Introducción a Node.js

Node.js es un entorno de ejecución JavaScript construido utilizando el motor V8 de Google Chrome. Esto permite ejecutar aplicaciones JavaScript en cualquier sistema que lo admita, incluyendo entornos de servidor donde hasta la aparición de Node.js el lenguaje JavaScript no había encontrado un lugar. Veremos ahora sus características principales y lo que supone en el mundo del desarrollo web.

1. Evolución de JavaScript

Como hemos comentado, Node.js es un entorno que permite ejecutar el lenguaje de programación JavaScript en el lado del servidor. Esta afirmación puede resultar mundana, pero en realidad es algo sorprendente. Si echamos la vista atrás, el lenguaje JavaScript ha pasado por varias etapas o fases de expansión sucesivas:

1. La primera tuvo lugar con el primer apogeo de la web, allá por los años 90. Se comenzaban a desarrollar webs con HTML, y el JavaScript que se empleaba entonces permitía añadir dinamismo a esas páginas, bien validando formularios, abriendo ventanas, o explorando el DOM (estructura de elementos de la página), añadiendo o quitando contenidos del mismo. Era lo que se conocía como HTML dinámico o DHTML.
2. La segunda etapa llegó con la incorporación de las comunicaciones asíncronas, es decir, con AJAX, más o menos a principios del siglo XXI. Se desarrollaron librerías como *Prototype* (primero) o *jQuery* (después) que abrieron todo un mundo nuevo de posibilidades con el lenguaje. Con ellas se podían actualizar fragmentos de la página, llamando desde JavaScript a documentos del servidor, recogiendo la respuesta y pegándola en una zona concreta de la página, sin necesidad de recargarla por completo.
3. Una siguiente etapa, vinculada a la anterior, tuvo lugar con la aparición de distintos frameworks JavaScript para desarrollo de aplicaciones web en el lado del cliente, o *frontend*. Mediante una serie de funcionalidades incorporadas, y de librerías externas, permiten dotar a la aplicación cliente de una estructura muy determinada con unos añadidos que facilitan, entre otras cosas, la compartición de variables entre vistas o páginas, o la generación dinámica de contenido HTML en la propia vista. Hablamos, fundamentalmente, de frameworks como Angular, React o Vue.
4. La última etapa ha llegado con la expansión de JavaScript al lado del servidor. Hasta este momento sólo se utilizaba en la parte cliente, es decir, fundamentalmente en los navegadores, por lo que sólo estábamos utilizando y viendo una versión reducida o restringida del lenguaje. Creíamos que JavaScript sólo servía para validaciones, exploración del contenido HTML de una página o carga de contenidos en zonas concretas. Pero la realidad es que JavaScript puede ser un lenguaje completo, y eso significa que podemos hacer con él cualquier cosa que se puede hacer con otros lenguajes completos, como Java o C#: acceder al sistema de ficheros, conectar con una base de datos, etc.

2. JavaScript en el servidor. El motor V8

Bueno, vayamos asimilando esta nueva situación. Sí, JavaScript ya no es sólo un lenguaje de desarrollo en el cliente, sino que se puede emplear también en el servidor. Pero... ¿cómo? En el caso de Node.js, como hemos comentado, lo que se hace es utilizar de manera externa el mismo motor de ejecución que emplea Google Chrome para compilar y ejecutar JavaScript en el navegador: el motor V8. Dicho motor se encarga de compilar y ejecutar el código JavaScript, transformándolo en código más rápido (código máquina). También se encarga de colocar los elementos necesarios en memoria, eliminar de ella los elementos no utilizados (*garbage collection*), etc.

V8 está escrito en C++, es open-source y de alto rendimiento. Se emplea en el navegador Google Chrome y "variantes" como Chromium (adaptación de Chrome a sistemas Linux), además de en Node.js y otras aplicaciones. Podemos ejecutarlo en sistemas Windows, Mac OS X (10.5 o posteriores) y Linux (con procesadores IA-32, x64, ARM o MIPS).

Además, al estar escrito en C++ y ser de código abierto, podemos extender las opciones del propio JavaScript. Como hemos comentado, inicialmente JavaScript era un lenguaje concebido para su ejecución en un navegador. No podíamos leer un fichero de texto local, por ejemplo. Sin embargo, con Node.js se ha añadido una capa de funcionalidad extra a la base proporcionada por V8, de modo que ya es posible realizar estas tareas, gracias a que con C++ sí podemos acceder a los ficheros locales, o conectar a una base de datos.

2.1. Requisitos para que JavaScript pueda correr en el servidor

¿Qué características tienen lenguajes como PHP, ASP.NET o JSP que no tenía JavaScript hasta la aparición de Node.js, y que les permitían ser lenguajes en entorno servidor? Quizá las principales sean:

- Disponen de mecanismos para acceder al sistema de ficheros, lo que es particularmente útil para leer ficheros de texto, o subir imágenes al servidor, por poner dos ejemplos.
- Disponen de mecanismos para conectar con bases de datos
- Permiten aceptar peticiones de clientes y enviar respuestas a dichas peticiones

Nada de esto era posible en JavaScript hasta la aparición de Node.js. Este nuevo paso en el lenguaje le ha permitido, por tanto, conquistar también el otro lado de la comunicación cliente-servidor para las aplicaciones web.

La **consecuencia** lógica de utilizar Node.js en el desarrollo de servidor es que, teniendo en cuenta que JavaScript es también un lenguaje de desarrollo en el cliente, nos hará falta conocer un único lenguaje para el desarrollo completo de una aplicación web. Antes de que esto fuera posible, era indispensable conocer, al menos, dos lenguajes: JavaScript para la parte de cliente y PHP, JSP, ASP.NET u otro lenguaje para la parte del servidor.

3. Características principales de Node.js

Entre las principales características que ofrece Node.js, podemos destacar las siguientes:

- Node.js ofrece una **API asíncrona**, es decir, que no bloquea el programa principal cuando llamamos a sus métodos esperando una respuesta, y **dirigida por eventos**, lo que permite recoger una respuesta cuando ésta se produce, sin dejar al programa esperando por ella. Comprenderemos mejor estos conceptos más adelante, cuando los pongamos en práctica.
- La ejecución de código es **muy rápida** (recordemos que se apoya en el motor V8 de Google Chrome, implementado en C++).
- Modelo **monohilo** pero muy **escalable**. Se tiene un único hilo atendiendo peticiones de clientes, a diferencia de otros servidores que permiten lanzar hasta N hilos en paralelo. Sin embargo, la API asíncrona y dirigida por eventos permite atender múltiples peticiones por ese único hilo, consumiendo muchos menos recursos que los sistemas multihilo.
- Se elimina la necesidad de ***cross-browser***, es decir, de desarrollar código JavaScript que sea compatible con todos los navegadores, que es a lo que el desarrollo en el cliente nos tiene acostumbrados. En este caso, sólo debemos preocuparnos de que nuestro código JavaScript sea correcto para ejecutarse en el servidor.

4. ¿Quién utiliza Node.js?

Hay varias empresas, algunas de ellas de un peso relevante a nivel internacional, que utilizan Node.js en su desarrollo. Por poner algunos ejemplos representativos, podemos citar a Netflix, PayPal, Uber o la NASA, entre otras. En el caso de España, grandes empresas de desarrollo de software como Everis, Accenture o Indra, también solicitan personal cualificado en Node.js para cubrir puestos de trabajo. Además, en webs de referencia como *InfoJobs*, se suelen encontrar hasta 4 o 5 veces más ofertas de trabajo relacionadas con Node.js que con otros frameworks populares como Laravel o Symfony.