JavaScript

Introducción a JavaScript

CertiDevs

Índice de contenidos

1. ¿Qué es JavaScript?

JavaScript (abreviado JS) es un lenguaje de programación de alto nivel y multiparadigma creado para dotar de dinamismo a las páginas web, evitando así una apariencia estática.

Los programas desarrollados en **JavaScript** se llaman scripts y se pueden escribir directamente sobre documentos html o en archivos externos con extensión `.js`, siendo ejecutados por el navegador cuando se carga la página web.

Actualmente JavaScript es soportado por todos los navegadores web, lo que lo convierte en el lenguaje de programación idóneo para ser ejecutado junto a HTML y CSS para crear páginas web dinámicas, permitiendo mejoras en la interfaz de usuario.

JavaScript es un lenguaje de programación multiparadigma en cuya sintaxis podemos encontrar: variables, operadores, funciones, estructuras de datos, estructuras de control condicional, estructuras de control repetitivo…

2. Evolución

2.1. Origen

JavaScript fue creado en mayo de **1995** en 10 días por Brendan Eich. Eich trabajó en Netscape e implementó JavaScript para su navegador web, Netscape Navigator.

La idea principal detrás de la creación de este lenguaje fue sobrepasar la limitación que había en aquella época de que las webs fueran *estáticas* una vez se cargaban en el navegador. JavaScript se convierte en un estándar ECMA en 1997.

La idea era que las principales partes interactivas de la web del lado del cliente se implementaran en Java. Se suponía que JavaScript era un lenguaje de unión para esas partes y también para hacer que HTML fuera un poco más interactivo.

Dada su función de ayudar a Java, JavaScript tenía que parecerse a Java. Eso descartó soluciones existentes como Perl, Python, TCL y otras.

Inicialmente, el nombre de JavaScript cambió varias veces:

1. Su nombre en clave era Mocha.
2. En las versiones beta de Netscape Navigator 2.0 (septiembre de 1995), se llamaba LiveScript.
3. En Netscape Navigator 2.0 beta 3 (diciembre de 1995), obtuvo su nombre final, JavaScript.

2.2. Estandarización

Hay dos estándares para JavaScript:

- ECMA-262 está alojado en Ecma International. Es el estándar primario.
- ISO/IEC 16262 está hospedado por la Organización Internacional de Normalización (ISO) y la

Comisión Electrotécnica Internacional (IEC). Este es un estándar secundario.

El lenguaje descrito por estos estándares se llama **ECMAScript**, no JavaScript. Se eligió un nombre diferente porque Sun (ahora Oracle) tenía una marca comercial para este último nombre.

La palabra “ECMA” en “ECMAScript” proviene de la organización que aloja el estándar principal.

El nombre original de esa organización era ECMA, un acrónimo de Asociación Europea de Fabricantes de Computadoras. Más tarde se cambió a Ecma International (siendo “Ecma” un nombre propio, no un acrónimo) porque las actividades de la organización se habían expandido más allá de Europa.

El acrónimo inicial en mayúsculas explica la ortografía de ECMAScript.

En principio, JavaScript y ECMAScript significan lo mismo. A veces se hace la siguiente distinción:

- El término **JavaScript** se refiere al lenguaje y sus implementaciones.
- El término **ECMAScript** se refiere al estándar y a las versiones del lenguaje.

Por lo tanto, ECMAScript 6 es una versión del lenguaje (su sexta edición).

2.3. Influencias

Cuando se creó JavaScript en 1995, estuvo influenciado por varios lenguajes de programación:

- La sintaxis de JavaScript se basa en gran medida en Java.
- Se centra en herencia prototipada, un sub-paradigma de la programación orientada a objetos (POO).
- Closures y environments fueron tomados de Scheme.
- AWK influyó en las funciones de JavaScript (incluida la palabra clave `function`).
- Los strings, las matrices y las expresiones regulares de JavaScript se inspiran en Perl.
- Gestión de eventos inspirada en HyperTalk a través de onclick en navegadores web.

Con ECMAScript 6, llegaron nuevas influencias a JavaScript:

- Se incorporan los generadores, concepto prestado de Python.
- La sintaxis de las funciones de flecha (`=>`) provino de CoffeeScript.
- C++ aportó la palabra clave `const`.
- La desestructuración de métodos (Destructuring) se inspiró en Lisp.
- Los Literales plantilla provienen del lenguaje E (donde se denominan quasi literals).

2.4. Versiones

JavaScript fue desarrollado en 1995 por Netscape.

En 1996 ECMA lo empieza a evaluar como una especificación para todos los navegadores y en 1997

se declara estándar (ECMA-262).

A partir de 2016 se empiezan a lanzar versiones cada año, utilizando el número de año como nombre de versión, en lugar de continuar la numeración ES7, ES8, ES9, ES10, etc.

Esta es una breve cronología de las versiones de ECMAScript:

- **ECMAScript 1 (junio de 1997)**: Primera versión del estándar.
- **ECMAScript 2 (junio de 1998)**: pequeña actualización para mantener ECMA-262 sincronizado con el estándar ISO.
- **ECMAScript 3 (diciembre de 1999)**: agrega muchas funciones básicas: como expresiones regulares, mejor manejo de cadenas, nuevas declaraciones de control do-while, switch, manejo de excepciones try/catch, etc.
- **ECMAScript 4 (abandonado en julio de 2008)**: Hubiera sido una actualización masiva (con escritura estática, módulos, espacios de nombres y más), pero terminó siendo demasiado ambicioso y dividió a los administradores del lenguaje.
- **ECMAScript 5 (ES5)(diciembre de 2009)**: trajo mejoras menores: algunas funciones de biblioteca estándar y modo estricto.
- **ECMAScript 5.1 (junio de 2011)**: otra pequeña actualización para mantener sincronizados los estándares Ecma e ISO.
- **ECMAScript 6 (ES6)(junio de 2015)**: una gran actualización que cumplió muchas de las promesas de ECMAScript 4. Esta versión es la primera cuyo nombre oficial, ECMAScript 2015, se basa en el año de publicación.
- **ECMAScript 2016 (junio de 2016)**: primer lanzamiento anual. El ciclo de vida de lanzamiento más corto resultó en menos características nuevas en comparación con el gran ES6. A partir de aquí todas las versiones posteriores de ECMAScript (ES2018, etc.) siempre se ratifican en junio.
- **ECMAScript 2017 (ES2017)** (junio de 2017).
- **ECMAScript 2018 (ES2018)**(junio de 2018). Soporte completo para ES6 en todos los navegadores.
- ECMAScript 2018 (ES2018)(junio de 2018).
- ECMAScript 2019 (ES2019)(junio de 2019)
- ECMAScript 2020 (ES2020)(junio de 2020).
- ECMAScript 2021 (ES2021)(junio de 2021).
- ECMAScript 2022 (ES2022)(junio de 2022).

Ver [changelog](https://tc39.es/ecma262/) de cada versión.

2.5. ECMAScript 2017 (ES8)

- async/await: Permite escribir código asíncrono de una manera más fácil y limpia, utilizando palabras clave async y await en lugar de Promesas y callbacks.
- Object.values() y Object.entries(): Nuevos métodos para extraer los valores y las entradas (pares clave-valor) de un objeto, respectivamente.

- String padding: Nuevos métodos para cadenas de texto, padStart() y padEnd(), que permiten agregar caracteres al principio o al final de una cadena hasta alcanzar una longitud determinada.

2.6. ECMAScript 2018 (ES9)

- Operador de propagación (spread operator) en objetos: Permite crear un nuevo objeto a partir de otro, copiando sus propiedades con una sintaxis más simple y clara (por ejemplo, {...obj}).
- Rest properties en objetos: Permite recoger las propiedades restantes de un objeto en otro objeto (por ejemplo, {a, b, ...rest} = obj).
- Promise.prototype.finally(): Nuevo método para las Promesas que permite ejecutar una función cuando la Promesa se resuelve o se rechaza, independientemente del resultado.
- Expresiones regulares mejoradas: Nuevas características y mejoras en las expresiones regulares, como el soporte para grupos de captura con nombre y el flag "dotAll" (s).

2.7. ECMAScript 2019 (ES10)

- Array.prototype.flat() y Array.prototype.flatMap(): Nuevos métodos para aplanar arrays anidados a diferentes niveles de profundidad y aplicar una función a cada elemento antes de aplanar, respectivamente.
- Object.fromEntries(): Nuevo método que permite crear un objeto a partir de una lista de pares clave-valor.
- String.prototype.trimStart() y String.prototype.trimEnd(): Nuevos métodos para eliminar espacios en blanco al principio y al final de una cadena, respectivamente.

2.8. ECMAScript 2020 (ES11)

- BigInt: Nuevo tipo de dato que permite representar números enteros más grandes que el límite máximo de Number.
- Operador de encadenamiento opcional: Permite acceder a las propiedades de un objeto sin necesidad de comprobar si el objeto es null o undefined (por ejemplo, obj?.prop).
- Nullish coalescing operator: Nuevo operador que devuelve el valor del operando derecho si el valor del operando izquierdo es null o undefined (por ejemplo, a ?? b).

2.9. ECMAScript 2021 (ES12)

- Numeric separators: Mejora la legibilidad de los números en el código al permitir separadores numéricos (por ejemplo, 1_000_000).
- Promise.any(): Nuevo método que devuelve la primera Promesa resuelta en un conjunto de Promesas, rechazando solo si todas las Promesas son rechazadas.
- WeakRef: Nuevo objeto que permite mantener una referencia débil a otro objeto, lo que significa que no evita que el objeto referenciado sea recogido por el recolector de basura.
- FinalizationRegistry: Nueva clase que permite registrar funciones de limpieza que se ejecutan

cuando un objeto es recolectado por el recolector de basura.

- Método replaceAll en Strings:

```
const str = "xxxyz";

// xxyz
console.log(str.replace("x", ""));

// yz
console.log(/x/g, "");

// yz
console.log(str.replaceAll("x", ""));
```

2.10. ECMAScript 2022 (ES12)

- Top-level await
- Private instance fields, methods, and accessors
- Static class fields and methods
- Static class initialization blocks
- Error: .cause
- Array, String, and TypedArray: .at() Method
- Object: .hasOwn()
- RegExp: match .indices ('d' flag)

2.11. TC39

El Ecma Technical Committee 39 o TC39 es el comité que desarrolla JavaScript. Sus miembros son, en sentido estricto, empresas: Adobe, Apple, Facebook, Google, Microsoft, Mozilla, Opera, Twitter y otras. Es decir, empresas que suelen ser competidoras están trabajando juntas por el bien del idioma.

Cada dos meses, el TC39 tiene reuniones a las que asisten delegados designados por los miembros y expertos invitados. Las actas de esas reuniones son públicas en un repositorio de [GitHub](https://github.com/tc39/notes).

3. Ventajas

3.1. Comunidad

La popularidad de JavaScript significa que está bien soportado y bien documentado. Cada vez que crea algo en JavaScript, puede confiar en que muchas personas estarán (potencialmente)

interesadas. Y hay una gran cantidad de programadores de JavaScript en el mercado.

Prueba de esta popularidad es que se mantiene en el top 10 de lenguajes de programación más populares según el índice [TIOBE](https://www.tiobe.com/tiobe-index/).

Ninguna parte/empresa controla JavaScript: lo desarrolla TC39, un comité que comprende muchas organizaciones. El lenguaje se desarrolla a través de un proceso abierto que fomenta la retroalimentación del público.

3.2. Práctico y útil

Con JavaScript, se puede escribir aplicaciones para muchas plataformas de clientes. Estas son algunas tecnologías de ejemplo:

- Las aplicaciones web progresivas se pueden instalar de forma nativa en Android y muchos sistemas operativos de escritorio.
- Electron le permite crear aplicaciones de escritorio multiplataforma.
- React Native le permite escribir aplicaciones para iOS y Android que tienen interfaces de usuario nativas.
- Node.js proporciona un amplio soporte para escribir scripts de shell (además de ser una plataforma para servidores web).

JavaScript es compatible con muchas plataformas y servicios de servidor, por ejemplo:

- Node.js (muchos de los siguientes servicios se basan en Node.js o son compatibles con sus API)
- ZEIT Now
- Microsoft Azure Functions
- AWS Lambda
- Google Cloud Functions

Hay muchas tecnologías de datos disponibles para JavaScript: muchas bases de datos lo admiten y existen capas intermedias (como GraphQL).

Además, el formato de datos estándar JSON (Notación de objetos de JavaScript) se basa en JavaScript y es compatible con su biblioteca estándar. Esto fomenta el uso de JSON como tecnología para el intercambio de comunicación cliente-servidor.

Por último, muchas, si no la mayoría, de las herramientas para JavaScript están escritas en JavaScript. Eso incluye IDE, herramientas de compilación y más. Como consecuencia, los instala de la misma manera que instala sus bibliotecas y puede personalizarlos en JavaScript.

3.3. Lenguaje

- Muchas bibliotecas están disponibles, a través del estándar de facto en el universo de JavaScript, el registro de software npm.
- Si no se está satisfecho con JavaScript "simple" o "vanilla", es relativamente fácil agregar más

funciones:

- Puede compilar funciones de lenguaje modernas y futuras para versiones actuales y pasadas de JavaScript, a través de Babel.
- Puede agregar escritura estática, a través de TypeScript y Flow.
- Puede trabajar con ReasonML, que es, más o menos, OCaml con sintaxis de JavaScript. Se puede compilar a JavaScript o código nativo.
- El lenguaje es flexible: es dinámico y soporta tanto la programación orientada a objetos como la programación funcional.
- JavaScript se ha vuelto sorprendentemente rápido para un lenguaje tan dinámico. – Siempre que no sea lo suficientemente rápido, puede cambiar a WebAssembly, una máquina virtual universal integrada en la mayoría de los motores de JavaScript. Puede ejecutar código estático a velocidades casi nativas.

3.4. Innovación

Hay mucha innovación en el ecosistema de JavaScript: nuevos enfoques para implementar interfaces de usuario, nuevas formas de optimizar la entrega de software y más.

La ventaja es que constantemente aprenderás cosas nuevas. La desventaja es que el cambio constante puede ser agotador a veces. Afortunadamente, las cosas se han ralentizado un poco recientemente: todo ES6 (que fue una modernización considerable del lenguaje) se está consolidando, al igual que ciertas herramientas y flujos de trabajo.

4. Desventajas

Entre los programadores, JavaScript no siempre es muy querido. Una razón es que tiene una buena cantidad de peculiaridades. Algunas de ellas son simplemente formas inusuales de hacer algo. Otros se consideran errores.

De cualquier manera, aprender por qué JavaScript hace algo de la manera en que lo hace, ayuda a lidiar con las peculiaridades y a aceptar JavaScript (tal vez incluso a que le guste).

Además, ahora se han eliminado muchas peculiaridades tradicionales. Por ejemplo:

- Tradicionalmente, las variables de JavaScript no tenían un alcance de bloque. ES6 introdujo let y const, que le permiten declarar variables de ámbito de bloque.
- Antes de ES6, la implementación de fábricas de objetos y herencia a través de funciones y .prototype era ardua y torpe. ES6 introdujo clases, que brindan una sintaxis más conveniente para estos mecanismos.
- Tradicionalmente, JavaScript no tenía módulos integrados. ES6 los agregó al idioma.

Por último, la biblioteca estándar de JavaScript es limitada, pero:

- Hay planes para agregar más funciones.
- Muchas bibliotecas están fácilmente disponibles a través del gestor de paquetes **npm**.

5. Naturaleza de JavaScript

Estos son algunos rasgos del lenguaje:

- Su sintaxis es parte de la familia de lenguajes C (llaves, etc.).
- Es un lenguaje dinámico: la mayoría de los objetos se pueden cambiar de varias formas en tiempo de ejecución, los objetos se pueden crear directamente, etc.
- Es un lenguaje de tipo dinámico: las variables no tienen tipos estáticos fijos y puede asignar cualquier valor a una variable dada (mutable).
- Tiene características de programación funcional: funciones de primera clase, cierres, aplicación parcial vía bind(), etc.
- Tiene características orientadas a objetos: estado mutable, objetos, herencia, clases, etc.
- A menudo falla silenciosamente, es decir, en muchos escenarios en lugar de arrojar una excepción se obtiene un valor undefined.
- Se despliega como código fuente. Pero ese código fuente a menudo se minimiza (se reescribe para requerir menos almacenamiento). Y hay planes para un formato de código fuente binario.
- JavaScript es parte de la plataforma web: es el lenguaje integrado en los navegadores web. Pero también se usa en otros lugares, por ejemplo, en Node.js, para cosas del servidor y secuencias de comandos de shell.
- Los motores de JavaScript a menudo optimizan los mecanismos de lenguaje menos eficientes. Por ejemplo, en principio, las matrices de JavaScript son diccionarios. Pero a bajo nivel de detalle, los motores almacenan arreglos contiguos si tienen índices contiguos.

5.1. Fallos silenciosos

JavaScript a menudo falla en silencio. Veamos dos ejemplos:

Primer ejemplo: si los operandos de un operador no tienen los tipos apropiados, se convierten según sea necesario.

- `'3' * '5'` da como resultado `15`

Segundo ejemplo: si falla un cálculo aritmético, obtiene un valor de error, no una excepción.

- `1 / 0` da como resultado `Infinity`

La razón de las fallas silenciosas es histórica: JavaScript no tuvo excepciones hasta ECMAScript 3. Desde entonces, sus diseñadores han tratado de evitar fallas silenciosas.