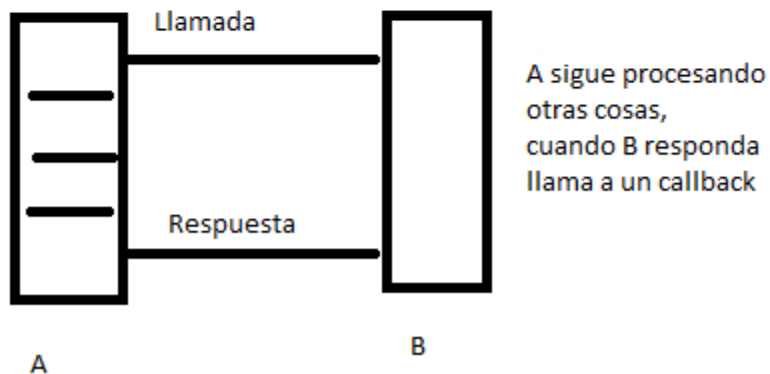
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Node JS - Ventajas

Sincronico



Asincronico



Node JS - Ventajas

Problema del código piramidal

El uso intensivo de callbacks en la programación asíncrona produce el poco deseable efecto de código piramidal, también conocido habitualmente como "código Espagueti" o "callback hell". Al utilizarse los callbacks, se meten unas funciones dentro de otras y se va entrando en niveles de profundidad que hacen un código menos sencillo de entender visualmente y, por tanto de mantener durante la vida de las aplicaciones.

La solución es hacer un esfuerzo adicional por estructurar nuestro código. Básicamente se trata de modularizar el código, escribiendo cada función aparte e indicando solamente su nombre cuando se define el callback. Podrías incluso definir las funciones en archivos aparte y requiriéndolas con `require("../ruta/al/archivo")` en el código de tu aplicación.

Node JS - Ventajas

Programación orientada a eventos

Conocemos la programación orientada a eventos porque la hemos utilizado en Javascript para escribir aplicaciones del lado del cliente. Estamos acostumbrados al sistema, que en NodeJS es algo distinto, aunque sigue el mismo concepto.

En Javascript del lado del cliente tenemos objetos como "window" o "document" pero en Node.JS no existen, pues estamos en el lado del servidor.

Eventos que podremos captar en el servidor serán diferentes, como "uncaughtError", que se produce cuando se encuentra un error por el cual un proceso ya no pueda continuar. El evento "data" es cuando vienen datos por un stream. El evento "request" sobre un servidor también se puede detectar y ejecutar cosas cuando se produzca ese evento.

Proceso de Ejecución

Node JS – Proceso de ejecución

Single thread (hilo único)

Una de las características de NodeJS es su naturaleza "Single Thread". Cuando pones en marcha un programa escrito en NodeJS se dispone de un único hilo de ejecución.

Esto, en contraposición con otros lenguajes de programación como Java, PHP o Ruby, donde cada solicitud se atiende en un nuevo proceso, tiene sus ventajas. Una de ellas es que permite atender mayor demanda con menos recursos, uno de los puntos a favor de NodeJS.

Node JS – Proceso de ejecución

Para conseguir que la programación Single Thread de NodeJS pueda producir resultados satisfactorios debemos entender su característica no bloqueante. En resumen, todas las operaciones que NodeJS no puede realizar al instante, liberan el proceso se libera para atender otras solicitudes. Pero como hemos dicho en este artículo nos proponemos aterrizar estas ideas en algo de código que nos permita ir aprendiendo cosas nuevas de NodeJS.

Node JS – Proceso de ejecución

Objeto process

El objeto process es una variable global disponible en NodeJS que nos ofrece diversas informaciones y utilidades acerca del proceso que está ejecutando un script Node. Contiene diversos métodos, eventos y propiedades que nos sirven no solo para obtener datos del proceso actual, sino también para controlarlo. Puede encontrar toda la documentación de este objeto en:

<https://nodejs.org/docs/latest/api/process.html>

Node JS – Proceso de ejecución

Prueba realizar lo siguiente:

```
console.log('id del proceso: ', process.pid);  
console.log('título del proceso: ', process.title);  
console.log('versión de node: ', process.version);  
console.log('sistema operativo: ', process.platform);
```

INSTALACIÓN



UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Node JS - Instalación

- Ingresar a: <https://nodejs.org/es/>
- Descargar la versión v6.9.2

característica de Chrome. Node.js es un motor
orientado a eventos, que lo hace liviano y ef
Node.js, npm, es el ecosistema mas grande de

Descargar para W

v6.9.2 LTS

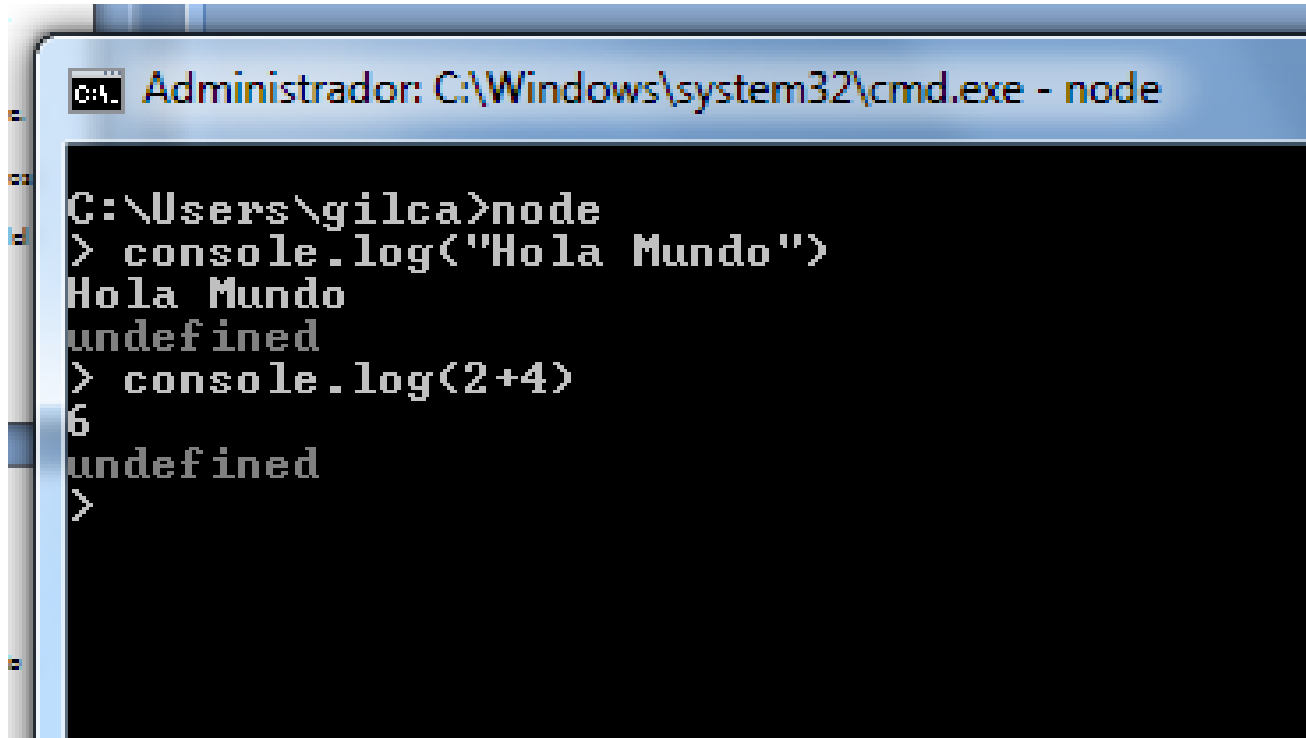
Recomendado para la mayoría

[Otras Descargas](#) | [Cambios](#) | [Documentación del API](#) | [Otr](#)

Ó revise la [Agenc](#)

Node JS - Instalación

- Una vez instalado debemos ejecutar la consola (Ejecutar -> cmd)
- Dentro de la consola ejecutamos el comando **node**



```
Administrador: C:\Windows\system32\cmd.exe - node

C:\Users\gilca>node
> console.log("Hola Mundo")
Hola Mundo
undefined
> console.log(2+4)
6
undefined
>
```

HOLA MUNDO

Node JS – Hola Mundo

Crear un nuevo documento llamado **hola_mundo.js**

Editar ese archivo e incluir un **console.log("hola mundo");**

Abrir la consola ubicarnos en el directorio en el cual esta ese archivo (utilizando línea de comandos, con el comando cd) y ejecutar **node hola_mundo.js**

Si todo está bien, deberías haber visto **hola mundo** en tu consola.

```
G:\sites\node\unidad 2>node hola_mundo.js  
Hola Mundo
```

Node JS – Ejercicio

Define un array en javascript que tenga los siguientes nombres de alumnos:

- Leandro
- Sebastián
- Diego
- Miguel

Luego muestra esos nombres por consola (debes realizar una iteración del array)

Módulos

Node JS – Módulos

En NodeJS el código se organiza por medio de módulos. Son como los paquetes o librerías de otros lenguajes como Java. Por su parte, NPM es el nombre del gestor de paquetes (package manager) que usamos en Node JS.

Node JS – Incluir Módulos

Javascript nativo no da soporte a los módulos. Esto es algo que se ha agregado en NodeJS y se realiza con la **sentencia require()**, que está inspirada en la variante propuesta por CommonJS.

La instrucción require() recibe como parámetro el nombre del paquete que queremos incluir e inicia una búsqueda en el sistema de archivos, en la carpeta "node_modules" y sus hijos, que contienen todos los módulos que podrían ser requeridos.

```
1  
2 var http = require("http");
```

Programación Asíncrona

Node JS – Programación Asíncrona

Imaginemos que un programa tiene un fragmento de código que tarda cinco segundos en resolverse.

En la mayoría de los lenguajes de programación precedentes, durante todo ese tiempo el hilo de ejecución se encuentra ocupado, esperando a que pasen esos cinco segundos, o los que sea, antes de continuar con las siguientes instrucciones.

En la programación los recursos se quedan disponibles para hacer otras cosas durante el tiempo de espera.

Node JS – Programación Asíncrona

La filosofía detrás de Node.JS es hacer programas que no bloqueen la línea de ejecución de código con respecto a entradas y salidas, de modo que los ciclos de procesamiento se queden disponibles durante la espera. Por eso todas las APIs de NodeJS usan callbacks de manera intensiva para definir las acciones a ejecutar después de cada operación I/O, que se procesan cuando las entradas o salidas se han completado.

Ver **ejemplo_2.js**

NPM

Node JS – NPM

Es un comando que funciona desde la línea de comandos de NodeJS. Por tanto lo tenemos que invocar con **npm seguido de la operación que queramos realizar.**

```
npm install async
```

Esto instalará el paquete async dentro de mi proyecto. Lo instalará dentro de la carpeta "node_modules" y a partir de ese momento estará disponible en mi proyecto y podré incluirlo por medio de "require":

```
require("async");
```

HTTP

Node JS – Modulo HTTP

Este es el módulo que nos sirve para trabajar con el protocolo HTTP, que es el que se utiliza en Internet para transferir datos en la Web. Nos servirá para crear un servidor HTTP que acepte solicitudes desde un cliente web

```
1 var http = require("http");  
?
```

Node JS – Modulo HTTP

```
var server = http.createServer(function (peticion, respuesta) {  
    respuesta.end("Hola Mundo");  
});
```

La función callback que enviamos a **createServer()** recibe dos parámetros que son la petición y la respuesta.

La petición por ahora no la usamos, pero contiene datos de la petición realizada.

La respuesta la usaremos para enviarle datos al cliente que hizo la petición. De modo que "**respuesta.end()**" sirve para terminar la petición y enviar los datos al cliente.

Node JS – Modulo HTTP

```
7 server.listen(3000, function(){  
3     console.log("tu servidor está listo en " + this.address().port);  
9 });|
```

Con esto le decimos al servidor que escuche en el puerto 3000, aunque podríamos haber puesto cualquier otro puerto que nos hubiera gustado.

Además "**listen()**" recibe también una función callback que realmente no sería necesaria, pero que nos sirve para hacer cosas cuando el servidor se haya iniciado y esté listo.

Simplemente, en esa función callback indico que estoy listo y escuchando en el puerto configurado.

Node JS – Poner ejecución el modulo HTTP

Nos vamos desde la línea de comandos a la carpeta donde hemos guardado el archivo servidor.js y ejecutamos el comando "node" seguido del nombre del archivo que pretendemos ejecutar:

node ejemplo_4.js

Entonces en consola de comandos nos debe aparecer el mensaje que informa que nuestro servidor está escuchando en el puerto 3000.

El modo de comprobar si realmente el servidor está escuchando a solicitudes de clientes en dicho puerto es acceder con un navegador. Dejamos activa esa ventana de línea de comandos y abrimos el navegador. Accedemos a:

<http://localhost:3000>

Lectura De Archivos

Node JS – Lectura de archivos

Ver **ejemplo_7.js**

```
let fs = require('fs');
fs.readFile('files/ejemplo_7.txt', 'utf-8', function (err, data) {
  if(err) {
    console.log('error: ', err);
  } else {
    console.log(data);
  }
});
```

Node JS – readFile

```
fs.readFile(archivo [, options], callback)
```

primer parámetro le indicamos el nombre del archivo que deseamos leer. Generalmente será una cadena de texto que contenga el nombre del archivo y la ruta que estamos accediendo, pero también podrá ser un buffer.

El segundo parámetro, el opcional, puede ser tanto un objeto como una cadena. Generalmente le indicará una cadena que contendrá el juego de caracteres en el que el archivo está codificado, por ejemplo 'utf-8'.

El tercer parámetro es la función callback, que se ejecutará en el momento que el archivo está leído y se encuentra disponible para hacer alguna cosa.

Ejercicio

Desarrolla un servidor en node que al ejecutar una petición desde el navegador web, lea el contenido de un archivo **unidad_2.txt** y muestre el mismo por pantalla.