Engineer

JavaScript

Lenguaje que se ejecuta en navegadores y otros entornos

Qué es

JavaScript es un lenguaje de programación de tipado dinámico, normalizado como ECMAScript, que todos los navegadores web pueden ejecutar y que también funciona en servidores y máquinas de compilación mediante entornos como Node.js. Se abrevia JS. Las ediciones anuales del estándar se llaman ES2015, ES2024, etc.; aún se oye la numeración antigua (ES5, ES6), y «vanilla JavaScript», o JavaScript puro, significa el lenguaje sin un framework encima. Aparte del nombre y cierta sintaxis con llaves, tiene poco en común con Java.

Brendan Eich escribió en Netscape la primera versión, que se distribuyó con el navegador Navigator 2.0. Ecma International adoptó la primera edición del estándar, ECMA-262, en junio de 1997, y su comité TC39 publica una edición nueva cada año desde ECMAScript 2016. Una propuesta solo entra en la especificación en la etapa 4 del proceso del comité, que exige implementaciones ya distribuidas, de modo que el texto y los motores rara vez discrepan durante mucho tiempo.

El lenguaje dice muy poco sobre el mundo exterior. ECMA-262 define la sintaxis, los tipos, los objetos, las promesas y los módulos, y deja la entrada y la salida al entorno que lo aloje. En un navegador, ese entorno es la plataforma web: el documento, los eventos, los temporizadores y las peticiones de red, especificados principalmente por el estándar HTML y sus estándares relacionados. Node.js proporciona, en cambio, archivos, sockets y procesos. Esa separación explica una sorpresa habitual: el código puede ser JavaScript perfectamente válido y aun así fallar al pasar de una página a un servidor, porque el lenguaje se ha trasladado y su entorno se ha quedado atrás.

En un navegador, los scripts de una página comparten un único hilo principal con la disposición visual, el pintado y la entrada. Una tarea se ejecuta hasta terminar antes de que el navegador pueda responder al siguiente clic. Las promesas y las funciones `async` mantienen la espera fuera de ese hilo, mientras que el cálculo permanece en él salvo que se traslade a un Web Worker. Los números son valores IEEE 754 de doble precisión, de modo que los enteros solo son exactos hasta 2^53 – 1 y los mayores necesitan `BigInt` .

Utilice el concepto en una decisión

Aplique «¿Qué es JavaScript?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Interfaces que cargan rápido, funcionan en el teléfono y se comportan igual en los navegadores que utilizan sus clientes.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Dirección del desarrollo integral de mapas GIS — cartografía integral, de la base de datos al navegador

Dirección del desarrollo integral de mapas GIS Póngase en contacto →

Por qué importa

La verdadera decisión es cuánto enviar a quienes leen su sitio. JavaScript es el lenguaje que permite que una página responda sin un viaje de ida y vuelta al servidor, y cada kilobyte se descarga, analiza, compila y ejecuta en un dispositivo que usted no controla. Un paquete que parece instantáneo en el portátil de quien lo desarrolla puede mantener ocupado durante segundos un teléfono de gama media, a costa de la batería y la paciencia del lector.

Los scripts también son el punto de mayor exposición de una página. El script de un atacante, si llega a ejecutarse, actúa con la autoridad de la página: puede leer lo que escribe el lector y enviarlo a cualquier destino al que la página pueda enviar datos. Ese es el riesgo que Content Security Policy pretende contener. No distribuir JavaScript ejecutable elimina los scripts propios del sitio en el navegador como superficie de ataque; los datos no confiables siguen necesitando un tratamiento seguro y una política restrictiva. El ecosistema de paquetes añade una segunda exposición. Una compilación típica del frontend incorpora cientos de dependencias transitivas, y cada una es código que se ejecuta con los mismos permisos que el suyo.

La otra cara también tiene peso. Ejecutar un lenguaje a ambos lados de un sistema permite que el navegador y el servidor compartan una regla de validación o una estructura de datos, y algunos productos necesitan muchos

scripts: estado sin conexión, arrastrar y soltar, mapas o colaboración en directo. La pregunta útil se plantea para cada función: ¿debe ocurrir esto en el navegador del lector sin consultar a un servidor? Si no es así, genere el contenido en el servidor o durante la compilación y entregue HTML.

Una decisión relacionada es JavaScript frente a TypeScript. TypeScript añade un comprobador de tipos y un paso de compilación, y se compila de nuevo a JavaScript corriente. En una base de código que varias personas modifican cada semana, el comprobador suele compensar; para una mejora de cien líneas en una página estática, bastan JavaScript y un linter estricto.

Cómo se aplica en la práctica

Este sitio entrega páginas estáticas sin JavaScript ejecutable a sus lectores. Conserva el código de búsqueda, pero lo tiene desactivado; JavaScript sigue ejecutándose en las herramientas de compilación y comprobación. Tres páginas muestran la excepción anterior en el navegador y el trabajo realizado fuera de él.

Una versión anterior en tres idiomas no tenía backend en tiempo de ejecución y entregaba un único script propio. El registro de la compilación con Hugo describe esa excepción acotada: un filtro del portfolio cuyos controles permanecían ocultos hasta que se ejecutaba el script. Sus 108 archivos de plantillas, entre ellos 53 parciales y 18 shortcodes, hacían su trabajo al compilar el sitio. El registro también señala el coste de esa decisión en aquel momento: las funciones interactivas debían resolverse durante la compilación, y se calculó el coste de un índice de búsqueda que quedó aparcado en lugar de publicarse. Esas cifras y esa excepción describen la implementación anterior, no el código que hoy se entrega al lector.

El mismo contenido se publica mediante Gemini y Gopher, dos protocolos sin lugar para un script. Una compilación escribió 777 documentos en cada formato en la etapa de implementación documentada a partir de la misma fuente que las páginas. Otra comparación de compilaciones confirmó que añadir esos formatos de salida no cambió ni un byte de HTML; el resultado no abarca cambios editoriales o visuales posteriores. El sitio sostiene que no hace falta un megabyte de scripts para leer un documento, y un formato que no puede contenerlos es la manera más clara de demostrarlo.

La mayor parte del JavaScript de este sitio está en las herramientas. En la etapa de implementación documentada por el trabajo enlazado, el control de calidad del sitio comprendía treinta y dos scripts y 10.242 líneas de JavaScript, entonces el mayor conjunto de código del repositorio. Durante un tiempo fue el único conjunto de código que no pasaba por un formateador, un linter ni una comprobación de tipos. Incorporarlo a un formateador y un linter empezó por adaptar el formateador al código: el valor predeterminado de cuatro espacios del repositorio habría reescrito 2.173 líneas del mayor comprobador, y un ajuste específico de dos espacios para esos archivos dejó 38 líneas de desviación real. Después, el linter detectó 13 problemas y se corrigieron los 13 sin suprimir ninguno. Un fallo de un comprobador se manifiesta como una comprobación que pasa cuando debería fallar, por lo que el código del propio control recibe el mismo tratamiento que aplica al resto.

Dónde surgen los problemas

Los fallos siguientes se derivan de cómo está construido el lenguaje y cada uno tiene una solución conocida.

La página se muestra y después ignora los toques durante un segundo o más. Una tarea larga, como hidratar un paquete grande, analizar una respuesta JSON extensa u ordenar miles de filas, ocupa el único hilo que atiende la entrada. Envíe menos código para la primera interacción dividiendo los paquetes por ruta y aplazando el resto, reparta el trabajo largo en partes que devuelvan el control al bucle de eventos y traslade el cálculo intensivo a un Web Worker. Mida en un teléfono lento, no en el portátil donde se escribió el código.

Los precios y los identificadores salen ligeramente mal. Todo número es un valor binario de coma flotante, de modo que las fracciones decimales se redondean y los enteros muy grandes se confunden con sus vecinos. Una clave de base de datos de 64 bits enviada como número JSON es la víctima habitual:

```
0.1 + 0.2 === 0.3; // false
Number.MAX_SAFE_INTEGER; // 9007199254740991
9007199254740993 === 9007199254740992; // true
```

Conserve los importes como cantidades enteras de unidades menores, como céntimos, envíe los identificadores grandes como cadenas y utilice `BigInt` cuando la aritmética deba ser exacta.

Una comparación deja pasar el valor equivocado. El operador `==` convierte sus operandos antes de compararlos, por lo que `"" == 0` y `"0" == false` son verdaderos. Utilice `===` en todo el código y deje que la regla `eqeqeq` del linter lo haga cumplir, para que la conversión sea algo que el código solicite expresamente.

El texto de un usuario acaba ejecutándose como código. Asignar texto no confiable a `innerHTML`, construir marcado concatenando cadenas o pasar datos de entrada a `eval` convierte datos en scripts. Escriba texto mediante `textContent`, envíe una Content Security Policy que admita scripts solo mediante nonce o hash y mantenga una lista de dependencias corta y fijada en un archivo de bloqueo, leyendo qué incorpora cada actualización.

El código que funciona en el navegador falla en el servidor. El lenguaje se ha trasladado, pero su entorno no: `window` y `document` solo existen en el navegador, y Node.js lee un archivo como CommonJS o como módulo ES según ajustes que el navegador nunca ve. Mantenga las llamadas específicas de cada entorno en los límites, establezca `"type": "module"` expresamente en `package.json` y ejecute cada parte del código en el entorno donde vaya a funcionar realmente.

Lecturas adicionales

- Especificación del lenguaje ECMAScript — el borrador vivo de ECMA-262, el texto que implementa cada motor, actualizado a medida que las propuestas alcanzan la etapa 4.
- El proceso de TC39 — las etapas que recorre una propuesta antes de incorporarse al lenguaje y lo que exige cada una.
- ECMA-262 en Ecma International — las ediciones anuales publicadas, para cuando un proyecto debe indicar la versión a la que se dirige.
- Estándar HTML: bucles de eventos — cómo organiza el navegador las tareas, las microtareas y el renderizado en torno a los scripts de una página.
- Estándar HTML: el elemento script — scripts clásicos y de módulos y cómo cambian `async` y `defer` el momento de su ejecución.



Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es JavaScript?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/javascript/