Resumen JavaScript

- JavaScript Errors - Throw and Try to Catch
- Capítulo 5

Herencia.
- Pseudoclassical
- Object Specifiers
- Prototypal
- Funcional
- Regiones

Bad parts
- Continuar Declaración
- Cambiar Caen A Través
- Declaraciones Bloque-less

Buenas Prácticas

- Evitar usar variables globales.
- Declarar siempre variables locales.
- Declarar las variables de primero.
- No declarar nunca números, strings o booleanos como objetos.
- No usar el new Object(), usar solo { } o “ ” …
- Tenga cuidado de que no se le cambie el tipo de variable, cuando le cambia el valor.
- Usar === en vez de ==.
- Asignar valores al parámetro por defecto.
- Evite el uso de eval()

Intro POO.

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 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 o una constante).

Conceptos fundamentales

La programación orientada a objetos es una forma de programar que trata de encontrar una solución a estos problemas. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

- ***Clase***: Definiciones de las propiedades y comportamiento de un tipo de objeto concreto.
- ***Herencia***: (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
- ***Objeto***: Instancia de una clase
- ***Método***: Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje".
- ***Evento*****:** Es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto).
- ***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
- ***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.

Constructor

En programación orientada a objetos (POO), un constructor es una subrutina cuya misión es inicializar un objeto de una clase. En el constructor se asignan los valores iniciales del nuevo objeto.

Qué es JavaScript?

JavaScript es un lenguaje de scripting multiplataforma, orientado a objetos. JavaScript es un pequeño y liviano lenguaje; no es útil como lenguaje independiente, pero esta diseñado para ser fácilmente embebido en otros productos y aplicaciones, como ser web browsers.

JavaScript y Java

Javascript y Java son similares en algunos puntos, pero fundamentalmente diferentes en otros. El lenguaje Javascript se parece al de Java pero no tiene el tipado estático y comprobación de tipos fuerte. Javascript 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.

Investigaciones

- ***LowerCase:*** Convierte un String a mayúsculas, pero sin cambiar el String original.
- ***Eval():*** Evalúa una cadena de código JavaScript sin referenciar a un objeto en particular. Su sintaxis es eval(*cadena* [, *objeto*]) y recibe por parámetros una cadena o un objeto.
- ***Prompt***: Sirve para la entrada de datos por teclad. Cada vez que necesitamos ingresar un dato con esta función, aparece una ventana donde cargamos el valor.

Palabras Reservadas

- ***Break***: Interrumpe un bloque de instrucciones saltando a la primera instrucción que sigue al bloque que contiene el "break". Un uso apropiado evitará la formación de loop sin salida.
- ***Case*** – Switch: Nos permite evaluar el valor de una expresión para tomar una decisión en el flujo del ejecución. Es muy parecida a la estructura de control i**f/else** pero con **switch/case** podemos evaluar más de dos casos.
- ***Const***: Constantes en JavaScript.
- ***Continue***: Indica que se continue un bloque de instrucciones, pero interrumpiendo la iteración en ese punto y volviendo a comenzar desde el inicio del bloque.
- ***Do*** – ***While***: Crea un bucle que ejecuta una sentencia especificada, hasta que la condición de comprobación se evalúa como falsa. La condición se evalúa después de ejecutar la sentencia, dando como resultado que la sentencia especificada se ejecute al menos una vez.
- ***Export***: Permite a un script firmado proporcionar propiedades, funciones, y objetos a otro script firmado o no.
- ***For***: Crea un bucle que consiste en tres expresiones opcionales, encerradas en paréntesis y separadas por puntos y comas, seguidas de una sentencia ejecutada en un bucle.
- ***Function***: Declara una función con los parámetros especificados. Puede también definir funciones usando el constructor Function y el function (expresión function).
- ***If – Else:*** Ejecuta una sentencia si una condición específicada es evaluada como verdadera. Si la condición es evaluada como falsa, otra sentencia puede ser ejecutada.
- ***Instanceof***: Devuelve verdadero si el objeto especificado es del tipo especificado.
- ***Let***: Controla el ámbito de una variable y controlar mejor su propagación.
- ***New***: Crea una instancia de un tipo de objeto a partir de una función constructora nativa ó definida por el usuario.
- ***Return***: Especifica el valor devuelto por una función.

- ***This***: Su valor está determinado por cómo se llama a la función. No puede ser establecida por una asignación en tiempo de ejecución, y esto puede ser diferente cada vez que la función es llamada.
- ***Throw***: Lanza una excepción definida por el usuario.
- ***Try – Catch – Finally:*** Con try especificamos una serie de sentencias Javascript que vamos a tratar de ejecutar. Con catch especificamos lo que queremos realizar si es que se ha cazado un error en el bloque try. Finally se ejecuta después de try y catch. Se ejecuta siempre.
- ***Typeof***: Devuelve una cadena que indica el tipo de variable, cadena, palabra clave u objeto sin evaluarlo.
- ***Var***: Declaración de una variable, opcionalmente inicializada a un valor.
- ***Void***: Específica una expresión que se evalúa sin devolver un valor.
- ***Yield***: Se utiliza para hacer una pausa y reanudar una función generadora

Motor de renderizado

- Es software que toma contenido marcado e información de formateo y luego muestra el contenido ya formateado en la pantalla de aplicaciones.
- Se usan típicamente en navegadores web, clientes de correo electrónico.
- Todos los navegadores web incluyen necesariamente algún tipo de motor de renderizado.

Algunos de los motores de renderizado más notables son:

- Gecko, utilizado en Mozilla Suite
- Trident, el motor de Internet Explorer para Windows.
- Tasman, el motor de Internet Explorer para Mac.
- WebKit, el motor de Epiphany, Safari.
- Blink, el nuevo motor de Google Chrome y Opera

V8 (motor JavaScript)

- Es un motor de código abierto para JavaScript creado por Google.
- Está escrito en C++.
- Es usado en Google Chrome.
- Está integrado en el navegador de internet del sistema operativo Android 2.2 “Froyo”.

Programación del lado del cliente y del lado del servidor

El desarrollo web es todo acerca de la comunicación. En este caso, la comunicación entre 2 partes, a través del protocolo HTTP:

- Servidor: Provee el servicio

- Cliente: Solicita el servicio.
- El Usuario: Utiliza el cliente con el fin de navegar por la web, rellenar formularios, ver videos en línea, etc.

Proceso de entrada del usuario.
Páginas de pantalla.
Aplicaciones web Estructura.
Interactuar con el almacenamiento permanente (SQL, archivos).

Ejemplo de lenguajes:
- PHP
- ASP.Net en C #, C ++ o Visual Basic.

Casi cualquier idioma (C ++, C #, Java). Estos no fueron diseñados específicamente para la tarea, pero ahora se utilizan a menudo para los servicios web a nivel de aplicación.
La programación del lado del cliente
Al igual que el del lado del servidor, la programación del lado del cliente es el nombre de todos los programas que se ejecutan en el cliente.

Usos
Hacer páginas web interactivas.
Haga cosas suceda de forma dinámica en la página web.
Interactuar con el almacenamiento temporal y el almacenamiento local (cookies, localStorage).
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 :
- JavaScript
- HTML *
- CSS *

Cualquier lenguaje que se ejecuta en un dispositivo cliente que interactúa con un servicio remoto es un lenguaje del lado del cliente.* HTML y CSS no son realmente "lenguajes de programación"

Lenguaje Script

Un lenguaje de programación o lenguaje de script es un lenguaje de programación que soporta scripts, programas escritos para un ambiente especial en tiempo de ejecución que

puede interpretar (en lugar de compilar) y automatizar la ejecución de tareas que, alternativamente, podría ser ejecutado de una en una por un ser humano operador.

Lista de 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
Chakra: Un motor de JScript utilizarse en Internet Explorer.
SpiderMonkey: Un motor de JavaScript en aplicaciones de Mozilla Geck
SquirrelFish: El motor JavaScript de WebKit de Apple Inc. También conocido como Nitro.
Tamarin: Un motor de ActionScript y ECMAScript usado en Adobe Flash.
V8: Un motor de JavaScript se utiliza en Google Chrome.
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).

Investigaciones

- Diferencia entre Window.Onload y Document.Ready

El Document.Ready ocurre después que el HTML ha sido cargado, mientras que el Window.Onload ocurre cuando el contenido (por ejemplo las imágenes) han sido cargadas. Document.Ready es un evento específico de Jquery, Window.Onload es un evento estándar del DOM.

- Diferencia entre Window.Onload y Window.Onunload

Onload es aquel que se produce cuando un navegador carga un documento HTML o una imagen.
Onunload tiene como misión ejecutar un script cuando la página web actual se descarga, ya sea porque se accede a otra página o porque se pulsan los botones de retroceder y avanzar.

- ¿Para qué sirve NoScript?

Muestra un mensaje al usuario cuando su navegador no puede ejecutar JavaScript.

- Diferencia entre Null y Undefined

- Undefined: para Javascript, **no existe**. O bien no ha sido declarada o jamás se le asignó un valor.
- Null: para Javascript, **la variable existe**. En algún momento, explícitamente, la variable se estableció a null.

La precedencia de operadores

La precedencia de operadores determina el orden en que se evalúan los operadores. Los operadores con mayor precedencia se evalúan primero.

Asociatividad (ARREGLAR!)

La asociatividad determina el orden en que se procesan los operadores con la misma precedencia. Por ejemplo, considere una expresión:
una OP OP b c
Asociatividad por la izquierda (de izquierda a derecha) significa que se procesa como (a OP b) c OP, mientras asociatividad por la derecha (de derecha a izquierda) significa que se interpreta como una OP (b OP c). Operadores de asignación son asociativo por la derecha, por lo que puede escribir:
a = b = 5;
con el resultado esperado que ayb obtener el valor 5. Esto se debe a que el operador de asignación devuelve el valor que se le asigna. En primer lugar, b se establece en 5. A continuación, la una se establece en el valor de b.

Capítulo 1

Porque JavaScript?

Hay 2 respuestas por las cuales se usa JS. La primera es que no tenemos otra opción ya que es el único lenguaje que se encuentra en todos los navegadores. Y la otra es que a pesar de sus deficiencias JS es realmente bueno. Es ligero y expresivo.

Converting Strings to Numbers

The global method **Number()** can convert strings to numbers.

Strings containing numbers (like "3.14") convert to numbers (like 3.14).

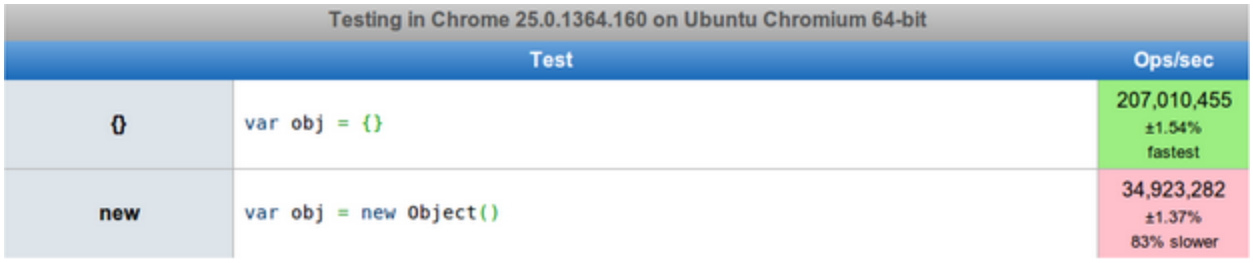
Empty strings convert to 0.

Anything else converts to NaN (Not a number).

```
Number("3.14")    // returns 3.14
Number(" ")       // returns 0
Number("")        // returns 0
Number("99 88")   // returns NaN
```

In the chapter Number Methods, you will find more methods that can be used to convert strings to numbers:

Method	Description
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer



Converting Dates to Numbers

The global method **Number()** can be used to convert dates to numbers.

```
d = new Date();
Number(d)          // returns 1404568027739
```

The date method **getTime()** does the same.

```
d = new Date();
d.getTime()        // returns 1404568027739
```

Automatic Type Conversion

When JavaScript tries to operate on a "wrong" data type, it will try to convert the value to a "right" type.

The result is not always what you expect:

```
5 + null    // returns 5         because null is converted to 0
"5" + null  // returns "5null"   because null is converted to "null"
"5" + 1     // returns "51"      because 1 is converted to "1"
"5" - 1     // returns 4         because "5" is converted to 5
```

String.prototype.indexOf()

El **`indexOf()`** método devuelve el índice, dentro del objeto `String` que realiza la llamada, de la primera ocurrencia del valor especificado, comenzando la búsqueda desde `indiceBusqueda`; o -1 si no se encuentra dicho valor.

Sintaxis:

```
cadena.indexOf(valorBuqueda[, indiceDesde])
```

Los caracteres de una cadena se indexan de izquierda a derecha. El índice del primer carácter es 0, y el índice del último carácter de una cadena llamada `nombreCadena` es `nombreCadena.length - 1`.

JavaScript Hoisting

Hoisting es el comportamiento predeterminado de JavaScript de las declaraciones de pasar a la parte superior.

JavaScript Declarations are Hoisted

En JavaScript, una variable puede ser declarado después de que se ha utilizado.

En otras palabras; una variable puede ser utilizado antes de que haya sido declarada.

JavaScript Use Strict

No es una declaración, sino una expresión literal, ignorado por las versiones anteriores de JavaScript.

El propósito de "use strict" es para indicar que el código debe ser ejecutado en el "modo estricto".

Con el modo estricto, no se puede, por ejemplo, utilizar variables no declaradas.

Declarando Modo estricto

El modo estricto se declara mediante la adición de "uso estricto"; al principio de un archivo JavaScript, o una función de JavaScript.

Declarado en el comienzo de un archivo JavaScript, tiene un alcance global (todo el código se ejecutará en modo estricto).

Declarado dentro de una función, tiene alcance local (sólo el código dentro de la función es en modo estricto).

Declaración global:

```
"use strict";
x = 3.14;       // This will cause an error
myFunction();   // This will also cause an error

function myFunction() {
      x = 3.14;
}
```

Declaración local:

```
x = 3.14;       // This will not cause an error.
myFunction();   // This will cause an error

function myFunction() {
   "use strict";
      x = 3.14;
}
```

Errores comunes de JavaScript

Accidentalmente Usando el operador de asignación
Programas JavaScript puede generar resultados inesperados si un programador utiliza accidentalmente un operador de asignación (=), en lugar de un operador de comparación (==) en una sentencia if.

Esta sentencia if devuelve false (como se esperaba), ya que x no es igual a 10:

```
var x = 0;
if (x == 10)
```

```
Esta sentencia if devuelve true (quizás no tan esperado), porque 10 es
verdadero:
var x = 0;
if (x = 10)
```

```
Esta sentencia if devuelve false (quizás no tan esperado), porque 0 es
falsa:
var x = 0;
if (x = 0)
```

Guía de estilo JavaScript y convenciones de codificación

Convenciones de codificación de JavaScript
Las convenciones de codificación son las directrices de estilo de programación. Por lo general se refieren a:
Reglas de nomenclatura y de declaración de variables y funciones.Reglas para el uso de espacios en blanco, la sangría, y los comentarios.
Programación de las prácticas y los principios
Las convenciones de codificación de calidad segura:

Mejora la legibilidad del código
Hacer el mantenimiento del código más fácil
Las convenciones de codificación se pueden documentar las reglas para los equipos a seguir, o simplemente ser su práctica de codificación individual.

Nombres de variables

En W3schools utilizamos camelCase de nombres de identificadores (variables y funciones).
Todos los nombres comienzan con una letra.
En la parte inferior de esta página, usted encontrará una discusión más amplia sobre las reglas de nombres.

```
firstName = "John";
lastName = "Doe";

price = 19.90;
tax = 0.20;

fullPrice = price + (price * tax);
```

Espacios alrededor de Operadores

Siempre ponga espacios alrededor de los operadores (= + / *), y después de las comas:

```
var x = y + z;
var values = ["Volvo", "Saab", "Fiat"];
```

JavaScript JSON

JSON es un formato para almacenar y transportar datos.

JSON se utiliza a menudo cuando se envían datos desde un servidor a una página web.

¿Qué es JSON?
JSON es sinónimo de JavaScript Object Notation
JSON es un formato de intercambio de datos ligero
JSON es independiente del idioma *
JSON es "auto-descripción" y fácil de entender

JSON utiliza la sintaxis de JavaScript, pero el formato JSON es sólo texto.

El texto puede ser leído y utilizado como un formato de datos por cualquier lenguaje de programación.

Ejemplo JSON

```
{"employees":[
      {"firstName":"John", "lastName":"Doe"},
      {"firstName":"Anna",    "lastName":"Smith"},
      {"firstName":"Peter", "lastName":"Jones"}
]}
```

Desempeño JavaScript

Cómo acelerar su código JavaScript.

Reducir la actividad en bucles
Loops se utilizan a menudo en la programación.

Cada declaración dentro de un bucle se ejecutará para cada iteración del bucle.

Búsqueda de declaraciones o asignaciones que se pueden colocar fuera del bucle.

Reducir DOM Acceso
Acceso al DOM HTML es muy lento, en comparación con otras sentencias de JavaScript.

Si usted espera para acceder a un elemento DOM varias veces, acceder a él una vez, y lo utilizan como una variable local:

ejemplo

obj = document.getElementById ("demo");
obj.innerHTML = "Hola";

Reducir DOM Tamaño

Mantenga el número de elementos en el DOM HTML pequeño.

Esto siempre mejorará carga de la página, y la velocidad de renderizado (visualización de la página), sobre todo en los dispositivos más pequeños.

Todo intento de buscar el DOM (como getElementsByTagName) se beneficiará de un DOM menor.

Evite variables innecesarias
No crear nuevas variables si no planea guardar valores.

A menudo se puede reemplazar código como este:

fullName var = name + "" + lastName;
document.getElementById ("demo") innerHTML = fullName.;
Con este:

document.getElementById ("demo"). innerHTML = Nombre + "" + lastName
Delay JavaScript Cargando
Poner sus guiones en la parte inferior del cuerpo de la página, permite que el navegador carga la página por primera vez.

Mientras que un script se está descargando, el navegador no se iniciará ningún otro descargas. Además toda la actividad de análisis y representación podría ser bloqueado.

Apéndice A

Variables globales

La peor de todas las malas características de JavaScript es su dependencia de las variables globales.

Hay tres maneras de definir las variables globales. La primera es la de colocar una declaración var fuera de cualquier función:

```
var foo = valor;
```

El segundo consiste en añadir una propiedad directamente al objeto global. El objetivo global es el contenedor de todas las variables globales. En los navegadores web, el objeto global va por la ventana del nombre:

```
window.foo = valor;
```

La tercera es usar una variable sin declararlo. Esto se llama implícita globales:

```
foo = valor;
```

Esto fue pensada como una conveniencia para los principiantes por lo que es necesario declarar las variables antes de usarlas. Desafortunadamente, olvidando que declarar una variable es un error muy común. La política de JavaScript de hacer las variables olvidadas global crea errores que pueden ser muy difíciles de encontrar.

Scope

La sintaxis de JavaScript viene de C. En todos los demás lenguajes de programación como C, un bloque (un conjunto de estados envueltos entre llaves) crea un ámbito. Las variables declaradas en un bloque que no son visibles fuera del bloque.

Es mejor que declarar todas las variables en la parte superior de cada función.

Semicolon Insertion

JavaScript tiene un mecanismo que intenta corregir los programas defectuosos por punto y coma insertar automáticamente. No dependen de esto. Esto puede enmascarar los errores más graves. A veces inserta un punto y coma en lugares donde no son bienvenidos. Tenga en cuenta las consecuencias de la inserción punto y coma en la sentencia return. Si una sentencia return devuelve un valor, que la expresión de valor debe comenzar en la misma línea que el retorno:

```
return { status: true };
```

Esto parece devolver un objeto que contiene un miembro de estado. Por desgracia, la inserción y coma lo convierte en una sentencia que devuelve indefinido. No hay ninguna advertencia que punto y coma inserción causó la mala interpretación del programa. El problema se puede evitar si el {se coloca al final de la línea anterior y no al principio de la línea siguiente:

```
return { status: true }
```

Palabras reservadas

Las siguientes palabras están reservadas en JavaScript:

- abstract
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- debugger
- default
- delete
- do
- double
- 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
- with

La mayoría de estas palabras no se usan en el lenguaje. No pueden ser utilizados para las variables o parámetros de nombres. Cuando las palabras reservadas se utilizan como claves en objetos literales, deben ser citados. Ellos no se pueden utilizar con la notación de puntos, por lo que es necesario a veces utilizar la notación de soporte en su lugar:

```
var method;              // ok
var class;               // illegal
object = {box: value};     // ok
object = {case: value};    // illegal
object = {'case': value};  // ok
object.box = value;        // ok
object.case = value;       // illegal
object['case'] = value;    // ok
```

Unicode

JavaScript fue diseñado en un momento en que se espera Unicode tener como máximo 65.536 caracteres. Desde entonces ha crecido para tener una capacidad de más de 1 millón de caracteres. Los personajes de JavaScript son 16 bits. Eso es suficiente para cubrir el original 65536 (que ahora se conoce como el plano básico multilingüe). Cada uno de los millones de caracteres restantes se puede representar como un par de caracteres. Unicode considera el par a ser un único carácter. JavaScript piensa la pareja es de dos personajes distintos.

Typeof

El operador typeof devuelve una cadena que identifica el tipo de su operando. Por lo tanto:

```
typeof 98.6 produces 'number'.
```

Desafortunadamente:
typeof nulos devuelve 'objeto' en vez de 'nulo'. Ups. Una mejor prueba para nula es simplemente:

```
my_value === null
```

Un problema mayor es la prueba de un valor para objetualidad. typeof no pueden distinguir entre nulo y objetos, pero usted puede porque nula es falso y todos los objetos son verdaderos:

```
if (my_value && typeof my_value === 'object') {    // my_value is an object or an array!
}
```

ParseInt

ParseInt es una función que convierte una cadena en un número entero.

+

El operador + puede agregar o concatenar. Cuál lo hace depende de los tipos de los parámetros. Si alguno de los operandos es una cadena vacía, produce el otro operando convierte en una cadena. Si ambos operandos son números, que produce la suma. De lo contrario, convierte ambos operandos de cadenas y las concatena. Este comportamiento complicado es una fuente común de errores. Si tiene la intención + añadir, asegúrese de que ambos operandos son números.

Floating Point

Números de punto flotante binarios son ineptos en el manejo de las fracciones decimales, por lo que 0.1 + 0.2 no es igual a 0,3.

Afortunadamente, la aritmética de enteros en coma flotante es exacta, por lo que los errores de representación decimales se puede evitar mediante la ampliación. Por ejemplo, los valores en dólares pueden ser convertidos a valores enteros centavos multiplicándolos por 100. La gente tiene una expectativa razonable cuando se cuentan el dinero que los resultados serán exactos.

NaN

El valor NaN representa no es un número, a pesar de que:

```
typeof NaN === 'number'  // true
```

El valor se puede producir al intentar convertir una cadena en un número cuando la cadena no está en la forma de un número. Por ejemplo:

```
+ '0'       // 0
+ 'oops'    // NaN
```

Si es un operando en una operación aritmética, entonces NaN será el resultado. Por lo tanto, si usted tiene una cadena de fórmulas que producen NaN como resultado, al menos una de las entradas era NaN, NaN o fue generada en alguna parte. Puede probar NaN. Como hemos visto, typeof no distingue entre los números y NaN, y resulta que NaN no es igual a sí mismo. Así, sorprendentemente:

```
NaN === NaN    // false
NaN !== NaN    // true
```

Phony Arrays

JavaScript no tiene matrices reales. Eso no es del todo malo. Matrices de JavaScript son muy fácil de usar. No hay necesidad de darles una dimensión, y nunca generar errores outof-grada. Sin embargo, su rendimiento puede ser considerablemente peor que las matrices reales.

hasOwnProperty

El método hasOwnProperty se ofrece como un filtro para evitar un problema con el en el comunicado. Desafortunadamente, hasOwnProperty es un método, no un operador, por lo que en cualquier objeto que pueda ser reemplazado con una función diferente o incluso un valor que no es una función de:

```
var name;
another_stooge.hasOwnProperty = null;       // trouble
for (name in another_stooge) {
        if (another_stooge.hasOwnProperty(name)) { // boom
                document.writeln(name + ': ' + another_stooge[name]);
        }
}
```

Objeto

Objetos de JavaScript nunca son verdaderamente vacío, ya que pueden recoger a los miembros de la cadena de prototipo. A veces lo que importa. Por ejemplo, supongamos que usted está escribiendo un programa que cuenta el número de ocurrencias de cada palabra en un texto. Podemos utilizar el método toLowerCase para normalizar el texto a minúsculas, y luego usar el método split con una expresión regular para producir una serie de palabras.

Capítulo 2

Whitespace

El espacio en blanco puede tomar la forma de caracteres de formato o comentarios.

JavaScript ofrece dos formas de comentarios, comentarios en bloque formados con / * * / y los comentarios de fin de línea comenzando con //.

Name

Un nombre es una letra seguido opcionalmente por una o más letras, dígitos o guiones bajos. Un nombre no puede ser una palabra reservada.

Numbers

JavaScript tiene un solo tipo de número. Internamente, se representa como punto flotante de 64 bits, la misma que la de doble Java. 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.

Strings

Una cadena literal 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.

Statements

Una unidad de compilación contiene un conjunto de instrucciones ejecutables. En los navegadores web, cada etiqueta <script> entrega una unidad de compilación que se compila y ejecuta inmediatamente. Al carecer de un enlazador, JavaScript todas lanza juntos en un espacio de nombres global común.

Capítulo 3

Objects

Los literales de objetos

Los literales de objetos proporcionan una notación muy conveniente para la creación de nuevos valores de objeto. Un objeto literal es un par de llaves que rodean cero o más pares nombre / valor. Un literal objeto puede aparecer en cualquier lugar puede parecer una expresión:

```
var empty_object = {};
var stooge = {
        "first-name": "Jerome",
        "last-name": "Howard"
};
```

El nombre de una propiedad puede ser cualquier cadena, incluyendo la cadena vacía. Las comillas en el nombre de una propiedad de un objeto literal son opcionales si el nombre sería un nombre legal JavaScript y no una palabra reservada. Así que se requieren comillas alrededor de "nombre", pero son opcionales alrededor first_name. Las comas se utilizan para separar los pares.

Recuperación

Los valores se pueden recuperar de un objeto envolviendo una expresión de cadena en un [] sufijo. Si la expresión de cadena es una constante, y si es un nombre legal JavaScript y no una palabra reservada, entonces el. notación se puede utilizar en su lugar. El. se prefiere la notación porque es más compacto y se lee mejor:

Update

Un valor de un objeto puede ser actualizado por asignación. Si el nombre de la propiedad ya existe en el objeto, el valor de la propiedad se sustituye.

Reference

Objetos se pasan alrededor por referencia. Nunca se copian.

Prototype

Cada objeto está vinculado a un objeto prototipo de la que puede heredar propiedades. Todos los objetos creados a partir de objetos literales están vinculados a Object.prototype, un objeto que viene de serie con JavaScript. Cuando usted hace un nuevo objeto, puede seleccionar el objeto que debe ser su prototipo. El mecanismo que proporciona JavaScript de hacer esto es desordenado y complejo, pero se puede simplificar de manera significativa. Vamos a añadir un método create a la función del objeto. El método create crea un nuevo objeto que utiliza un objeto antiguo como su prototipo.

Reflection

Es fácil de inspeccionar un objeto para determinar qué propiedades tiene al tratar de recuperar las propiedades y el examen de los valores obtenidos. El operador typeof puede ser muy útil para determinar el tipo de una propiedad.

Enumeration

El en el comunicado puede bucle sobre todos los nombres de las propiedades de un objeto. La enumeración incluirá todas las propiedades, incluyendo funciones y propiedades de prototipo que podría no estar interesados en lo que es necesario para filtrar los valores que no desea. Los filtros más comunes son el método hasOwnProperty y el uso de typeof para excluir funciones.

Delete

El operador de eliminación se puede utilizar para eliminar una propiedad de un objeto. Se eliminará una propiedad del objeto, si lo tiene. No va a tocar cualquiera de los objetos en el vínculo prototipo.

Global Abatement

JavaScript hace que sea fácil de definir variables globales que pueden contener todos los activos de su aplicación. Desafortunadamente, las variables globales debilitan la resistencia de programas y deben evitarse. Una forma de minimizar el uso de variables globales es crear una única variable global para su aplicación:

```
var myapp= {};
```

Capítulo 4

Functions

Se utilizan para la reutilización de código, ocultación de información, y la composición.Las funciones se utilizan para especificar el comportamiento de los objetos.

Function Objects

- Funciones en JavaScript son objetos.

Los objetos son colecciones de pares name/value que tengan un vínculo oculto a un objeto prototipo. Objetos producidos a partir de objetos literales están vinculados a Object.prototype. Objetos de función están vinculados a Function.prototype (que está a su vez vinculado a Object.prototype). Cada función también se crea con dos propiedades adicionales ocultos:

- Contexto de la función
- Y el código que implementa el comportamiento de la función.

Cada objeto función también se crea con una propiedad de prototipo. Su valor es un objeto con una propiedad constructor cuyo valor es la función. Esto es distinto de el enlace oculto para Function.prototype.

Dado que las funciones son objetos, se pueden utilizar como cualquier otro valor. Las funciones pueden ser almacenados en las variables, objetos y arrays. Las funciones se pueden pasar como argumentos a funciones, y funciones pueden ser devueltos por funciones. Además, dado que las funciones son objetos, las funciones se tienen métodos. Lo que es especial acerca de las funciones es que pueden ser invocados.

Función Literal

Objetos de función se crean con literales de función:

```
// Crear una variable llamada complemento y almacenar una función
// En lo que suma dos números.
        var add = function (a, b) {
                        return a + b;
                };
```

Un literal de función tiene cuatro partes. La primera parte es la función de palabra reservada. La segunda parte opcional es el nombre de la función. La función se puede utilizar su nombre para llamarse a sí mismo de forma recursiva. El nombre también puede ser utilizado por los depuradores y herramientas de desarrollo para identificar la función. Si una función no se le da un nombre, como se muestra en el ejemplo anterior, se dice ser anónimo. La tercera parte es el conjunto de parámetros de la función, envueltos en paréntesis.

Invocation

Invocar una función suspende la ejecución de la función actual, pasando de control y parámetros para la nueva función. Además de los parámetros declarados, cada función recibe dos parámetros adicionales: this y arguments. El este parámetro es muy importante en la programación orientada a objetos, y su valor se determina por el patrón de invocación. Existen cuatro patrones de invocación en JavaScript: el patrón de invocación de métodos, el patrón de invocación de la función, el patrón de invocación del constructor, y el patrón de invocación se aplican. Los patrones difieren en cómo el parámetro this se inicializa.

The Method Invocation Pattern

Cuando una función se almacena como una propiedad de un objeto, lo llamamos un método. Cuando se invoca un método, esto está ligado a ese objeto. Si una expresión de invocación contiene un refinamiento (. Es decir, una expresión de puntos o [índice] expresión), se invoca como un método:

```
// Create myObject. It has a value and an increment
// method. The increment method takes an optional
// parameter. If the argument is not a number, then 1
// is used as the default.

var myObject = {
        value: 0,
        increment: function (inc) {
                this.value += typeof inc === 'number' ? inc : 1;
        }
};

myObject.increment(); document.writeln(myObject.value);    // 1
myObject.increment(2); document.writeln(myObject.value);    // 3
```

The Function Invocation Pattern

Cuando una función no es la propiedad de un objeto, entonces se invoca como una función:

```
suma var = sumar (3, 4);
// Suma es 7
```

Cuando se invoca una función con este patrón, esto se enlaza con el objeto global

```
// Augment myObject with a double method.
myObject.double = function () {
        var that = this;    // Workaround.

        var helper = function () {
                that.value = add(that.value, that.value);
        };

        helper();    // Invoke helper as a function.
};

// Invoke double as a method.

myObject.double(); document.writeln(myObject.getValue());    // 6
```

The Constructor Invocation Pattern

JavaScript es un lenguaje de herencias de prototipos. Eso significa que los objetos pueden heredar propiedades directamente de otros objetos. El lenguaje es de clase gratis. Se trata de un cambio radical de la moda actual. La mayoría de los lenguajes de hoy son clásicos. Herencias de prototipos es poderosamente expresivo, pero no es ampliamente entendido. Sí JavaScript no confía en su naturaleza prototípica, por lo que ofrece una sintaxis objeto de decisiones que es una reminiscencia de las lenguas clásicas. Pocos programadores clásicos encontraron herencias de prototipos para ser aceptable, y la sintaxis de inspiración clásica oscurece verdadera naturaleza prototípica de la lengua. Es lo peor de ambos mundos. Si una función se invoca con el nuevo prefijo, a continuación, un nuevo objeto se crea con un vínculo oculto por el importe de miembro prototipo de la función, y esto estará vinculado a ese nuevo objeto. El nuevo prefijo también cambia el comportamiento de la instrucción de retorno. Vamos a ver más de eso después.

La recursividad

Una función recursiva es una función que llama a sí misma, ya sea directa o indirectamente. La recursión es una técnica de programación de gran alcance en el que un problema se divide en un conjunto de subproblemas similares, cada uno resuelto con una solución trivial. En general, una función recursiva se llama a resolver sus subproblemas.

Scope(Alcance)

Alcance en un lenguaje de programación controla la visibilidad y tiempos de vida de las variables y parámetros. Este es un servicio importante para el programador, ya que reduce las colisiones de nombres y proporciona gestión automática de memoria:

Cierre

La buena noticia sobre el alcance es que las funciones internas tienen acceso a los parámetros y variables de las funciones que están definidas dentro (con la excepción de esto y argumentos). Esto es una cosa muy buena. Nuestra función getElementsByAttribute trabajado, por haber declarado una variable resultados, y la función interna que pasó a walk_the_DOM también tenía acceso a la variable de resultados. Un caso más interesante es cuando la función de interior tiene un tiempo de vida más largo que su función externa. Más temprano, hicimos una miObjeto que tenía un valor y un método de incremento. Supongamos que queremos proteger el valor de cambios no autorizados. En vez de inicializar miObjeto con un objeto literal, vamos a inicializar miObjeto llamando a una función que devuelve un objeto literal. Esa función define una variable de valor.

Las devoluciones de llamada

Las funciones pueden hacer que sea más fácil lidiar con eventos discontinuos. Por ejemplo, supongamos que hay una secuencia que comienza con la interacción del usuario, haciendo una petición del servidor, y finalmente mostrar la respuesta del servidor.

Módulo

Podemos utilizar las funciones y cierre para hacer módulos. Un módulo es una función u objeto que presenta una interfaz pero que esconde su estado y la aplicación. Mediante el uso de funciones para producir módulos, podemos eliminar casi por completo el uso de variables globales, mitigando de esta manera una de las peores características de JavaScript. Por ejemplo, supongamos que queremos aumentar la secuencia con un método deentityify. Su trabajo consiste en buscar las entidades HTML en una cadena y reemplazarlos con sus equivalentes. Tiene sentido mantener los nombres de las entidades y sus equivalentes en un objeto. Pero ¿dónde debemos mantener el objeto? Podríamos ponerlo en una variable

global, pero las variables globales son malas. Podríamos definirlo en la función en sí, sino que tiene un costo de tiempo de ejecución porque el literal debe ser evaluado cada vez que se invoca la función

Cascada

Algunos métodos no tienen un valor de retorno. Por ejemplo, es habitual que los métodos que establecen o modifican el estado de un objeto a devolver nada. Si tenemos estos métodos devuelven esto en vez de definir, podemos permitir a las cascadas. En una cascada, podemos llamar a muchos métodos en el mismo objeto en secuencia en una sola sentencia.

Memoization

Las funciones se pueden usar objetos para recordar los resultados de las operaciones anteriores, por lo que es posible evitar trabajo innecesario. Esta optimización se llama memoization. Objetos y arrays de JavaScript son muy conveniente para esto.

----------Lecturas

JavaScript Errors - Throw and Try to Catch

La sentencia try le permite probar un bloque de código para los errores.
La sentencia catch le permite manejar el error.
La sentencia throw permite crear errores personalizados.
La sentencia finally permite ejecutar código, después de tratar de atrapar, sin importar el resultado.

Los errores sucederán!
Cuando se ejecuta el código JavaScript, pueden ocurrir errores diferentes.

Los errores pueden ser los errores cometidos por el programador, los errores debidos a la entrada equivocada, y otras cosas imprevisibles codificación:

Capítulo 5

Herencia.

La herencia es un tema importante en la mayoría de los lenguajes de programación. En las lenguas clásicas (como Java), la herencia (o se extiende) proporciona dos servicios útiles. En primer lugar, es una forma de reutilización de código. Si una nueva clase es principalmente similar a una clase existente, sólo tiene que especificar las diferencias. Los patrones de reutilización de código son extremadamente importantes porque tienen el potencial de reducir significativamente el coste de desarrollo de software. La otra ventaja de la herencia clásica es que incluye la especificación de un sistema de tipos. Esto libera sobre todo al programador de tener que escribir las operaciones de fundición explícitas, que es una cosa muy buena porque cuando se lanza, se pierden los beneficios de seguridad de un sistema de tipos. JavaScript es un lenguaje de programación relajado escrito, nunca arroja. El linaje de un objeto es irrelevante. Lo importante de un objeto es lo que puede hacer, no lo que es descendiente de. JavaScript proporciona un conjunto mucho más rico de patrones de reutilización de código. Puede imitar el patrón clásico, pero también es compatible con otros modelos que son más expresivos. El conjunto de posibles patrones de herencia en JavaScript es enorme. En este capítulo, vamos a ver algunos de los modelos más sencillos. Mucho más complicado construcciones son posibles, pero por lo general es mejor que sea sencillo. En las lenguas clásicas, los objetos son instancias de clases, y una clase puede heredar de otra clase. JavaScript es un lenguaje de prototipos, lo que significa que los objetos heredan directamente de otros objetos

Pseudoclassical

JavaScript está en conflicto acerca de su carácter prototípico. Su mecanismo prototipo está oscurecida por una complicada empresa sintáctica que se parece vagamente clásico. En lugar de tener objetos heredan directamente de otros objetos, un nivel innecesario de indirección se inserta de manera que los objetos son producidos por las funciones constructoras. Cuando se crea un objeto función, el constructor Función que produce el objeto función se ejecuta algún código como este:
this.prototype = {constructor: this}; El nuevo objeto de función se da una propiedad prototipo cuyo valor es un objeto que contiene una propiedad constructor cuyo valor es el nuevo objeto de función. El objeto prototipo es el lugar donde los rasgos hereditarios son a depositar. Cada función obtiene un objeto prototipo porque el lenguaje no proporciona una manera de determinar qué funciones están destinados a ser utilizados como constructores. La propiedad constructor no es útil. Es el objeto prototipo que es importante. Cuando se invoca una función con el patrón de invocación del constructor utilizando el nuevo prefijo, esto modifica la forma en que se ejecuta la función.

Object Specifiers

A veces sucede que un constructor se le da un número muy grande de parámetros. Esto puede ser problemático porque puede ser muy difícil de recordar el orden de los argumentos. En tales casos, puede ser mucho más amigable si escribimos el constructor para aceptar un único objeto especificador lugar. Ese objeto contiene la especificación del objeto a ser construido.

Prototypal

En un patrón puramente prototípico, prescindimos de clases. Nos concentramos en cambio en los objetos. Herencias de prototipos es conceptualmente más simple que la herencia clásica: un nuevo objeto puede heredar las propiedades de un objeto antiguo. Esta es tal vez desconocida, pero es muy fácil de entender. Se empieza por hacer un objeto útil.

Funcional

Una de las debilidades de los patrones de herencia que hemos visto hasta ahora es que no recibimos ninguna privacidad.Todas las propiedades de un objeto son visibles. Llegamos sin variables privadas y no hay métodos privados. A veces, eso no importa, pero a veces importa mucho. En la frustración, algunos programadores desinformadas han adoptado un patrón de privacidad de simulación. Si tienen una propiedad que desean hacer privado, que le dan un nombre oddlooking, con la esperanza de que los

demás usuarios del código pretenderán que no pueden ver a los miembros que buscan impares.Afortunadamente, tenemos una alternativa mucho mejor en una aplicación del modelo de módulo. Empezamos haciendo una función que va a producir objetos. Vamos a darle un nombre que comienza con una letra minúscula, ya que no será necesario el uso del nuevo prefijo. La función consta de cuatro pasos:

1. Se crea un nuevo objeto. Hay un montón de maneras de hacer que un objeto. Puede hacer que un objeto literal, o puede llamar a una función constructora con el nuevo prefijo, o se puede utilizar el método Object.create hacer una nueva instancia de un objeto existente, o puede llamar a cualquier función que devuelve un objeto.

2. Define opcionalmente variables de instancia y métodos privados. Estos son sólo vars ordinarias de la función. 3. Se aumenta ese nuevo objeto con métodos. Esos métodos tendrán un acceso privilegiado a los parámetros y las vars definidas en el segundo paso. 4. Se devuelve ese nuevo objeto. Aquí está una plantilla pseudocódigo para un constructor funcional

Regiones

Podemos componer objetos de conjuntos de piezas. Por ejemplo, podemos hacer una función que puede agregar características simples de procesamiento de eventos a cualquier objeto. Añade una sobre el método, un método de fuego, y un registro de evento privado

Apéndice B

Bad parts

Continuar Declaración

La sentencia continue salta a la parte superior del bucle. Nunca he visto un trozo de código que no fue mejorada por la refactorización para eliminar la sentencia continue.

Cambiar Caen A Través

La sentencia switch fue modelado después del FORTRAN IV computarizada ir a la declaración. Cada caso entra a través en el siguiente caso a menos que interrumpes explícitamente el flujo. Alguien me escribió una vez lo que sugiere que JSLint debe dar un aviso cuando pasa al siguiente caso en otro caso. Señaló que se trata de una fuente muy común de los errores, y es un error difícil de ver en el código. Le respondí que eso era todo cierto, pero que el beneficio de compacidad obtenida por la caída a través de más que compensado por la posibilidad de error. Al día siguiente, se informó que hubo un error en JSLint. Se misidentifying un error. Investigué, y resultó que tenía un caso que estaba cayendo a través. En ese momento, he logrado la iluminación. Ya no uso through intencionales otoño. Esa disciplina hace que sea mucho más fácil encontrar los pasantes no intencionales de la caída. Los peores características de un lenguaje no son las características que son obviamente peligroso o inútil. Esos son fácilmente evitados. Los peores características son las molestias atractivos, las características que son a la vez útil y peligroso.

Declaraciones Bloque-less

Un caso o mientras, hacer o para la declaración puede tener un bloque o una sola sentencia. El formulario de declaración solo es otra molestia atractiva. Ofrece la ventaja de ahorrar dos personajes, una ventaja dudosa. Se oscurece la estructura del programa para que los manipuladores posteriores del código pueden insertar fácilmente los insectos.