07 NodeJS: JS del navegador VS JS del servidor

Ahora que hemos interactuado con el REPL de Node.js, es fundamental entender las diferencias clave entre el entorno de ejecución de JavaScript al que están acostumbrados (el navegador) y este nuevo entorno (Node.js). Aunque **el lenguaje JavaScript es el mismo** (la sintaxis, los tipos de datos, las estructuras de control como `if`, `for`, `while`, las funciones, los objetos, etc.), el **contexto de ejecución** y las **capacidades disponibles** varían significativamente.

La diferencia principal radica en el *propósito* de cada entorno:

- **Navegador:** Su propósito es renderizar páginas web e interactuar con el usuario a través de una interfaz gráfica. Por lo tanto, proporciona APIs para manipular la estructura visual (DOM), gestionar la ventana del navegador (BOM) y responder a acciones del usuario. Por razones de seguridad, su acceso al sistema del usuario es muy restringido.
- **Node.js:** Su propósito es ejecutar JavaScript en un servidor o directamente en el sistema operativo para tareas de backend, scripting, herramientas, etc. No tiene una interfaz gráfica intrínseca ni una página web que mostrar. En cambio, necesita interactuar directamente con el sistema operativo, el sistema de archivos, las redes, etc.

Veamos las diferencias más importantes:

1. **Ausencia del DOM y el BOM:**
    - **Navegador:** Tienen acceso a objetos globales cruciales como `window` (el objeto global principal que representa la ventana del navegador) y `document` (que representa la estructura HTML/XML de la página, es decir, el DOM - Document Object Model). A través de ellos, pueden manipular elementos HTML, escuchar eventos de UI (`click`, `mouseover`), gestionar el historial de navegación (`history`), obtener información de la ubicación (`location`), etc.
    - **Node.js: No existe el objeto `window` ni el objeto `document`**. Al ejecutar código en Node.js (sea en el REPL o en un script), no hay una página web asociada ni una ventana de navegador. Intentar acceder a `window` o `document` resultará en un error (`ReferenceError`). Por consiguiente, todas las APIs relacionadas con el DOM (como `getElementById`, `createElement`, `querySelector`) y el BOM (`alert`, `confirm`, `navigator`, `location`) no están disponibles.
    - **Relevancia en el REPL:** Esto explica por qué, en el REPL que acabamos de usar, no podemos manipular elementos HTML. El REPL es una interfaz de línea de comandos, no una página web.
2. **El Objeto Global:**
    - **Navegador:** El objeto global es `window`. Las variables declaradas con `var` en el ámbito global (fuera de funciones) se convierten en propiedades de `window`. También se puede acceder a funciones globales como `setTimeout` o `parseInt` directamente o a través de `window.setTimeout`, etc.
    - **Node.js:** El objeto global principal se llama `global`. Contiene funciones y objetos incorporados disponibles en cualquier parte de una aplicación Node.js (como `setTimeout`, `setInterval`, `clearTimeout`, `clearInterval`, `console`, y los propios `global` y `process`). Sin embargo, a diferencia del navegador, las variables declaradas con `var`, `let` o `const` en el ámbito más externo de un archivo de módulo Node.js **no** se añaden automáticamente al objeto `global`.

Están encapsuladas dentro del ámbito de ese módulo. En el REPL, el comportamiento es un poco más parecido al de la consola del navegador por conveniencia, y las variables declaradas pueden parecer globales.

- Pueden inspeccionar el objeto `global` directamente en el REPL:

```
> global
<ref *1> Object [global] {
  global: [Circular *1],
  clearInterval: [Function: clearInterval],
  clearTimeout: [Function: clearTimeout],
  setInterval: [Function: setInterval],
  setTimeout: [Function: setTimeout] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  },
  queueMicrotask: [Function: queueMicrotask],
  clearImmediate: [Function: clearImmediate],
  setImmediate: [Function: setImmediate] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  },
  // ... y muchas otras propiedades
}
```

3. **Acceso al Sistema y Módulos Incorporados:**
   - **Navegador:** Tiene un acceso muy limitado al sistema de archivos local (por seguridad). Las capacidades de red se limitan principalmente a realizar peticiones HTTP/HTTPS (`fetch`, `XMLHttpRequest`) y WebSockets.
   - **Node.js:** Proporciona un conjunto rico de **módulos incorporados** que otorgan acceso directo a funcionalidades del sistema operativo:
     - `fs` (File System): Para leer y escribir archivos.
     - `http`, `https`: Para crear servidores y clientes HTTP/HTTPS.
     - `net`: Para crear servidores y clientes TCP.
     - `os`: Para obtener información sobre el sistema operativo (CPU, memoria, etc.).
     - `path`: Para trabajar con rutas de archivos y directorios.
     - `process`: Un objeto global (también accesible directamente) que proporciona información y control sobre el proceso actual de Node.js.
4. **El Objeto `process`:**
   - **Navegador:** No existe un objeto equivalente directo con el mismo nivel de detalle y control.
   - **Node.js:** El objeto `process` es fundamental. Proporciona información como:
     - `process.argv`: Argumentos pasados al script Node.js desde la línea de comandos.
     - `process.env`: Variables de entorno del sistema.
     - `process.cwd()`: Directorio de trabajo actual.
     - `process.platform`: Plataforma del sistema operativo (`'win32'`, `'linux'`, `'darwin'`).

- `process.exit()`: Para terminar el proceso de Node.js.
- Y mucho más... (Pueden inspeccionarlo en el REPL: `> process`).

5. **Sistema de Módulos:**
   - **Navegador:** Históricamente dependía de cargar scripts globalmente (`<script>`) o usar patrones como IIFE. Ahora, el estándar es **ES Modules (ESM)** con `import` y `export`.
   - **Node.js:** Históricamente utilizó el sistema **CommonJS (CJS)** con `require()` y `module.exports`. Aunque Node.js ahora soporta completamente ES Modules, CommonJS sigue siendo muy prevalente en el ecosistema. Comprender ambos sistemas es importante al trabajar con Node.js.

En resumen:

Característica	JavaScript en Navegador	JavaScript en Node.js
Entorno Principal	Navegador Web	Servidor / Sistema Operativo
Objeto Global	`window`	`global`
DOM/BOM	Disponible (`document`, `location`, etc.)	**No disponible**
Acceso a Archivos	Muy limitado / Sandboxeado	Completo (módulo `fs`)
Red	`fetch`, `XMLHttpRequest`, WebSockets	`http`, `https`, `net`, `dgram`, etc.
Info del Proceso	Limitado	Extenso (objeto `process`)
Sistema de Módulos	ES Modules (estándar actual)	CommonJS (histórico), ES Modules (soportado)
APIs Principales	DOM API, BOM API, Web APIs	Node.js API (`fs`, `http`, `path`, etc.)