JavaScript

JavaScript (abreviado comúnmente **JS**) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos,[2] basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas[3] y JavaScript del lado del servidor (*Server-side JavaScript* o *SSJS*). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

Desde 2012, todos los navegadores modernos soportan completamente ECMAScript 5.1, una versión de JavaScript. Los navegadores más antiguos soportan por lo menos ECMAScript 3. La sexta edición se liberó en julio de 2015.[4]

JavaScript se diseñó con una sintaxis similar a C++ y Java,[5] [6] aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, Java y JavaScript tienen semánticas y propósitos diferentes. Su relación es puramente comercial, tras la compra del creador de Java (Sun Microsystems) de Nestcape Navigator (creador de LiveScript) y el cambio de nombre del lenguaje de programación.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM). Javascript es el único lenguaje de programación que entienden de forma nativa los navegadores.

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. Actualmente es ampliamente utilizado para enviar y recibir información del servidor junto con ayuda de otras tecnologías como

JavaScript

?

Información general	
Extensiones comunes	`.js`
Paradigma	Multiparadigma, programación funcional,[1] programación basada en prototipos, imperativo, interpretado (*scripting*)
Apareció en	4 de diciembre de 1995
Diseñado por	Netscape Communications, Fundación Mozilla
Última versión estable	ECMAScript 2021 (01 de junio de 2021 (1 año, 1 mes y 13 días))
Sistema de tipos	Débil, dinámico, *duck*
Implementaciones	SpiderMonkey, Rhino, KJS,

AJAX. JavaScript se interpreta en el agente de usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

Desde el lanzamiento en junio de 1997 del estándar ECMAScript 1, han existido las versiones 2, 3 y 5, que es la más usada actualmente (la 4 se abandonó[7]). En junio de 2015 se cerró y publicó la versión ECMAScript 6.[8]

	JavaScriptCore, V8, Chakra.
Dialectos	ECMAScript
Influido por	Java, Perl, Self, Python, C, Scheme
Ha influido a	ObjectiveJ, JScript .NET, TIScript

Índice

Historia

Nacimiento de JavaScript

JavaScript fue desarrollado originalmente por Brendan Eich de Netscape con el nombre de *Mocha*, el cual fue renombrado posteriormente a *LiveScript*, para finalmente quedar como JavaScript. El cambio de nombre coincidió aproximadamente con el momento en que Netscape agregó compatibilidad con la tecnología Java en su navegador web Netscape Navigator en la versión 2002 en diciembre de 1995. La denominación produjo confusión, dando la impresión de que el lenguaje es una prolongación de Java, y fue considerada por muchos como una estrategia de mercadotecnia de Netscape para obtener prestigio e innovar en el ámbito de los nuevos lenguajes de programación web.[9] [10]

«JAVASCRIPT» es una marca registrada de Oracle Corporation.[11] Es usada con licencia por los productos creados por Netscape Communications y entidades actuales como la Fundación Mozilla.[12] [13]

Microsoft dio como nombre a su dialecto de JavaScript «JScript», para evitar problemas relacionadas con la marca. JScript fue adoptado en la versión 3.0 de Internet Explorer, liberado en agosto de 1996, e incluyó compatibilidad con el Efecto 2000 con las funciones de fecha, una diferencia de los que se basaban en ese momento. Los dialectos pueden parecer tan similares que los términos «JavaScript» y «JScript» a menudo se utilizan indistintamente, pero la especificación de JScript es incompatible con la de ECMA en muchos aspectos.

Para evitar estas incompatibilidades, el World Wide Web Consortium diseñó el estándar Document Object Model (DOM, o Modelo de Objetos del Documento en español), que incorporan Konqueror, las versiones 6 de Internet Explorer y Netscape Navigator, Opera la versión 7, Mozilla Application Suite y Mozilla Firefox desde su primera versión.[*cita requerida*]

En 1997 los autores propusieron[14] JavaScript para que fuera adoptado como estándar de la European Computer Manufacturers 'Association ECMA, que a pesar de su nombre no es europeo sino internacional, con sede en Ginebra. En junio de 1997 fue adoptado como un estándar ECMA, con el nombre de ECMAScript. Poco después también como un estándar ISO.

JavaScript en el lado servidor

Netscape introdujo una implementación de script del lado del servidor con Netscape Enterprise Server, lanzada en diciembre de 1994 (poco después del lanzamiento de JavaScript para navegadores web).[15] [16] A partir de mediados de la década de los 2000, ha habido una proliferación de implementaciones de JavaScript para el lado servidor. Node.js es uno de los notables ejemplos de JavaScript en el lado del servidor, siendo usado en proyectos importantes.[17] [18]

Desarrollos posteriores

JavaScript se ha convertido en uno de los lenguajes de programación más populares en internet y más usados. Al principio, sin embargo, muchos desarrolladores renegaban del lenguaje porque el público al que va dirigido lo formaban publicadores de artículos y demás aficionados, entre otras razones.[19] La llegada de Ajax devolvió JavaScript a la fama y atrajo la atención de muchos otros programadores. Como resultado de esto hubo una proliferación de un conjunto de frameworks y bibliotecas de ámbito general, mejorando las prácticas de programación con JavaScript, y aumentado el uso de JavaScript fuera de los navegadores web, como se ha visto con la proliferación de entornos JavaScript del lado del servidor. En enero de 2009, el proyecto CommonJS fue inaugurado con el objetivo de especificar una librería para uso de tareas comunes principalmente para el desarrollo fuera del navegador web.[20]

En junio de 2015 se cerró y publicó el estándar ECMAScript 6[21] [22] con un soporte irregular entre navegadores[23] y que dota a JavaScript de características avanzadas que se echaban de menos y que son de uso habitual en otros lenguajes como, por ejemplo, módulos para organización del código, verdaderas clases para programación orientada a objetos, expresiones de flecha, iteradores, generadores o promesas para programación asíncrona.

La versión 7 de ECMAScript se conoce como ECMAScript 2016,[24] y es la última versión disponible, publicada en junio de 2016. Se trata de la primera versión para la que se usa un nuevo procedimiento de publicación anual y un proceso de desarrollo abierto.[25]

Características

Las siguientes características son comunes a todas las implementaciones que se ajustan al estándar ECMAScript, a menos que especifique explícitamente en caso contrario.

Imperativo y estructurado

JavaScript es compatible con gran parte de la estructura de programación de C (por ejemplo, sentencias `if`, bucles `for`, sentencias `switch`, etc.). Con una salvedad, en parte: en C, el ámbito de las variables alcanza al bloque en el cual fueron definidas; sin embargo JavaScript no es compatible con esto, puesto que el ámbito de las variables es el de la función en la cual fueron declaradas. Esto cambia con la versión de ECMAScript 2015, ya que añade compatibilidad con block scoping por medio de la palabra clave `let`. Como en C, JavaScript hace distinción entre expresiones y sentencias. Una diferencia sintáctica con respecto a C es la inserción automática de punto y coma, es decir, en JavaScript los puntos y coma que finalizan una sentencia pueden ser omitidos.[26]

Dinámicos

Tipado dinámico

Como en la mayoría de lenguajes de scripting, el tipo está asociado al valor, no a la variable. Por ejemplo, una variable `x` en un momento dado puede estar ligada a un número y más adelante, religada a una cadena. JavaScript es compatible con varias formas de comprobar el tipo de un objeto, incluyendo duck typing.[27] Una forma de saberlo es por medio de la palabra clave `typeof`.

Objetual

JavaScript está formado casi en su totalidad por objetos. Los objetos en JavaScript son arrays asociativos, mejorados con la inclusión de prototipos (ver más adelante). Los nombres de las propiedades de los objetos son claves de tipo cadena: `obj.x = 10` y `obj['x'] = 10` son equivalentes, siendo azúcar sintáctico la notación con punto. Las propiedades y sus valores pueden ser creados, cambiados o eliminados en tiempo de ejecución. La mayoría de propiedades de un objeto (y aquellas que son incluidas por la cadena de la herencia prototípica) pueden ser enumeradas por medio de la instrucción de bucle `for... in`. JavaScript tiene un pequeño número de objetos predefinidos como son `Function` y `Date`.

Evaluación en tiempo de ejecución

JavaScript incluye la función `eval` que permite evaluar expresiones expresadas como cadenas en tiempo de ejecución. Por ello se recomienda que `eval` sea utilizado con

precaución y que se opte por utilizar la función `JSON.parse()` en la medida de lo posible, pues puede resultar mucho más segura.

Funcional

Funciones de primera clase

A las funciones se les suele llamar ciudadanos de primera clase; son objetos en sí mismos. Como tal, poseen propiedades y métodos, como `.call()` y `.bind()`.[28] Una función anidada es una función definida dentro de otra. Esta es creada cada vez que la función externa es invocada. Además, cada función creada forma una clausura; es el resultado de evaluar un ámbito conteniendo en una o más variables dependientes de otro ámbito externo, incluyendo constantes, variables locales y argumentos de la función externa llamante. El resultado de la evaluación de dicha clausura forma parte del estado interno de cada objeto función, incluso después de que la función exterior concluya su evaluación.[29]

Prototípico

Prototipos

JavaScript usa prototipos en vez de clases para el uso de herencia.[30] Es posible llegar a emular muchas de las características que proporcionan las clases en lenguajes orientados a objetos tradicionales por medio de prototipos en JavaScript.[31]

Funciones como constructores de objetos

Las funciones también se comportan como constructores. Prefijar una llamada a la función con la palabra clave `new` crear una nueva instancia de un prototipo, que heredan propiedades y métodos del constructor (incluidas las propiedades del prototipo de `Object`).[32] ECMAScript 5 ofrece el método `Object.create`, permitiendo la creación explícita de una instancia sin tener que heredar automáticamente del prototipo de Object (en entornos antiguos puede aparecer el prototipo del objeto creado como `null`).[33] La propiedad `prototype` del constructor determina el objeto usado para el prototipo interno de los nuevos objetos creados. Se pueden añadir nuevos métodos modificando el prototipo del objeto usado como constructor. Constructores predefinidos en JavaScript, como `Array` u `Object`, también tienen prototipos que pueden ser modificados. Aunque esto sea posible se considera una mala práctica modificar el prototipo de `Object` ya que la mayoría de los objetos en Javascript heredan los métodos y propiedades del objeto `prototype`, objetos los cuales pueden esperar que estos no hayan sido modificados.[34]

Otras características

Entorno de ejecución

JavaScript normalmente depende del entorno en el que se ejecute (por ejemplo, en un navegador web) para ofrecer objetos y métodos por los que los scripts pueden interactuar con el "mundo exterior". De hecho, depende del entorno para ser capaz de proporcionar la capacidad de incluir o importar scripts (por ejemplo, en HTML por medio del tag `<script>`). (Esto no es una característica del lenguaje, pero es común en la mayoría de las implementaciones de JavaScript.)

Funciones variádicas

Un número indefinido de parámetros pueden ser pasados a la función. La función puede acceder a ellos a través de los parámetros o también a través del objeto local

`arguments`. Las funciones variádicas también pueden ser creadas usando el método `.apply()`.

Funciones como métodos

A diferencia de muchos lenguajes orientados a objetos, no hay distinción entre la definición de función y la definición de método. Más bien, la distinción se produce durante la llamada a la función; una función puede ser llamada como un método. Cuando una función es llamada como un método de un objeto, la palabra clave `this`, que es una variable local a la función, representa al objeto que invocó dicha función.

Arrays y la definición literal de objetos

Al igual que muchos lenguajes de script, arrays y objetos (arrays asociativos en otros idiomas) pueden ser creados con una sintaxis abreviada. De hecho, estos literales forman la base del formato de datos JSON.

Expresiones regulares

JavaScript también es compatible con expresiones regulares de una manera similar a Perl, que proporcionan una sintaxis concisa y poderosa para la manipulación de texto que es más sofisticado que las funciones incorporadas a los objetos de tipo string.[35]

Extensiones específicas del fabricante

JavaScript se encuentra oficialmente bajo la organización de Mozilla Foundation, y periódicamente se añaden nuevas características del lenguaje. Sin embargo, sólo algunos motores JavaScript son compatibles con estas características:

- Las propiedades `get` y `set` (también compatibles con WebKit, Opera,[36] ActionScript y Rhino).[37]
- Cláusulas `catch` condicionales.
- Protocolo iterador adoptado de Python.
- Corrutinas también adoptadas de Python.
- Generación de listas y expresiones por comprensión también adoptado de Python.
- Establecer el ámbito a bloque a través de la palabra clave `let`.
- Desestructuración de arrays y objetos (forma limita de emparejamiento de patrones).
- Expresiones concretas en funciones (function(args) expr).
- ECMAScript para XML (E4X), una extensión que añade compatibilidad nativa XML a ECMAScript.

Sintaxis y semántica

la última versión del lenguaje es ECMAScript 2016 publicada el 17 de junio de 2016.[38]

Ejemplos sencillos

Las variables en JavaScript se definen usando la palabra clave var:[39]

```
var x; // define la variable x, aunque no tiene ningún valor asignado por defecto
var y = 2; // define la variable y y le asigna el valor 2 a ella
```

A considerar los comentarios en el ejemplo de arriba, los cuales van precedidos con 2 barras diagonales.

No existen funcionalidades para I/O incluidas en el lenguaje; el entorno de ejecución ya lo proporciona. La especificación ECMAScript en su edición 5.1 hace mención:[40]

> ... en efecto, no existen provisiones en esta especificación para entrada de datos externos o salida para resultados computados.

Sin embargo, la mayoría de los entornos de ejecución tiene un objeto[41] llamado `console` que puede ser usado para imprimir por el flujo de salida de la consola de depuración. He aquí un simple programa que imprime "Hello world!":

```
console.log("Hello world!");
```

Una función recursiva:

```
function factorial(n) {
    if (n === 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

Ejemplos de función anónima (o función lambda) y una clausura:

```
var displayClosure = function() {
    var count = 0;
    return function () {
        return ++count;
    };
}
var inc = displayClosure();
inc(); // devuelve 1
inc(); // devuelve 2
inc(); // devuelve 3
```

Las expresiones con invocación automática permiten a las funciones pasarle variables por parámetro dentro de sus propias clausuras.

```
var v;
v = 1;
var getValue = (function(v) {
  return function() {return v;};
}(v));

v = 2;

getValue(); // 1
```

Ejemplos más avanzados

El siguiente código muestra varias características de JavaScript.

```
/* Busca el mínimo común múltiplo (MCM) de dos números */
function LCMCalculator(x, y) { // función constructora
```

```
    var checkInt = function (x) { // función interior
        if (x % 1 !== 0) {
            throw new TypeError(x + " no es un entero"); // lanza una excepción
        }
        return x;
    };
    this.a = checkInt(x) // puntos y coma son opcionales
    this.b = checkInt(y);
}
// El prototipo de las instancias de objeto creados por el constructor es el de la propiedad "prototype" del constructor.
LCMCalculator.prototype = { // objeto definido como literal
        constructor: LCMCalculator, // cuando reasignamos un prototipo, establecemos correctamente su propiedad constructor
    gcd: function () { // método que calcula el máximo común divisor
        // Algoritmo de Euclides:
        var a = Math.abs(this.a), b = Math.abs(this.b), t;
        if (a < b) {
            // intercambiamos variables
            t = b;
            b = a;
            a = t;
        }
        while (b !== 0) {
            t = b;
            b = a % b;
            a = t;
        }
        // Solo necesitamos calcular el MCD una vez, por lo tanto 'redefinimos' este método.
          // (Realmente no es una redefinición—está definida en la propia instancia, por lo tanto
                    //  this.gcd se refiere a esta 'redefinición' en vez de a LCMCalculator.prototype.gcd).
        // Además, 'gcd' === "gcd", this['gcd'] === this.gcd
        this['gcd'] = function () {
            return a;
        };
        return a;
    },
      // Los nombres de las propiedades del objeto pueden ser especificados con cadenas delimitadas con comillas simples (') o dobles (").
    "lcm" : function () {
         // Los nombres de las variables no colisionan con las propiedades del objeto. Por ejemplo: |lcm| no es |this.lcm|.
        // No usar |this.a * this.b| para evitar problemas con cálculos en coma flotante.
        var lcm = this.a / this.gcd() * this.b;
        // Sólo necesitamos calcular MCM una vez, por lo tanto "redefinimos" este método.
        this.lcm = function () {
            return lcm;
        };
        return lcm;
    },
    toString: function () {
        return "LCMCalculator: a = " + this.a + ", b = " + this.b;
    }
};

// Definimos una función genérica para imprimir un resultado; esta implementación solo funciona en los navegadores web
function output(x) {
    document.body.appendChild(document.createTextNode(x));
    document.body.appendChild(document.createElement('br'));
}

// Nota: Los métodos.map() y.forEach() del prototipo Array están definidos en JavaScript 1.6.
// Estos métodos son usados aquí para demostrar la naturaleza funcional inherente del lenguaje.
[[25, 55], [21, 56], [22, 58], [28, 56]].map(function (pair) { // construcción literal de un Array + función de mapeo.
    return new LCMCalculator(pair[0], pair[1]);
}).sort(function (a, b) { // ordenamos la colección por medio de esta función
    return a.lcm() - b.lcm();
}).forEach(function (obj) {
    output(obj + ", gcd = " + obj.gcd() + ", lcm = " + obj.lcm());
});
```

El siguiente ejemplo muestra la salida que debería ser mostrada en la ventana de un navegador.

```
LCMCalculator: a = 28, b = 56, gcd = 28, lcm = 56
LCMCalculator: a = 21, b = 56, gcd = 7, lcm = 168
LCMCalculator: a = 25, b = 55, gcd = 5, lcm = 275
LCMCalculator: a = 22, b = 58, gcd = 2, lcm = 638
```

Uso en páginas web

Véanse también: Intérprete de JavaScript *y* AJAX.

El uso más común de JavaScript es escribir funciones embebidas o incluidas en páginas HTML y que interactúan con el Document Object Model (DOM o Modelo de Objetos del Documento) de la página. Algunos ejemplos sencillos de este uso son:

- Cargar nuevo contenido para la página o enviar datos al servidor a través de AJAX sin necesidad de recargar la página (por ejemplo, una red social puede permitir al usuario enviar actualizaciones de estado sin salir de la página).
- Animación de los elementos de página, hacerlos desaparecer, cambiar su tamaño, moverlos, etc.
- Contenido interactivo, por ejemplo, juegos y reproducción de audio y vídeo.
- Validación de los valores de entrada de un formulario web para asegurarse de que son aceptables antes de ser enviado al servidor.
- Transmisión de información sobre los hábitos de lectura de los usuarios y las actividades de navegación a varios sitios web. Las páginas Web con frecuencia lo hacen para hacer análisis web, seguimiento de anuncios, la personalización o para otros fines.[42]

Dado que el código JavaScript puede ejecutarse localmente en el navegador del usuario (en lugar de en un servidor remoto), el navegador puede responder a las acciones del usuario con rapidez, haciendo una aplicación más sensible. Por otra parte, el código JavaScript puede detectar acciones de los usuarios que HTML por sí sola no puede, como pulsaciones de teclado. Las aplicaciones como Gmail se aprovechan de esto: la mayor parte de la lógica de la interfaz de usuario está escrita en JavaScript, enviando peticiones al servidor (por ejemplo, el contenido de un mensaje de correo electrónico). La tendencia cada vez mayor por el uso de la programación Ajax explota de manera similar esta técnica.

Un motor de JavaScript (también conocido como intérprete de JavaScript o implementación JavaScript) es un intérprete que interpreta el código fuente de JavaScript y ejecuta la secuencia de comandos en consecuencia. El primer motor de JavaScript fue creado por Brendan Eich en Netscape Communications Corporation, para el navegador web Netscape Navigator. El motor, denominado SpiderMonkey, está implementado en C. Desde entonces, ha sido actualizado (en JavaScript 1.5) para cumplir con el ECMA-262 edición 3. El motor Rhino, creado principalmente por Norris Boyd (antes de Netscape, ahora en Google) es una implementación de JavaScript en Java. Rhino, como SpiderMonkey, es compatible con el ECMA-262 edición 3.

Un navegador web es, con mucho, el entorno de acogida más común para JavaScript. Los navegadores web suelen crear objetos no nativos, dependientes del entorno de ejecución, para representar el Document Object Model (DOM) en JavaScript. El servidor web es otro entorno común de servicios. Un servidor web JavaScript suele exponer sus propios objetos para representar objetos de petición y respuesta HTTP, que un programa JavaScript podría entonces interrogar y manipular para generar dinámicamente páginas web.

Debido a que JavaScript es el único lenguaje por el que los más populares navegadores comparten su apoyo, se ha convertido en un lenguaje al que muchos frameworks en otros lenguajes compilan, a pesar de que JavaScript no fue diseñado para tales propósitos.[43] A pesar de las limitaciones de rendimiento inherentes a su naturaleza dinámica, el aumento de la velocidad de los motores de JavaScript ha hecho de este lenguaje un entorno para la compilación sorprendentemente factible.

Ejemplo de script

A continuación se muestra un breve ejemplo de una página web (ajustándose a las normas del estándar para HTML5) que utiliza JavaScript para el manejo del DOM:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Ejemplo sencillo</title>
</head>
<body>
    <h1 id="header">Esto es JavaScript</h1>

    <script>
        document.body.appendChild(document.createTextNode('Hola Mundo!'));

        var h1 = document.getElementById('header'); // contiene la referencia al tag <h1>
        h1 = document.getElementsByTagName('h1')[0]; // accediendo al mismo elemento <h1>
    </script>

    <noscript>Tu navegador no admite JavaScript, o JavaScript está deshabilitado.</noscript>
</body>
</html>
```

Consideraciones acerca de la compatibilidad

Debido a que JavaScript se ejecuta en entornos muy variados, una parte importante de las pruebas y la depuración es probar y verificar que el código JavaScript funciona correctamente en múltiples navegadores. La interfaz DOM para acceder y manipular páginas web no es parte del estándar ECMAScript, o de la propia JavaScript. El DOM es definido por los esfuerzos de estandarización del W3C, una organización independiente. En la práctica, las implementaciones que hacen de JavaScript los distintos navegadores difieren tanto entre ellos mismos como de las normas del estándar.

Para hacer frente a estas diferencias, los autores de JavaScript pudieron ser capaces de escribir código compatible con los estándares que también fuera capaz de ejecutarse correctamente en la mayoría de los navegadores, o en su defecto, que al menos se pudiera escribir código capaz de comprobar la presencia de ciertas funcionalidades del navegador y que se comportase de manera diferente si no se dispusiese de dicha funcionalidad.[44] Existen casos en los que dos navegadores pueden llegar a implementar la misma característica, pero con un comportamiento diferente, hecho que a los programadores les puede resultar de ayuda para detectar qué navegador se está ejecutando en ese instante y así cambiar el comportamiento de su escritura para que coincida.[45] [46] Los programadores también suelen utilizar bibliotecas o herramientas que tengan en cuenta las diferencias entre navegadores.

Además, los scripts pueden no funcionar para algunos usuarios. Por ejemplo, un usuario puede:

- Utilizar un navegador antiguo sin compatibilidad completa con la API DOM,
- Utilizar un navegador PDA o teléfono móvil que no puede ejecutar JavaScript
- Tener la ejecución de JavaScript deshabilitada, como precaución de seguridad,

- Utilizar un navegador de voz debido a, por ejemplo, una discapacidad visual.

Para apoyar a estos usuarios, los programadores web suelen crear páginas que sean tolerante de fallos según el agente de usuario (tipo de navegador) que no admita JavaScript. En particular, la página debe seguir siendo útil aunque sin las características adicionales que JavaScript habría añadido. Un enfoque alternativo que muchos encuentran preferible es primero crear contenido utilizando las tecnologías que funcionan en todos los navegadores, y mejorar el contenido para los usuarios que han permitido JavaScript.

Accesibilidad

Suponiendo que el usuario no haya desactivado la ejecución de código JavaScript, en el lado del cliente JavaScript debe ser escrito tanto con el propósito de mejorar las experiencias de los visitantes con discapacidad visual o física, como el de evitar ocultar información a estos visitantes.[47]

Los lectores de pantalla, utilizados por los ciegos y deficientes visuales, pueden ser tenidos en cuenta por JavaScript y así poder acceder y leer los elementos DOM de la página. El código HTML escrito debe ser lo más conciso, navegable y semánticamente rico posible, tanto si JavaScript se ejecuta como si no.

JavaScript no debería de ser totalmente dependiente de los eventos de ratón del navegador y debería ser accesible para aquellos usuarios que no quieran hacer uso del ratón (informática) para navegar o que opten por utilizar solamente el teclado. Hay eventos independientes del dispositivo, tales como `onfocus` y `onchange` que son preferibles en la mayoría de los casos.[47]

JavaScript no debe ser utilizado para crear confusión o desorientación al usuario web. Por ejemplo, modificar o desactivar la funcionalidad normal del navegador, como cambiar la forma en que el botón de navegar hacia atrás o el evento de actualización se comportan, son prácticas que generalmente son mejores evitar. Igualmente, desencadenar eventos que el usuario puede no tener en cuenta reduce la sensación de control del usuario y provoca cambios inesperados al contenido de la página.[48]

A menudo, el proceso de dotar a una página web compleja el mayor grado accesibilidad posible, se convierte en un problema no trivial donde muchos temas se acaban llevando al debate y a la opinión, siendo necesario el compromiso de todos hasta el final. Sin embargo, los agentes de usuario y las tecnologías de apoyo a personas con discapacidad están en constante evolución y nuevas directrices e información al respecto siguen publicándose en la web.[47]

Seguridad

Véase también: Seguridad del navegador

JavaScript y el DOM permite que existan programadores que hagan un uso inapropiado para introducir scripts que ejecuten código con contenido malicioso sin el consentimiento del usuario y que pueda así comprometer su seguridad.

Los desarrolladores de los navegadores tienen en cuenta este riesgo utilizando dos restricciones. En primer lugar, los scripts se ejecutan en un sandbox en el que sólo se pueden llevar a cabo acciones relacionadas con la web, no con tareas de programación de propósito general, como la creación de archivos. En segundo lugar, está limitada por la política del mismo origen: los scripts de un sitio web no tienen acceso a la información enviada a otro sitio web (de otro dominio) como pudiera ser nombres de usuario, contraseñas o cookies. La mayoría de los fallos de seguridad de JavaScript están relacionados con violaciones de cualquiera de estas dos restricciones.

Existen proyectos como AdSafe o Secure ECMA script (SES) que proporcionan mayores niveles de seguridad, en especial en el código creado por terceros (tales como los anuncios).[49] [50]

La Política de Contenido Seguro (CSP) es el método principal previsto para garantizar que sólo código de confianza pueda ser ejecutado en una página web.

Vulnerabilidades cross-site

Un problema común de seguridad en JavaScript es el cross-site scripting o XSS, una violación de la política de mismo origen. Las vulnerabilidades XSS permiten a un atacante inyectar código JavaScript en páginas web visitadas por el usuario. Una de esas webs podría ser la de un banco, pudiendo el atacante acceder a la aplicación de banca con los privilegios de la víctima, lo que podría revelar información secreta o transferir dinero sin la autorización de la víctima. Una solución para las vulnerabilidades XSS es utilizar *HTML escaping* cuando se muestre información de fuentes no confiables.

Algunos navegadores incluyen una protección parcial contra los ataques XSS reflejados (el atacante está en la misma petición web). El atacante proporciona una URL incluyendo código malicioso. Sin embargo, incluso los usuarios de los navegadores son vulnerables a otros ataques XSS, tales como aquellos en los que el código malicioso se almacena en una base de datos. Solo el correcto diseño de las aplicaciones Web en la parte servidora puede prevenir totalmente XSS. Las vulnerabilidades XSS también pueden ocurrir debido a errores de ejecución por los desarrolladores del navegador.[51]

Otra vulnerabilidad es la falsificación de petición de sitio cruzado o CSRF. En CSRF, el código del sitio web atacante engaña al navegador de la víctima, permitiendo al atacante realizar peticiones en nombre de la víctima, haciendo imposible saber a la aplicación de destino (por ejemplo, la de un banco haciendo una transferencia de dinero) saber si la petición ha sido realizada voluntariamente por el usuario o por un ataque CSRF.

El ataque funciona porque, si el sitio de destino hace uso únicamente de las cookies para autenticar las solicitudes de la víctima, las peticiones iniciadas por el código del atacante tendrán las mismas credenciales de acceso legítimo que las solicitudes iniciadas por el propio usuario.

En general, la solución a CSRF consiste en introducir un campo de formulario oculto cuyo valor se utilice para realizar la autenticación, y no solo por medio de las cookies, en solicitudes que puedan tener efectos duraderos. La comprobación de la cabecera HTTP referer también puede servir de ayuda.

"Hijacking JavaScript" es un tipo de ataque CSRF en el que una etiqueta <script> en el sitio web del atacante explota una vulnerabilidad en la página del sitio de la víctima que le hace devolver información privada, en forma de JSON o código JavaScript. Las posibles soluciones son:

- que se requiera un token de autenticación en los parámetros de las peticiones POST y GET para aquellas peticiones que requieran devolver información privada del usuario.
- usar POST y nunca GET para solicitudes que devuelven información privada

Herramientas de desarrollo

En JavaScript, disponer de un depurador se convierte en necesario cuando se desarrollan grandes aplicaciones, no triviales. Dado que puede haber diferencias de implementación entre los diferentes navegadores (especialmente en cuanto al DOM), es útil tener acceso a un depurador para cada uno de los navegadores a los cuales nuestra aplicación web irá dirigido.[52]

Los depuradores web están disponibles para Internet Explorer, Firefox, Safari, Google Chrome y Opera.[53]

Existen tres depuradores disponibles para Internet Explorer: Microsoft Visual Studio es el más avanzado de los tres, seguido de cerca por Microsoft Script Editor (un componente de Microsoft Office)[54] y, finalmente, Microsoft Script Debugger, que es mucho más básico que el otro dos, aunque es gratuito. El IDE gratuito Microsoft Visual Web Developer Express ofrece una versión limitada de la funcionalidad de depuración de JavaScript en el Microsoft Visual Studio. Internet Explorer ha incluido herramientas de desarrollo desde la versión 8 (se muestra pulsando la tecla F12). Las aplicaciones web dentro de Firefox se pueden depurar usando el Firebug add-on o el antiguo depurador Venkman. Firefox también tiene integrada una consola de errores básica, que registra y evalúa JavaScript. También registra errores de CSS y advertencias. Opera incluye un conjunto de herramientas llamado Dragonfly.[55] El Inspector Web de WebKit incluye un depurador de JavaScript[56] utilizado en Safari, junto con una versión modificada de Google Chrome.

Existen algunas herramientas de ayuda a la depuración, también escritas en JavaScript y construidas para ejecutarse en la Web. Un ejemplo es el programa JSLint, desarrollado por Douglas Crockford, quien ha escrito extensamente sobre el lenguaje. JSLint analiza el código JavaScript para que este quede conforme con un conjunto de normas y directrices y que aseguran su correcto funcionamiento y mantenibilidad.

Véase también

- VBScript
- AJAX
- Aplicación web
- Document Object Model
- jQuery
- JSON
- HTML
- HTML5

Referencias

1. Douglas Crockford on Functional JavaScript (https://web.archive.org/web/20090923001111/http://www.blinkx.com/video/douglas-crockford-on-functional-javascript/xscZz8XhfuNQ_aaVuyUB2A) (2:49): " [JavaScript] es el lenguaje funcional más popular del mundo. JavaScript es y siempre ha sido, al menos desde [la versión] 1.2, un lenguaje de programación funcional."
2. «Copia archivada» (https://web.archive.org/web/20150412040502/http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf). Archivado desde el original (http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf) el 12 de abril de 2015. Consultado el 23 de noviembre de 2010.
3. M. Domínguez-Dorado,. Todo Programación. Nº 12. Págs. 48-51. Editorial Iberprensa (Madrid). DL M-13679-2004. Septiembre de 2005. *Bases de datos en el cliente con JavaScript DB*.
4. «JavaScript» (https://developer.mozilla.org/es/docs/Web/JavaScript). *Mozilla Developer Network*. Consultado el 16 de septiembre de 2016.
5. «About JavaScript - JavaScript | MDN» (https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript). *developer.mozilla.org* (en inglés estadounidense). Consultado el 4 de julio de 2022.
6. «Introduction» (https://web.stanford.edu/class/cs98si/slides/overview.html).

web.stanford.edu. Consultado el 4 de julio de 2022.

7. Eich, Brendan (13 de agosto de 2008). «ECMAScript Harmony» (https://mail.mozilla.org/pipermail/es-discuss/2008-August/003400.html). Consultado el 22 de julio de 2015.
8. campusMVP (19 de junio de 2015). «ECMAScript 6 es ya un estándar cerrado» (http://www.campusmvp.es/recursos/post/ECMAScript-6-es-ya-un-estandar-cerrado.aspx). Consultado el 22 de julio de 2015.
9. Lenguajes de programación usados en Internet y la World Wide Web (WWW) (castellano) (http://www.webdevelopersnotes.com/basics/languages_on_the_internet.php3)
10. JavaScript: The Definitive Guide, 6th Edition (http://shop.oreilly.com/product/9780596805531.do)
11. http://tsdr.uspto.gov/#caseNumber=75026640&caseType=SERIAL_NO&searchType=statusSe
12. Marcas registradas de Oracle (http://www.oracle.com/us/legal/third-party-trademarks/index.html).
13. «About JavaScript» (https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript?redirectlocale=en-US&redirectslug=JavaScript%2FAbout_JavaScript) (en inglés). Consultado el 29 de agosto de 2013. «JavaScript is a trademark or registered trademark of Oracle in the U.S. and other countries».
14. Netscape Comunicado de prensa (castellano) (http://cgi.netscape.com/newsref/pr/newsrelease289.html)
15. «Chapter 2: Getting Started» (http://docs.oracle.com/cd/E19957-01/816-6411-10/getstart.htm). *Server-Side JavaScript Guide*. Netscape Communications Corporation. 1998. Consultado el 25 de abril de 2012.
16. Mike Morgan (1996). «Chapter 6: Netscape Internet Application Framework» (https://web.archive.org/web/20121221071020/http://vampire.rulez.org/onlinedoc/book/NetscapeLiveWire/ch6.htm). *Using Netscape™ LiveWire™, Special Edition*. Que. Archivado desde el original (http://vampire.rulez.org/onlinedoc/book/NetscapeLiveWire/ch6.htm) el 21 de diciembre de 2012. Consultado el 19 de mayo de 2013.
17. «Server-Side Javascript: Back With a Vengeance» (https://web.archive.org/web/20120602170854/http://www.readwriteweb.com/archives/server-side_javascript_back_with_a_vengeance.php). *Read Write Web*. 17 de diciembre de 2009. Archivado desde el original (http://www.readwriteweb.com/archives/server-side_javascript_back_with_a_vengeance.php) el 2 de junio de 2012. Consultado el 28 de mayo de 2012.
18. «Node's goal is to provide an easy way to build scalable network programs» (http://nodejs.org/about/). *About Node.js*. Joyent.
19. «JavaScript: The World's Most Misunderstood Programming Language» (http://www.crockford.com/javascript/javascript.html). Crockford.com. Consultado el 19 de mayo de 2009.
20. Kris Kowal (1 de diciembre de 2009). «CommonJS effort sets JavaScript on path for world domination» (http://arstechnica.com/web/news/2009/12/commonjs-effort-sets-javascript-on-path-for-world-domination.ars). *Ars Technica*. Condé Nast Publications. Consultado el 18 de abril de 2010.
21. ECMA International (Junio de 2015). «Standard ECMA-262 6th Edition» (http://www.ecma-international.org/publications/standards/Ecma-262.htm).
22. campusMVP (19 de junio de 2015). «ECMAScript 6 ya es un estándar cerrado» (http://www.campusmvp.es/recursos/post/ECMAScript-6-es-ya-un-estandar-cerrado.aspx). Consultado el 22 de julio de 2015.
23. «Tabla de compatibilidad de navegadores con ECMAScript 6 (Inglés)» (http://kangax.github.io/compat-table/es6/).
24. ECMA International (Junio de 2016). «Standard ECMA-262 7th Edition» (http://www.ecma-international.org/publications/standards/Ecma-262.htm).
25. ECMA International (Junio de 2016). «ECMAScript 2016 Language Specification» (http://www.ecma-international.org/ecma-262/7.0/index.html).
26. Flanagan, 2006, p. 16.
27. Flanagan, 2006, pp. 176–178.
28. Properties of the Function Object (https://es5.github.com/#x15.3.4-toc)
29. Flanagan, 2006, p. 141.
30. «Inheritance and the prototype chain» (https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Inheritance_and_the_prototype_chain). *Mozilla Developer Network*. Mozilla. Consultado el 6 de abril de 2013.

31. Herman, David (2013). *Effective Javascript*. Addison-Wesley. p. 83. ISBN 9780321812186.
32. Haverbeke, Marjin (2011). *Eloquent Javascript* (https://archive.org/details/eloquentjavascri00have_492). No Starch Press. pp. 95 (https://archive.org/details/eloquentjavascri00have_492/page/n102)-97. ISBN 9781593272821.
33. Katz, Yehuda. «Understanding "Prototypes" in JavaScript» (http://yehudakatz.com/2011/08/12/understanding-prototypes-in-javascript/). Consultado el 6 de abril de 2013.
34. Herman, David (2013). *Effective Javascript*. Addison-Wesley. pp. 125-127. ISBN 9780321812186.
35. Haverbeke, Marijn (2011). *Eloquent JavaScript* (https://archive.org/details/eloquentjavascri00have_492). No Starch Press. pp. 139 (https://archive.org/details/eloquentjavascri00have_492/page/n146)-149. ISBN 978-1593272821.
36. Robert Nyman, Getters And Setters With JavaScript – Code Samples And Demos (http://robertnyman.com/2009/05/28/getters-and-setters-with-javascript-code-samples-and-demos/), published 29 May 2009, accessed 2 January 2010.
37. John Resig, JavaScript Getters and Setters (http://ejohn.org/blog/javascript-getters-and-setters/), 18 July 2007, accessed 2 January 2010
38. http://www.ecma-international.org/publications/standards/Ecma-262.htm
39. «var - JavaScript - MDN» (https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Statements/var). The Mozilla Developer Network. Consultado el 22 de diciembre de 2012.
40. «ECMAScript Language Specification - ECMA-262 Edition 5.1» (http://www.ecma-international.org/ecma-262/5.1/#sec-4). Ecma International. Consultado el 22 de diciembre de 2012.
41. «console» (https://developer.mozilla.org/en-US/docs/DOM/console). *Mozilla Developer Network*. Mozilla. Consultado el 6 de abril de 2013.
42. «JavaScript tracking - Piwik» (https://web.archive.org/web/20131031182703/http://piwik.org/docs/javascript-tracking/). Piwik. Archivado desde el original (http://piwik.org/docs/javascript-tracking/) el 31 de octubre de 2013. Consultado el 31 de marzo de 2012.
43. Hamilton, Naomi (31 de junio de 2008). «The A-Z of Programming Languages: JavaScript» (http://www.computerworld.com.au/article/255293/-z_programming_languages_javascript). computerworld.com.au.
44. Peter-Paul Koch, Object detection (http://www.quirksmode.org/js/support.html)
45. Peter-Paul Koch, Mission Impossible - mouse position (http://www.evolt.org/node/23335)
46. Peter-Paul Koch, Browser detect (http://www.quirksmode.org/js/detect.html)
47. Flanagan, 2006, pp. 262–263.
48. «Creating Accessible JavaScript» (http://www.webaim.org/techniques/javascript/). WebAIM. Consultado el 8 de junio de 2010.
49. ADsafe - Making JavaScript Safe for Advertising (http://www.adsafe.org/)
50. Secure ECMA Script (SES) (https://code.google.com/p/es-lab/wiki/SecureEcmaScript)
51. MozillaZine, Mozilla Cross-Site Scripting Vulnerability Reported and Fixed (http://www.mozillazine.org/talkback.html?article=4392)
52. «Advanced Debugging With JavaScript» (http://www.alistapart.com/articles/advanced-debugging-with-javascript/). alistapart.com. 3 de febrero de 2009. Consultado el 28 de mayo de 2010.
53. «The JavaScript Debugging Console» (http://javascript.about.com/od/problemsolving/ig/JavaScript-Debugging/). javascript.about.com. 28 de mayo de 2010. Consultado el 28 de mayo de 2010.
54. JScript development in Microsoft Office 11 (http://msdn2.microsoft.com/en-us/library/aa202668(office.11).aspx) (MS InfoPath 2003)
55. «Opera DragonFly» (https://www.webcitation.org/618EWbOQ4?url=http://www.opera.com/dragonfly/). Opera Software. Archivado desde el original (http://www.opera.com/dragonfly/) el 22 de agosto de 2011. Consultado el 19 de mayo de 2013.
56. «Introducing Drosera - Surfin' Safari» (http://webkit.org/blog/61/introducing-drosera/). Webkit.org. 28 de junio de 2006. Consultado el 19 de mayo de 2009.

Enlaces externos

- Mozilla Developer Center (https://developer.mozilla.org/es/docs/Web/JavaScript).
- (en inglés) Javascript tutorial (http://www.w3schools.com/js/).
- JavaScript - Mirando tras las bambalinas (https://web.archive.org/web/20220327060138/https://reanimandowebs.com/articulos-sobre-programacion-web/javascript/).
