APRENDERAPROGRAMAR.COM

JAVASCRIPT HOISTING. ERRORES FRECUENTES DE PROGRAMADORES Y CONSEJOS. PROBLEMAS PRECISIÓN DECIMAL (CU01190E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº90 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

HOISTING JAVASCRIPT

La programación JavaScript tiene algunos detalles que conviene ir conociendo poco a poco, ya que en algunos momentos pueden generarnos quebraderos de cabeza si no los conocemos. Uno de estos detalles es el hoisting de las declaraciones de variables, que vamos a explicar.

ELEVACIÓN O HOISTING DE DECLARACIONES DE VARIABLES

Como hemos visto a lo largo del curso, JavaScript admite el uso directo de variables sin necesidad de declaración previa. También hemos dicho que recomendamos declarar las variables.

Cuando se declara una variable en un punto intermedio del código, dicha declaración es "elevada" (hoisted) a la parte inicial del código en el ámbito donde se encuentra dicha variable. Como ámbito más básico podemos pensar en una función. Supongamos que dicha función comienza con diversas instrucciones en las líneas iniciales y que en la línea 15 introducimos una declaración como var x;

Debido a que JavaScript realiza el izado o elevación de las declaraciones de variables, el intérprete JavaScript ejecutará el código como si dicha declaración se encontrara en la primera línea de la función en lugar de en la línea 15.

El hoisting afecta sólo a la declaración de variables. Si hacemos una declaración-inicialización, se realiza el izado sólo de la declaración, pero no de la asignación. Esto implica que la variable constará como declarada desde el comienzo del ámbito, pero el valor asignado no será conocido hasta que se llegue a la línea correspondiente. Por ejemplo var x; es izado pero var x=67; no es izado en su totalidad. En este caso se iza var x; mientras que x=67 no surtirá efecto hasta que se llegue a la línea correspondiente.

Con un ejemplo lo entenderemos mejor. Escribe este código y comprueba sus resultados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
nombre = 'Carlos';
alert('Soy '+nombre);
var nombre;
alert('Soy yo otra vez: '+nombre);
 }
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es que se muestre por pantalla el mensaje <<Soy Carlos>> y seguidamente el mensaje <<Soy yo otra vez: Carlos>>.

Este resultado puede resultar un poco contradictorio respecto a lo que podríamos esperar (sobre todo si hemos programado en otros lenguajes). Mucha gente piensa que al declarar var nombre; en un punto intermedio sería como si esa variable se destruyera y volviera a ser creada con un contenido <<undefined>>. Sin embargo, esto no ocurre en este caso en JavaScript porque lo que hace el intérprete es:

a) Izar la declaración de variable al comienzo de la función

b) Ejecutar el código

Por tanto lo que se ejecuta realmente después de este "preprocesamiento" es lo siguiente:

```
function ejemplo() {
var nombre;
nombre = 'Carlos';
alert('Soy '+nombre);
alert('Soy yo otra vez: '+nombre);
 }
```

Esto puede generar efectos un tanto extraños o confusos. La recomendación para evitarlos es:

a) Declarar siempre las variables antes de usarlas

b) Realizar la declaración de variables al comienzo del ámbito donde se van a usar, en este caso sería al comienzo de la función. Si JavaScript las va a izar, mejor que se vea claramente dónde se sitúan que tenerlas entremezcladas en el código.

Escribe y comprueba los resultados de este código:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Ejemplo aprenderaprogramar.com</title>
<meta charset="utf-8">
<script type="text/javascript">
function ejemplo() {
nombre = 'Carlos';
alert('Soy '+nombre+ ' ' +apellidos);
var nombre = 'Juan';
var apellidos = 'Fernández';
alert('Soy '+nombre+ ' ' +apellidos);
 }
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

El resultado esperado es que se muestre por pantalla: << Soy Carlos undefined>> y a continuación << Soy Juan Fernández>>. Aquí vemos que var nombre = 'Juan'; tiene el mismo efecto que si escribiéramos nombre = 'Juan'; y que la declaración de apellidos combinada con una inicialización, no es izada completamente: sólo se iza la declaración, mientras que la asignación queda pendiente y no se ejecuta hasta llegar a la línea correspondiente.

No comprender el mecanismo de hoisting ocasiona errores en el código. Es algo que los programadores JavaScript deben conocer y saber manejar.

ERRORES FRECUENTES DE PROGRAMADORES JAVASCRIPT

Es frecuente oir ¡JavaScript no funciona!, pero JavaScript sí funciona. De hecho, es una tecnología cada vez más usada. También es cierto que es un lenguaje más rico y complejo de lo que la mayoría de la gente suele creer. La mayor parte de las ocasiones los problemas se deben a que los programadores no entienden lo que ocurre o han cometido equivocaciones.

Existen diferentes listados de errores frecuentes cometidos por programadores JavaScript (sobre todo por programadores que están empezando con JavaScript). A continuación resumimos algunos de estos errores frecuentes. Tenlos en cuenta cuando crees código JavaScript:

Equivocación frecuente	Ejemplo	Explicación y solución
Pensar que el código no funciona a pesar de estar bien programado	No se muestra un efecto que tenía que mostrarse	Muy frecuente: problemas de compatibilidad del navegador. No todos los navegadores, en especial los más antiguos, se comportan como sería de esperar. Realiza un test con otros navegadores.
Mal uso de this	Se invoca this pensando que se refiere a un objeto y se está refiriendo a window, o al revés	Revisar las entregas de este curso donde se habla de this. Es frecuente poder solucionar este error usando bind.
Mal uso del operador = dentro de if	if (numero = 20) { ... }	Esto devuelve siempre true. Error por no usar el operador correcto en comparaciones, == ó ===, normalmente por despiste.

JavaScript no crea ámbitos locales en bucles for	for (var i = 0; i < 10; i++) {
	alert(i);
	}
	alert (i);
		En otros lenguajes i es una variable local que se destruye al terminar el bucle. En JavaScript no, sigue existiendo (como si se tratara de una variable normal) y además con el valor que causa la salida del bucle, en este ejemplo 10.

Pensar que un elemento no responde sin darnos cuenta de que está mal escrito	getElementByID('myId');	Basta cambiar una minúscula por una mayúscula, o al revés, para que el código no responda. En este ejemplo tendría que ser byId en lugar de byID.
Sobreescribir una variable global dentro de una función sin querer	Cuando el código es largo y complejo, podemos repetir nombres de variables sin querer.	Restringir el uso de variables globales. Crear espacios de nombres.

Equivocación frecuente	Ejemplo	Explicación y solución
Hacer roturas de línea sin pensar que equivale a que exista un ; de cierre	var txt = '<ul id="lista1"> - Aprender - Programa ';	Un salto de línea equivale a un ; de cierre. Para concatenar la cadena usar + y delimitar los fragmentos con apóstrofes: var txt = '<ul id="lista1">' + ' - Aprender ' + ' - Programa ' + ' ';
Repetir nombres de funciones	A diferencia de otros lenguajes como Java, en JavaScript no se puede repetir el nombre de una función ni siquiera teniendo distinto número de parámetros.	Mantener un buen control de nombres.

Llamadas a funciones sin los parámetros necesarios	Llamar a obtenerDomicilio (direccion, ciudad, provincia, pais) sin pasarle el país.	Function obtenerDomicilio (direccion, ciudad, provincia, pais) {
		pais = pais || "Colombia"; // Si no se recibe pais, indicar el valor que se debe tomar
		}

Pensar que un objeto no definido contiene null	Intentar comprobar si un objeto contiene null pero el objeto no está definido	Comprobar primero si el objeto está definido y luego comprobar si contiene null.
		if(typeof(myObject) !== 'undefined' && myObject !== null) { //código
		}

No tener en cuenta algunas peculiaridades de sintaxis	No usar sintaxis JavaScript para invocar la clase de un elemento HTML ó la propiedad for.	Usar element.className para clases, element.htmlFor para atributos for.

Tratar de operar con precisión decimal	var x = 0.1;
	var y = 0.2;
	var z = x + y // no da 0.3
	if (z == 0.3) { // ¡No cumple!
		JavaScript (y muchos más lenguajes) tiene dificultades con la precisión decimal. Realizar comparaciones basadas en enteros:
		var x = 0.1;
		var y = 0.2;
		var z = x*10 + y*10
		if (z == 3)

Confundir objetos y propiedades con arrays asociativos	nombreObjeto.['nombre'] alude a una propiedad de un objeto, no a un elemento de un array.	Recordar que JavaScript no admite arrays asociativos como otros lenguajes.
Errores en la concepción del código	Crear closures indebidamente y muchos otros errores a la hora de concebir el código	Deben ir entendiéndose y corrigiéndose progresivamente mediante práctica y estudio.
Pensar que JavaScript es un mal lenguaje de programación	Pensar que JavaScript es un mal lenguaje cuando el problema es que no se conoce	Estudiar JavaScript

EJERCICIO

Dado este fragmento de código. Revísalo y responde a las siguientes preguntas:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
var x = 'Hola amigos'; // variable global
function ejemplo(){
 alert( x ); // esperamos el valor global
 var x;
 x = 'Saludos desde Costa Rica'; // redefinimos la variable en contexto local
 alert( x );  // esperamos el nuevo valor local
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemplo()"> Probar </div>
</body></html>
```

a) Realiza una prueba pulsando en “Probar”. ¿Qué resultados obtienes? ¿Cómo se explican esos resultados?

b) Cambia el código y declara la variable x dentro de la función al mismo tiempo que la inicializas. ¿Qué resultados obtienes? ¿Cómo se explican esos resultados?

c) De los dos casos anteriores (a y b). ¿En cuáles se produce hoisting: en el a), en el b) ó en ambos?

d) En este caso, ¿el hoisting está afectando a los resultados obtenidos? ¿Por qué? ¿Cuáles serían los resultados de ejecutar estos códigos si no existiera hoisting?

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01191E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206