**Resumen JavaScript**

**Introducción**

JavaScript es un lenguaje de scripting multiplataforma, orientado a objetos. Esta diseñado para ser fácilmente embebido en otros productos y aplicaciones, como ser web browsers. Dentro de un entorno de desarrollo, además puede ser conectado a los objetos de este entorno y proveer un control programático sobre ellos.

El núcleo de JavaScript contiene un set de objetos predefinidos y un set de elementos del lenguaje. El núcleo puede ser complementado para una variedad de propósitos utilizando objetos adicionales; por ejemplo:

* *JavaScript* *del lado del cliente* extiende el núcleo del lenguaje proporcionando objetos para controlar el browser (Navigator u otros Browsers) y su Modelo de Documentos Orientados a Objeto (DOM). Por ejemplo, las extensiones del lado del cliente permiten a una aplicación colocar elementos en un formulario HTML y responder a los eventos del usuario como clicks del mouse, ingresos en formularios, y navegación en la página.
* *Javascript del lado del servidor* extiende el núcleo del lenguaje proporcionando objetos relevantes para el servidor. Por ejemplo, las extensiones del lado del servidor permiten a las aplicaciones comunicarse con una base de datos relacional, proveyendo un flujo constante de información, o realizar movimientos de archivos en el servidor.

**JavaScript y Java**

A través de LiveConnect, se puede lograr que el código java y el de javascript se comuniquen entre sí. Desde JavaScript, se puede instanciar objetos de java y acceder a sus métodos públicos y campos. Desde Java, podes acceder a Objetos de Javascript, propiedades y métodos.

Netscape invento JavaScript, y JavaScript fue usado por primera vez en navegadores Netscape.

|  |  |
| --- | --- |
| **JavaScript** | **Java** |
| El lenguaje se parece al de Java pero no tiene el tipado estático y comprobación de tipos fuerte. | Java es un lenguaje de programación basado en clases diseñado para una rápida ejecución y la seguridad de tipos. |
| Tiene sintaxis, convenciones de nombres y controles básicos de flujo parecidos a los de Java, por eso es que se le cambió el nombre de Livescript a javascript. | Usted no tiene que declarar todas las variables, clases y métodos. Usted no tiene que preocuparse por si los métodos son públicos, privados o protegidos, y usted no tiene que implementar interfaces. Variables, parámetros y tipos de devolución de funciones no se escriben de forma explícita. |
| Soporta un sistema en tiempo de ejecución basado en un pequeño número de tipos de datos que representan valores numéricos, booleanos, y cadenas. | Sistema de clases construidas por declaraciones en tiempo de compilación de Java. |
| Javascript tiene un modelo de objetos basado en prototipos en lugar del más común modelo de objetos basado en clases. | Modelo basado en la clase significa que los programas consisten exclusivamente de clases y sus métodos. |
| El modelo basado en prototipos provee herencia dinámica; esto es, lo que se hereda puede variar para objetos individuales. | Herencia y la tipificación estricta generalmente requieren estrechamente unidas jerarquías de objetos. |
| Soporta funciones sin ningún requerimiento declarativo especial. Las funciones pueden ser propiedades de objetos, ejecutándose como métodos débilmente tipados. | Clase de base. Los objetos se dividen en clases e instancias con toda herencia a través de la jerarquía de clases. Las clases y los casos no pueden tener propiedades o métodos añadidos dinámicamente. |
| Orientada a objetos. No se distingue entre tipos de objetos. La herencia es a través del mecanismo de prototipo, y las propiedades y métodos se puede añadir a cualquier objeto de forma dinámica. | Tipos de datos de variables deben ser declaradas (tipos estáticos). |
| Tipos de datos variables no se declaran (tipado dinámico). |  |

**JavaScript y la especificación ECMAScript**

La asociación europea para la normalización de los sistemas de información y comunicación (ECMA antes era un acrónimo de la Asociación Europea de fabricantes de equipo) para entregar un lenguaje de programación internacional estandarizado basado en el núcleo de JavaScript. Esta versión estandarizada de JavaScript, llamado ECMAScript, se comporta de la misma manera en todas las aplicaciones que soportan el estándar. Las empresas pueden utilizar el lenguaje estándar abierto para desarrollar su aplicación de JavaScript.

El estándar ECMAScript está documentado en la especificación ECMA-262.

El estándar ECMA-262 también está aprobado por la ISO (Organización Internacional de Normalización) como ISO-16262.

La especificación ECMAScript no describe el Document Object Model (DOM), que está estandarizado por el World Wide Web Consortium (W3C). El DOM define la forma en que los objetos de documento HTML se exponen a su guión.

**Relación entre versiones de JavaScript y ediciones de ECMAScript**

Netscape trabajó estrechamente con Ecma Internacional para producir la Especificación ECMAScript (ECMA-262).

La referencia de JavaScript indica las funciones del lenguaje que son ECMAScript compatible.

JavaScript siempre incluirá características que no forman parte de la especificación ECMAScript; JavaScript es compatible con ECMAScript, mientras que proporciona características adicionales.

|  |  |
| --- | --- |
| **JavaScript version** | **Relación con la edición de ECMAScript** |
| JavaScript 1.1 | ECMA-262, Edición 1 se basa en JavaScript 1.1 |
| JavaScript 1.2 | ECMA-262 no fue completa cuando JavaScript 1.2 fue lanzado. JavaScript 1.2 no es totalmente compatible con ECMA-262, Edición 1, por las siguientes razones:   * Netscape desarrolló características adicionales en JavaScript 1.2 que no fueron considerados para ECMA-262. * ECMA-262 añade dos nuevas características: la internacionalización mediante Unicode, y un comportamiento uniforme en todas las plataformas. Varias características de JavaScript 1.2, como el objeto Date, eran dependientes de la plataforma y utilizan el comportamiento específico de la plataforma. |
| JavaScript 1.3 | JavaScript 1.3 es completamente compatible con ECMA-262, Edición 1.  JavaScript 1.3 resolvió las inconsistencias que JavaScript 1.2 tenía con ECMA-262, manteniendo todas las características adicionales de JavaScript 1.2, excepto == y! =, Que se han cambiado para ajustarse a ECMA-262. |
| JavaScript 1.4 | JavaScript 1.4 es completamente compatible con ECMA-262, Edición 1.  La tercera versión de la especificación ECMAScript no se ha finalizado cuando JavaScript 1.4 fue lanzado. |
| JavaScript 1.5 | JavaScript 1.5 es completamente compatible con ECMA-262, Edición 3 |

**Documentación JavaScript frente a las especificaciones ECMAScript**

La especificación ECMAScript es un conjunto de requisitos para la implementación de ECMAScript; es útil si se desea determinar si una función JavaScript está soportada en otras implementaciones ECMAScript. Si va a escribir código JavaScript que sólo utiliza las funciones compatibles con ECMAScript, entonces es posible que tenga que revisar la especificación ECMAScript.

El documento ECMAScript no está destinado a ayudar a los programadores de scripts; usar la documentación de JavaScript para la información sobre la escritura de guiones.

**Terminología**

La especificación ECMAScript utiliza una terminología y una sintaxis que puede ser desconocido para un programador de JavaScript. Aunque la descripción de la lengua puede ser diferente en ECMAScript, el lenguaje en sí sigue siendo el mismo. JavaScript soporta toda la funcionalidad se describe en la especificación ECMAScript.

La documentación de JavaScript describe aspectos de la lengua que son apropiados para un programador de JavaScript. Por ejemplo:

* El objeto global no se discute en la documentación de JavaScript, ya que no se utiliza directamente. Los métodos y propiedades del objeto global, lo que usted hace uso, se analizan en la documentación de JavaScript pero se llaman funciones y propiedades de nivel superior.
* El no parámetro (sin argumentos) junto con los objetos Number y String no se discute en la documentación de JavaScript, ya que lo que se genera es de poca utilidad. Un Number sin un argumento devuelve 0, y un String sin argumentos devuelve "" (una cadena vacía).

# **Programación orientada a objetos**

Los objetos son entidades que tienen un determinado *estado*, *comportamiento (método)* e *identidad*:

* El *estado* está compuesto de datos o informaciones; serán uno o varios atributos a los que se habrán asignado unos valores concretos (datos).
* El comportamiento está definido por los [métodos](http://es.wikipedia.org/wiki/M%C3%A9todo_%28inform%C3%A1tica%29) o mensajes a los que sabe responder dicho objeto, es decir, qué operaciones se pueden realizar con él.
* La identidad es una propiedad de un objeto que lo diferencia del resto; dicho con otras palabras, es su identificador (concepto análogo al de identificador de una [variable](http://es.wikipedia.org/wiki/Variable_%28programaci%C3%B3n%29) o una [constante](http://es.wikipedia.org/wiki/Constante_%28programaci%C3%B3n%29)).

Un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos.

* Métodos: los objetos disponen de mecanismos de interacción, que favorecen la comunicación entre ellos. Esta comunicación favorece a su vez el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separa el estado y el comportamiento.
* Propiedad de conjunto: los métodos (comportamiento) y atributos (estado) están estrechamente relacionados por la propiedad de conjunto. Esta propiedad destaca que una clase requiere de métodos para poder tratar los atributos con los que cuenta.

El [programador](http://es.wikipedia.org/wiki/Programador) debe pensar indistintamente en ambos conceptos, sin separar ni darle mayor importancia a alguno de ellos. Hacerlo podría producir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que manejen a las primeras por el otro. De esta manera se estaría realizando una [programación estructurada](http://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada) camuflada en un lenguaje de programación orientado a objetos.

Los programadores que emplean Programación Orientada a Objetos, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

**Origen**

Los conceptos de la programación orientada a objetos tienen origen en [Simula 67](http://es.wikipedia.org/wiki/Simula), un lenguaje diseñado para hacer simulaciones, creado por [Ole-Johan Dahl](http://es.wikipedia.org/wiki/Ole-Johan_Dahl) y [Kristen Nygaard](http://es.wikipedia.org/wiki/Kristen_Nygaard), del Centro de Cómputo Noruego en [Oslo](http://es.wikipedia.org/wiki/Oslo).La idea surgió al agrupar los diversos tipos de naves en diversas clases de objetos, siendo responsable cada clase de objetos de definir sus propios datos y comportamientos.

Fueron refinados más tarde en [Smalltalk](http://es.wikipedia.org/wiki/Smalltalk), desarrollado en Simula en [Xerox PARC](http://es.wikipedia.org/wiki/Xerox_PARC) (cuya primera versión fue escrita sobre [Basic](http://es.wikipedia.org/wiki/Basic)) pero diseñado para ser un sistema completamente dinámico en el cual los objetos se podrían crear y modificar "sobre la marcha" (en tiempo de ejecución) en lugar de tener un sistema basado en programas estáticos.

La programación orientada a objetos se fue convirtiendo en el estilo de programación dominante a mediados de los años ochenta, en gran parte debido a la influencia de [C++](http://es.wikipedia.org/wiki/C%2B%2B), una extensión del [lenguaje de programación C](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C).

El [Eiffel](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Eiffel) de Bertrand Meyer fue un temprano y moderadamente acertado lenguaje con esos objetivos, pero ahora ha sido esencialmente reemplazado por [Java](http://es.wikipedia.org/wiki/Java_%28lenguaje_de_programaci%C3%B3n%29), en gran parte debido a la aparición de [Internet](http://es.wikipedia.org/wiki/Internet) .

[PHP](http://es.wikipedia.org/wiki/PHP) en su versión 5 se ha modificado; soporta una orientación completa a objetos, cumpliendo todas las características propias de la orientación a objetos.

**Conceptos fundamentales**

* [**Clase**](http://es.wikipedia.org/wiki/Clase_%28inform%C3%A1tica%29): Definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ella.
* [**Herencia**](http://es.wikipedia.org/wiki/Herencia_%28inform%C3%A1tica%29): (Por ejemplo, herencia de la clase C a la clase D) es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables públicas declaradas en C. Los componentes registrados como "privados" (private) también se heredan, pero como no pertenecen a la clase, se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. Esto es así para mantener hegemónico el ideal de POO.
* [**Objeto**](http://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29): Instancia de una clase. Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los objetos reales del mundo que nos rodea, o con objetos internos del sistema (del programa). Es una instancia a una clase.
* [**Método**](http://es.wikipedia.org/wiki/M%C3%A9todo_%28programaci%C3%B3n_orientada_a_objetos%29): Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.
* **Evento:** Es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento la reacción que puede desencadenar un objeto; es decir, la acción que genera.
* **Atributos:** Características que tiene la clase.
* **Mensaje:** Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.
* **Propiedad o atributo:** Contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.
* **Estado interno:** Es una variable que se declara privada, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos). No es visible al programador que maneja una instancia de la clase.
* **Componentes de un objeto:** Atributos, identidad, relaciones y métodos.
* **Identificación de un objeto:** Un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

En comparación con un lenguaje imperativo, una "variable" no es más que un contenedor interno del atributo del objeto o de un estado interno, así como la "función" es un procedimiento interno del método del objeto.

# **Motor de renderizado**

Un **motor de renderizado** es [software](http://es.wikipedia.org/wiki/Software) que toma contenido marcado (como [HTML](http://es.wikipedia.org/wiki/HTML), [XML](http://es.wikipedia.org/wiki/XML), archivos de imágenes, etc.) e información de formateo (como [CSS](http://es.wikipedia.org/wiki/CSS), [XSL](http://es.wikipedia.org/wiki/XSL), etc.) y luego muestra el contenido ya formateado en la pantalla de aplicaciones.

Alcanzó un uso popular cuando el proyecto [Mozilla](http://es.wikipedia.org/wiki/Fundaci%C3%B3n_Mozilla) diseñó el motor de su navegador ([Gecko](http://es.wikipedia.org/wiki/Gecko_(motor_de_renderizado))) como un componente aparte del propio navegador, el motor de Mozilla era reutilizable por otros navegadores diferentes, y mucha gente se empezó a referir a Gecko como un "motor de renderizado" en sí, en lugar de como una parte del navegador.

Algunos de los motores de renderizado más notables son:

* [Gecko](http://es.wikipedia.org/wiki/Gecko_(motor_de_navegaci%C3%B3n)), utilizado en [Mozilla Suite](http://es.wikipedia.org/wiki/Mozilla_Application_Suite), y otros navegadores como [Galeon](http://es.wikipedia.org/wiki/Galeon).
* [Trident](http://es.wikipedia.org/wiki/Trident_(motor_de_navegaci%C3%B3n)), el motor de [Internet Explorer](http://es.wikipedia.org/wiki/Internet_Explorer) para [Windows](http://es.wikipedia.org/wiki/Microsoft_Windows).
* [KHTML](http://es.wikipedia.org/wiki/KHTML)/[WebCore](http://es.wikipedia.org/wiki/WebCore), el motor de [Konqueror](http://es.wikipedia.org/wiki/Konqueror). Antecesor del [WebKit](http://es.wikipedia.org/wiki/WebKit).
* [Presto](http://es.wikipedia.org/wiki/Presto_(motor_de_renderizado)), el antiguo motor de [Opera](http://es.wikipedia.org/wiki/Opera_(navegador)).
* [Tasman](http://es.wikipedia.org/wiki/Tasman), el motor de [Internet Explorer para Mac](http://es.wikipedia.org/wiki/Internet_Explorer_para_Mac).
* [gzilla](http://es.wikipedia.org/wiki/Gzilla), el motor de [Dillo](http://es.wikipedia.org/wiki/Dillo).
* [GtkHTML](http://es.wikipedia.org/w/index.php?title=GtkHTML&action=edit&redlink=1), el motor de [Links](http://es.wikipedia.org/wiki/Links).
* [WebKit](http://es.wikipedia.org/wiki/WebKit), el motor de [Epiphany](http://es.wikipedia.org/wiki/Epiphany), [Safari](http://es.wikipedia.org/wiki/Safari).
* [Blink](http://es.wikipedia.org/wiki/Blink), el nuevo motor de [Google Chrome](http://es.wikipedia.org/wiki/Google_Chrome) y [Opera](http://es.wikipedia.org/wiki/Opera_(navegador)) (se trata de un [fork](http://es.wikipedia.org/wiki/Fork) de WebKit).
* [Servo](http://es.wikipedia.org/wiki/Servo_(motor_de_renderizado)), nuevo motor en desarrollo por parte de Mozilla (con el apoyo de Samsung), está siendo optimizado para la arquitectura ARM y la plataforma Android.

El motor hace la mayoría del trabajo. En esencia, toma una URL y un conjunto de ventana de contenido del área de rectángulo coordina como argumentos. A continuación, recuperar el documento correspondiente a la URL y pinta una representación gráfica de ello en el rectángulo dado. También maneja enlaces, formularios, cookies, scripting del lado del cliente, plug-in de carga, y otros asuntos.

La aplicación host proporciona la barra de menú, barra de direcciones, barra de estado, administrador de marcadores, historial y preferencias funcionalidad (entre otras cosas). Se incrusta el motor y sirve como una interfaz entre el usuario, el motor, y el sistema operativo subyacente.

**V8 (motor de JavaScript)**

Es un motor de JavaScript de código abierto desarrollado por Google para el navegador web Google Chrome.

V8 compila JavaScript para el código nativo de la máquina antes de ejecutarlo, en lugar de las técnicas más tradicionales, como la interpretación de código de bytes o la compilación de todo el programa en código máquina y ejecutarlo desde un sistema de archivos. El código compilado se optimiza adicionalmente (y re-optimizado) dinámicamente en tiempo de ejecución, basado en la heurística del perfil de ejecución de código. Las técnicas de optimización utilizadas incluyen procesos en línea, elisión de costosas propiedades de ejecución, y el almacenamiento en caché en línea, entre muchos otros.

Implementa [ECMAScript](http://es.wikipedia.org/wiki/ECMAScript) como especifica [ECMA-262](http://es.wikipedia.org/w/index.php?title=ECMA-262&action=edit&redlink=1) 5.ª edición y corre en [Windows XP](http://es.wikipedia.org/wiki/Windows_XP), [Vista](http://es.wikipedia.org/wiki/Windows_Vista), [Mac OS X 10.5](http://es.wikipedia.org/wiki/Mac_OS_X)(Leopard) y [Linux](http://es.wikipedia.org/wiki/Linux) en procesadores [IA-32](http://es.wikipedia.org/wiki/IA-32) y [ARM](http://es.wikipedia.org/wiki/Arquitectura_ARM).

V8 está diseñado para ser utilizado tanto en un navegador (sobre todo en Chrome y Chromium navegadores) y como un motor de alto rendimiento independiente que se puede integrar en los proyectos independientes, por ejemplo del lado del servidor JavaScript en Node.js, [10] o del lado del cliente JavaScript en .NET / Mono usando V8.NET.

**Programación del lado del cliente y del lado del servidor**

La comunicación entre dos (2) partes, a través del protocolo HTTP:

**↳ El servidor** - Esta parte es responsable de servir páginas.

**↳ El cliente** - Esta parte solicita páginas desde el servidor, y los muestra al usuario.

En la mayoría de los casos, el cliente es un navegador web.

**↳El Usuario** - El usuario utiliza el cliente con el fin de navegar por la web, rellenar formularios, ver videos en línea, etc.

La programación de cada lado, se refiere al código que se ejecuta en una máquina específica, del servidor o del cliente.

**Ejemplo básico**

El usuario abre su navegador web (el cliente) y navega a http://google.com.

|  |  |  |
| --- | --- | --- |
| * El cliente (en el nombre del usuario), envía una petición a http://google.com (el servidor), por su página de inicio. | → | El servidor reconoce la solicitud y responde al cliente con algunos meta-datos (calledheaders), seguido de la fuente de la página. |
| * El cliente recibe entonces la fuente de la página, y lo hace en un sitio web visible. | → | El usuario escribe “Stack Overflow” en la barra de búsqueda y presiona Buscar. |
| * El cliente envía los datos al servidor. | → | El servidor procesa esos datos, y responde con una página de búsqueda de los resultados de búsqueda. |
| * El Cliente, una vez más, hace que esa página sea visible para el l usuario. | | |

|  |  |
| --- | --- |
| **Programación del lado del Servidor** | **Programación del lado del Servidor** |
| **Usos**   * Proceso de entrada del usuario. * Mostrar Páginas. * Estructuras de aplicaciones web. * Interactuar con base de datos (SQL, archivos). | **Usos**   * Páginas web interactivas. * Hace posible acciones dinámicas en la página. * Interactuar con el almacenamiento temporal y el almacenamiento local. (Cookies) * Enviar solicitudes al servidor, y recuperar datos de él. * Proporcionar un servicio remoto para aplicaciones del lado del cliente, como el registro de software, la entrega de contenido, o juegos con varios jugadores a distancia. |
| **Ejemplo de lenguajes**   * PHP * ASP.Net en C #, C ++ o Visual Basic. * Casi cualquier idioma (C ++, C #, Java). | **Ejemplo de lenguajes**   * JavaScript (principalmente) * HTML * CSS |

### ¿Que es scripting language?

Un lenguaje de programación o lenguaje de script es un lenguaje de programación que soporta scripts, programas escritos para un ambiente especial en un determinado tiempo de ejecución que puede interpretar (en lugar de compilar) y automatizar la ejecución de tareas que, alternativamente, podría ser ejecutado una por una, por un operador humano .

Los ambientes que se pueden automatizar mediante scripts incluyen aplicaciones de software, páginas web dentro de un navegador web y sistemas embebidos. Un lenguaje de script puede ser visto como un lenguaje de dominio específico para un entorno particular; en el caso de secuencias de comandos de una aplicación, esto se conoce como un lenguaje de extensión. Los lenguajes de script también se refieren a veces lenguajes de programación como de muy alto nivel, ya que operan en un alto nivel de abstracción, o como lenguajes de control, en particular para lenguajes de control de trabajo en mainframes

Lengua de uso general dinámico de alto nivel, como Perl, [1] Tcl y Python, [2] con el término "script" a menudo se utiliza para programas pequeños (hasta a unos pocos miles de líneas de código) en estas lenguas, o en lenguajes específicos de dominio, como las lenguas de procesamiento de textos sed y AWK. Algunas de estas lenguas fueron desarrollados originalmente para su uso dentro de un entorno particular, y más tarde se convirtió en lenguajes específicos de dominio o de uso general portátiles.

Python es un lenguaje de propósito general que también se usa comúnmente como un lenguaje de extensión, mientras que ECMAScript sigue siendo principalmente un lenguaje de scripts para los navegadores web.

Por lo general un lenguaje de script se interpreta a partir de código fuente o de código de bytes, por el contrario en el software los script están escritos en un lenguaje compilado y distribuido en forma de código máquina. Los lenguajes de scripting pueden ser diseñados para su uso por los usuarios finales de un programa - usuario final - o pueden ser sólo para uso interno de los desarrolladores, para que puedan escribir partes del programa en el lenguaje de scripting.

Lenguaje glue es un programa de lenguajes (por lo general un lenguaje de script interpretado). que fue diseñado para conectarse con los componentes del software.

El desarrollo web puede considerarse como un uso de las lenguas glueque unen, la interfaz entre una base de datos y el servidor web..

**Lista de los motores de ECMAScript**

Un motor de ECMAScript es un programa que ejecuta el código fuente escrito en una versión de la norma del lenguaje ECMAScript, por ejemplo, JavaScript.

* Carakan: Un motor de JavaScript desarrollado por Opera Software ASA,, hasta el cambio a V8 con Opera 15.
* Chakra: Un motor de JScript utilizado en Internet Explorer
* SpiderMonkey: Un motor de JavaScript en aplicaciones de Mozilla Gecko, incluyendo Firefox.
* SquirrelFish: El motor JavaScript de WebKit de Apple Inc.. También conocido como Nitro
* V8: Un motor de JavaScript se utiliza en Google Chrome, Node.js y V8.NET
* JavaScriptCore: Un intérprete de JavaScript derivado originalmente de RV. Se utiliza en el proyecto WebKit y aplicaciones como Safari.
* Nashorn: Un motor de JavaScript se utiliza en Oracle Java Development Kit (JDK).
* Tamarin: Un motor de ActionScript y ECMAScript usado en Adobe Flash.

**Resumen libro:**

**Javascript, las partes buenas.**

**CAPÍTULO 1.**

**\_ Partes buenas**

A veces los diseñadores de lenguajes cometen errores.La mayoría de los lenguajes de programación contienen partes buenas y partes malas.Descubrí que podría ser un mejor programador utilizando sólo las partes buenas y evitar la mala partes. Después de todo, ¿cómo se puede construir algo bueno de partes malas?

Cuando los Java™ applets fracaso, JavaScript se convirtió en el "Idioma de la Web" por defecto. La popularidad de JavaScript es casi completamente independiente de sus cualidades como un lenguaje de programación.

Al centrarse en sólo las partes buenas, podemos reducir el tiempo de aprendizaje, aumentar la robustez,y salvar algunos árboles.Tal vez el mayor beneficio de estudiar las partes buenas es que se puede evitar la necesidad desaprender las partes malas. Desaprender malos patrones es muy difícil. Es una tarea dolorosa que la mayoría de nosotros nos enfrentamos con extrema renuencia.

JavaScript es un lenguaje importante porque es el lenguaje del navegador web. Su

asociación con el navegador hace que sea uno de los lenguajes de programación más populares en el mundo. Lo sorprendente de JavaScript es que es posible conseguir el trabajo hecho con lenguaje sin saber mucho sobre él, o incluso saber mucho acerca de la programación.

Las muy buenas ideas incluyen funciones, mecanografía suelta, objetos dinámicos, y una notación literal objeto expresivo. Las malas ideas incluyen un modelo de programación basado en variables globales. JavaScript es un lenguaje de programación relajado escrito, por lo que los compiladores de JavaScript son incapaces de detectar errores de tipo.

JavaScript tiene una muy poderosa notación literal de objetos. Los objetos pueden ser creados simplemente haciendo una lista de sus componentes. Esta anotación fue la inspiración para JSON, el formato de intercambio de datos.

Una característica controversial de JavaScript es la herencias de prototipos. JavaScript tiene un sistema de objetos libres de clase en la que los objetos heredan las propiedades directamente de otros objetos. Este es realmente de gran alcance, pero es poco familiar para los programadores de formación clásica. Si intentan aplicar los patrones de diseño clásicos directamente a JavaScript, estarás frustrado, pero si aprendes a trabajar con la naturaleza prototípica de JavaScript, serán recompensados sus esfuerzos.

JavaScript depende de variables globales para la vinculación. Todas las variables de alto nivel de todas las unidades de compilación son lanzados juntos en un espacio de nombres común que se llama el objeto global. Esta es una mala cosa, porque las variables globales son malas, y en JavaScript son fundamentales. Afortunadamente, como veremos, JavaScript también nos da las herramientas para mitigar este problema.

Apéndice C describe una herramienta de programación llamado JSLint, un analizador de JavaScript que puede analizar un programa JavaScript e informar sobre las partes malas que contiene. JSLint proporciona un grado de rigor que generalmente se carecen los desarrolladores de JavaScript. Dar confianza de que sus programas contienen sólo las partes buenas. JavaScript es un lenguaje de muchos contrastes. Contiene muchos errores y bordes afilados, por lo que puede que se pregunte, "¿Por qué debo usar JavaScript?" Hay dos respuestas. la

primero es que usted no tiene una elección. La Web se ha convertido en una importante plataforma para el desarrollo de aplicaciones y JavaScript es el único idioma que se encuentra en todos los navegadores.La otra respuesta es que, a pesar de sus deficiencias, JavaScript es muy bueno.

**CAPÍTULO 2.**

**\_ Gramática**

El espacio en blanco es generalmente insignificante, pero aveces es necesario usar espacios en blanco para separar secuencias de caracteres que de otro modo se combinan en un solo token. Por ejemplo, en:

**var que = esto;**

El espacio entre var y “que” no se puede quitar, pero los otros espacios se puede quitar.

JavaScript ofrece dos formas para comentar, comentarios en bloque formados con / \* \* / y los comentarios de fin de línea comenzando con //. Los comentarios deben ser usados libremente para mejorar la legibilidad de los programas. En JavaScript, estas formas también pueden ocurrir en literales de expresiones regulares, por lo que bloquear comentarios no son seguros para comentar los bloques de código. Por ejemplo:

**/ \***

**var rm\_a = /a\*/.match(s);**

**\* /**

provoca un error de sintaxis. Por lo tanto, se recomienda que / \* \* / comentarios evitarse y comentarios // utilizarse en su lugar.

Un nombre (name) es una carta seguido opcionalmente por una o más letras, dígitos o guiones bajos. Un nombre no puede ser una de estas palabras reservadas:

|  |  |
| --- | --- |
| abstract  boolean break byte  case catch char class const continue  debugger default delete do double  space  tab  line  end  any character  except line end  any character  except \*and /  //  \*  /  \*/ | whitespace  Numbers | 7  else enum export extends  false final finally float for function  goto  if implements import in instanceof int interface  long  native new null  package private protected public  return  short static super switch synchronized  this throw throws transient true try typeof  var volatile void  while wit |

No está permitido nombrar una variable o parámetro con un reservado palabra. Peor aún, no se permite el uso de una palabra reservada como el nombre de una propiedad de un objeto en un objeto literal o después de un punto en un refinamiento.

Los nombres se utilizan para declaraciones, variables, parámetros, nombres de propiedades, operadores y etiquetas.

JavaScript tiene un solo tipo de número. Internamente, se representa como punto flotante de 64 bits, lo mismo que Java'sdouble. A diferencia de la mayoría de otros lenguajes de programación, no hay ningún tipo de entero por separado, por lo que 1 y 1.0 son el mismo valor. Esta es una conveniencia significativa porque los problemas de desbordamiento de enteros cortos se evitan completamente, y todo lo que necesita saber acerca de un número es que es un número. Se evita una gran clase de errores de tipo numérico.

Los números negativos se pueden formar utilizando el prefijo del operador.

El valor NaN es un valor numérico que es el resultado de una operación que no puede producir un resultado normal. NaN no es igual a cualquier valor, incluyendo sí mismo. Puede detectar NaN con la función es NaN (número).

Strings puede ser envuelto en comillas simples o dobles. Puede contener cero o más caracteres. El \ (barra invertida) es el carácter de escape. JavaScript se construyó en un momento en Unicode era un juego de caracteres de 16 bits, por lo que todos los caracteres de JavaScript son de 16 bits de ancho.

JavaScript no tiene un tipo de carácter. Para representar un carácter, hacer una cadena con un solo carácter en ella.

Las secuencias de escape permiten la inserción de caracteres en cadenas que normalmente no están permitidas, tales como barras invertidas, citas, y caracteres de control. El \ uconvention permite especificar los puntos de código de carácter numérico.

**"A" === "\ u0041"**

Strings tienen una propiedad length. Por ejemplo, "siete" .length es 5.

Strings son inmutables. Una vez que está hecho, una cadena nunca se puede cambiar. Pero es fácil para hacer una nueva cadena concatenando otras cuerdas junto con el operador +.

Cuando se utiliza en el interior de una función, la sentencia var define variables privadas de la función. El switch, while, for, y do declaraciones se les permite tener un prefijo de etiqueta opcional que interactúa con la sentencia break.

Declaraciones tienden a ser ejecutados en orden de arriba a abajo. La secuencia de ejecución puede ser alterado por las sentencias condicionales (if y switch), por las declaraciones de bucle (while, for, y do), por las declaraciones disruptivas ((break, return, y throw), y por la invocación de la función.

Un bloque es un conjunto de sentencias envueltos entre llaves. A diferencia de muchos otros idiomas, bloques de JavaScript no crean un nuevo ámbito, por lo que las variables deben ser definidas en la parte superior de la función, no en bloques.

La sentencia if cambia el flujo del programa basado en el valor de la expresión. El entonces bloque se ejecuta si la expresión es verdadera; de lo contrario, se toma la rama else opcional.

Estos son los valores falsos:

• false

• null

• String vacío

• El Number 0

• El number NaN

Todos los demás valores son verdaderos, incluyendo cierto, la cadena false, y todos los objetos.

La sentencia switch realiza una rama de múltiples vías. En él se compara la expresión de igualdad con todos los casos especificados. La expresión puede producir un número o una cadena. Cuando se encuentra una coincidencia exacta, se ejecutan las sentencias de la cláusula de caso a juego. Si no hay ninguna coincidencia, las declaraciones por defecto opcionales se ejecutan.

Una case clause contiene una o más expresiones de caso. Las expresiones de caso no tienen que ser constantes. El comunicado tras una cláusula debe ser una declaración perjudicial para evitar la caída a través en el siguiente caso. La sentencia break se puede utilizar para salir de un interruptor.

La declaración while realiza un bucle simple. Si la expresión es falsa, a continuación, el bucle se romperá. Mientras que la expresión es verdadera, se ejecutará el bloque.

La sentencia for es una declaración de bucle más complicado. Viene en dos forma convencional forms.Controlado por tres cláusulas opcionales: la inicialización, el estado, y el incremento. En primer lugar, la inicialización se lleva a cabo, el cual típicamente se inicializa la variable de bucle. A continuación, se evalúa la condición. Normalmente, esta prueba la variable de bucle en contra de un criterio de terminación. Si la condición se omite, se asume entonces que condición es verdadera. Si la condición es falsa, los saltos de bucle. De lo contrario, se ejecuta el bloque, entonces el incremento se ejecuta, y luego el bucle se repite con la condición

La otra forma (llamada for in) enumera los nombres de propiedad (o keys) de un objeto. En cada iteración, otra cadena de nombre de la propiedad del objeto se asigna a la variable.

Por lo general, es necesario probar Object.hasOwnProperty (variable) para determinar si el nombre de la propiedad es realmente un miembro del objeto o se encontró en cambio en la cadena de prototipo.

**for (myvar in obj) {**

**if (obj.hasOwnProperty(myvar)) {**

**…**

**}**

**}**

La instrucción do es como la instrucción while excepto que la expresión se prueba después de que el bloque se ejecuta en lugar de antes. Eso significa que el bloque siempre se ejecutará al menos una vez.

La sentencia try ejecuta un bloque y las capturas de las excepciones que fueron arrojados por el bloque. La cláusula catch define una nueva variable que recibirá el objeto de excepción.

La sentencia throw lanza una excepción. Si la sentencia throw está en un bloque try, el control continúa en la cláusula catch. De lo contrario, la invocación de la función es abandonado, y el control pasa a la cláusula catch de la prueba en la función que llama.

La expresión suele ser un literal objeto que contiene una propiedad de nombre y una propiedad de mensaje. El receptor de la excepción pueden utilizar esa información para determinar qué hacer.

La sentencia return hace que el pronto retorno de una función. También puede especificar el valor a devolver. Si no se especifica una expresión de retorno, entonces el valor de retorno será indefinido. JavaScript no permite que un extremo de la línea entre return y la expresión.

La sentencia break hace que la salida de una sentencia de bucle o sentencia switch. Opcionalmente puede tener una etiqueta que provocará una salida de la sentencia etiquetada. JavaScript no permite que un extremo de la línea entre el break y la etiqueta.

Una declaración de la expresión puede asignar cualquiera de los valores de una o más variables o miembros, invocar un método, eliminar una propiedad de un objeto. El operador = se utiliza para la asignación. No lo confunda con el operador de igualdad ===. El operador = se puede agregar o concatenar.

Las expresiones más simples son un valor literal (como una cadena o un número), una variable, un valor integrado (true, false, null, indefinido, NaN o infinito), una expresión invocación precedido de nuevo, una expresión de refinamiento precedido por la cancelación, una expresión envuelto entre paréntesis, una expresión precedida por un operador de prefijo, o una expresión seguida de:

• Un operador infijo y otra expresión

• El ? operador ternario seguido por otra expresión, entonces por : , y luego por otra expresión

• Una invocación

• Un refinamiento

El ? operador ternario tres operandos. Si el primer operando es verdadero, produce el valor del segundo operando. Pero si el primer operando es falso, produce el valor de la tercera operando.

Los operadores en la parte superior de la lista de prioridad de los operadores en la Tabla 1.2 tienen mayor precedencia. Se unen el más apretado. Los operadores en el fondo tienen la prioridad más baja. Los paréntesis se pueden utilizar para alterar la prioridad normal, así:

**2 + 3 \* 5 === 17**

**(2 + 3) \* 5 === 25**

|  |  |
| --- | --- |
| . [] ( ) | El refinamiento y la invocación borrar |
| delete new typeof + - ! | operadores unarios |
| \* / % | multiplicación, división, módulo |
| + - | Suma / concatenación, resta |
| >= <= > < | desigualdad |
| === !== | igualdad |
| && | lógico and |
| || | lógico or |
| ?: | ternario |

Los valores producidos por typeof es 'número', 'cadena',, 'indefinido', 'función' 'booleano' y 'objeto'. Si el operando es una array o null, el resultado es "objeto", lo cual es incorrecto.

Si el operando de ! es verdadero , produce falso. De lo contrario, se produce cierto. El operador + añade o concatenada. Si usted quiere que añadir, asegúrese de que ambos operandos son números.

El operador / puede producir un resultado no entero, incluso si ambos operandos son enteros.

El operador && produce el valor de su primer operando si el primer operando es falso. De lo contrario, se produce el valor del segundo operando.

El operador || produce el valor de su primer operando del primer operando es verdadero. De lo contrario, se produce el valor del segundo operando.

Invocation provoca la ejecución de un valor de la función. El operador de invocación es un par de paréntesis que siguen el valor de la función. Los paréntesis pueden contener argumentos que serán entregados a la función.

Los object literals son una notación conveniente para especificar los objetos nuevos. Los nombres de las propiedades se pueden especificar como nombres o como cadenas. Los nombres se tratan como nombres literales, no como nombres de variables, por lo que los nombres de las propiedades del objeto deben ser conocidos en tiempo de compilación. Los valores de las propiedades son expresiones.

Function literal define un valor de función. Puede tener un nombre opcional que puede usar para llamar a sí mismo de forma recursiva. Puede especificar una lista de parámetros que actuarán como variables inicializadas por los argumentos de invocación. El cuerpo de la función incluye definiciones y declaraciones de variables.