



Clase Introductoria a Node.js

Bienvenidos a la clase de introducción a Node.js.

Hoy exploraremos el ecosistema de Node.js, incluyendo su funcionamiento, ventajas, desventajas y una explicación detallada de sus módulos principales.

Nuestro objetivo será comprender a fondo Node.js y su capacidad para desarrollar aplicaciones eficientes y escalables.



Índice de la Clase

- 1 ¿Qué es Node.js?
 - 2 Características principales
 - 3 Ventajas y desventajas
 - 4 Extensión `.mjs` : ¿Por qué es mejor?
 - 5 Módulos principales en Node.js (Teoría y Comparación)
 - `fs` (File System)
 - `path` (Rutas de archivos)
 - `http` (Servidores web)
 - `os` (Sistema operativo)
 - `events` (Manejo de eventos)
 - 6 Ejemplo práctico: Servidor HTTP básico
 - 7 Resumen y preguntas
-

1 ¿Qué es Node.js?

Node.js es un **entorno de ejecución de JavaScript**, construido sobre el motor **V8** de Google Chrome. Permite ejecutar código JavaScript **fuera del navegador**, haciendo posible el desarrollo de servidores, APIs, aplicaciones en tiempo real y mucho más.



Diferencia clave con JavaScript en el navegador:

- En el **navegador**, JavaScript se ejecuta en un entorno restringido con acceso limitado al sistema.
- En **Node.js**, el código puede acceder a archivos, bases de datos y servidores de red.

♦ Modelo de ejecución de Node.js



Node.js es **asíncrono y basado en eventos**, lo que lo hace extremadamente eficiente para tareas de I/O (entrada/salida), como el manejo de archivos o conexiones de red.

- ✓ Usa **un solo hilo** (`single-threaded`), pero aprovecha un **modelo de I/O no bloqueante** para gestionar múltiples solicitudes en paralelo.
- ✓ Se basa en el **Event Loop**, que permite ejecutar código sin interrupciones mientras espera eventos externos.
- ✓ Utiliza **callbacks, promesas y async/await** para manejar la asincronía.

● **Limitación:** Node.js no es ideal para aplicaciones con **cargas pesadas en CPU** (como procesamiento de imágenes o Machine Learning), ya que es de un solo hilo.

2 Características principales de Node.js

- ✓ **I/O No Bloqueante:** Maneja múltiples operaciones en paralelo sin detener el flujo de ejecución.
- ✓ **Basado en el motor V8:** Ejecuta JavaScript de forma eficiente y rápida.
- ✓ **NPM (Node Package Manager):** Un ecosistema con más de un millón de paquetes disponibles.
- ✓ **Multiplataforma:** Funciona en Windows, Mac y Linux sin problemas.
- ✓ **Compatible con TypeScript:** Mejora la robustez del código con tipado estático.
- ✓ **Adecuado para microservicios y APIs REST:** Perfecto para arquitecturas escalables.

3 Ventajas y Desventajas de Node.js

✓ Ventajas

- **Alto rendimiento** 🚀 gracias a su ejecución con el motor V8.
- **Escalabilidad** 🔄 : Puede manejar miles de solicitudes concurrentes con pocos recursos.
- **Gran ecosistema de paquetes** 📦 con `npm`.
- **Desarrollo full-stack** 🌐 con JavaScript en frontend y backend.
- **Ideal para aplicaciones en tiempo real** 🗣️ (chat, sockets, juegos en línea).

❌ Desventajas

- **Monohilo** 🔥 : No es óptimo para tareas computacionales intensivas.
- **Callback Hell** 🕸 : Un mal diseño puede generar código difícil de mantener.
- **Mayor consumo de memoria** 📦 en comparación con entornos como Go o Rust.

4 ¿Por qué usar archivos `.mjs`?

Node.js soporta dos sistemas de módulos:

🔗 Comparación de CommonJS y ES Modules:

Característica	CommonJS (.js)	ES Modules (.mjs)
Importación	<code>require('fs')</code>	<code>import fs from 'fs'</code>
Exportación	<code>module.exports = {}</code>	<code>export default {}</code>
Carga	Síncrona	Asíncrona
Navegadores	❌ No compatible	✅ Compatible

💡 Razones para preferir `.mjs` en Node.js:

- ✅ Es el estándar oficial de ECMAScript.
- ✅ Compatible con navegadores sin herramientas adicionales.
- ✅ Permite la importación dinámica con `import()`.
- ✅ Evita problemas de compatibilidad en proyectos modernos que mezclan frontend y backend.
- ✅ Facilita la interoperabilidad con herramientas modernas como Webpack, Babel y TypeScript.
- ✅ Mejora la gestión de dependencias en proyectos modulares y reutilizables.

💡 Consideraciones al usar `.mjs`:

- Si usas `"type": "module"` en `package.json`, puedes usar `.js` en lugar de `.mjs`.
- Al usar `import`, no puedes acceder a variables globales de CommonJS como `__dirname` sin trabajo adicional.
- La importación de módulos nativos de Node.js (`fs`, `http`, etc.) debe realizarse con `import fs from 'fs'` en lugar de `require`.
- No es compatible con algunos módulos más antiguos que dependen de `require`.

🔴 Cuándo usar CommonJS (`.js`) en lugar de ES Modules (`.mjs`):

- Cuando trabajas con paquetes antiguos de npm que aún usan `require`.

- Si necesitas compatibilidad con versiones antiguas de Node.js.
- En proyectos donde el ecosistema aún no ha migrado completamente a ES Modules.

Conclusión: Si trabajas en un proyecto moderno o deseas compatibilidad con el ecosistema actual de JavaScript, usa **ES Modules** (`.mjs`). Si mantienes código heredado, puede que necesites seguir con **CommonJS** (`.js`).

5 Módulos principales en Node.js

`fs` (File System) - Manejo de archivos

 **Objetivo:** Permite leer, escribir, modificar y eliminar archivos del sistema.

Ventajas:

- Lectura/escritura asíncrona para evitar bloqueos.
- Soporta Streams para manejar archivos grandes.

Comparación con `path`:

- `fs` manipula archivos, mientras que `path` solo maneja rutas de archivos.

Ejemplo:

```
import fs from 'fs';
fs.writeFileSync('example.txt', 'Hola, Node.js!');
console.log(fs.readFileSync('example.txt', 'utf8'));
```

`path` - Manejo de rutas de archivos

 **Objetivo:** Permite resolver y construir rutas de archivos de forma portable.

Ventajas:

- Evita problemas con rutas en Windows y Linux.
- Permite obtener nombres, extensiones y directorios.

Ejemplo:

```
import path from 'path';
console.log(path.join(__dirname, 'folder', 'file.txt'));
```

http - Creación de servidores web

 **Objetivo:** Permite construir servidores HTTP sin frameworks adicionales.


 **Ventajas:**

- Más ligero que Express.js para servidores simples.
- Permite manejar solicitudes HTTP y WebSockets.

Ejemplo:

```
import http from 'http';
const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('¡Hola desde Node.js!');
});
server.listen(3000, () => console.log('Servidor en http://localhost:3000'));
```


os - Información del sistema operativo

 **Objetivo:** Proporciona información sobre el sistema donde se ejecuta Node.js.

Ejemplo:

```
import os from 'os';
console.log('Sistema Operativo:', os.platform());
console.log('Memoria Libre:', os.freemem());
```

events - Manejo de eventos


 **Objetivo:** Implementa un sistema de eventos personalizado en Node.js.

Ejemplo:

```
import { EventEmitter } from 'events';
const emitter = new EventEmitter();
emitter.on('mensaje', (data) => console.log('Mensaje recibido:', data));
emitter.emit('mensaje', '¡Hola, Node.js!');
```

7 Resumen y preguntas

- ✓ Node.js es eficiente y escalable gracias a su modelo asíncrono.
- ✓ Soporta módulos nativos que facilitan el desarrollo.
- ✓ `.mjs` es el estándar moderno de importaciones en JavaScript.

 **¡Ahora es tu turno!** Practica y experimenta con los módulos de Node.js. ¿Tienes dudas? ¡Pregunta!

