APRENDERAPROGRAMAR.COM

THIS JAVASCRIPT: SIGNIFICADOS. AMBITOS (SCOPE). ANIDAMIENTO. NAMESPACES. EJEMPLO EJERCICIO RESUELTO. (CU01168E)

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

Fecha revisión: 2029

Resumen: Entrega nº68 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

MÁS SOBRE ÁMBITOS (SCOPES)

Hemos hablado básicamente de dos ámbitos: el global, cuyas variables son definidas por todas las funciones, y el local, correspondiente a una función. No obstante, los ámbitos son anidables en el sentido de que una función se puede definir dentro de otra.

AMBITOS INTERNOS

Una función se puede definir dentro de otra. En este caso, cada función define un ámbito local a ella misma.

La regla para saber si una variable es visible es: todo variable definida en un ámbito externo (que envuelve a otro) es conocida en un ámbito interno. Por el contrario, las variables definidas en un ámbito interno no son conocidas en ámbitos externos a dicho ámbito.

Ejecuta 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() { //Ambito de ejemplo
var combustible=40;
var llenarDeposito = function() {
//Ambito de llenarDeposito (interno a ejemplo)
alert('Se han introducido en el depósito '+combustible+' litros');
var faltaParaLlenado = 200-combustible;
}
llenarDeposito();
if (typeof faltaParaLlenado == 'undefined') {alert('Aquí no se conoce la variable faltaParaLlenado');}
}
</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: <<Se han introducido en el depósito 40 litros>> y <<Aquí no se conoce la variable faltaParaLlenado.>>

En resumen, las funciones se pueden definir dentro de funciones, y cada función define un ámbito interno no conocido por las funciones externas.

En cambio, las variables definidas por funciones externas sí son conocidas por las funciones internas. Por ejemplo, la variable combustible definida en la función ejemplo sí es conocida dentro de la función interna referenciada por llenarDeposito.

ESPACIOS DE NOMBRES (NAMESPACES) JAVASCRIPT

Cuando se crea código donde se tienen decenas de archivos JavaScript y miles de líneas, es probable que se den colisiones de nombres: dos variables que toman el mismo nombre, ó dos funciones que toman el mismo nombre, etc. Una colisión de nombres no siempre es negativa en el sentido de que si está bien resuelta y planteada, no tiene por qué generar un conflicto. No obstante, con frecuencia se producen colisiones con efectos indeseados. Veámoslo con un ejemplo: partimos 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">
var capacidadDeposito = 200;
function obtenerNecesidades(contenidoActual){
alert('La capacidad del depósito es ' + capacidadDeposito + ' litros y faltan ' + (capacidadDeposito-contenidoActual) +
' litros para llenarlo');
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="obtenerNecesidades(20)"> Probar </div>
</body></html>
```

El resultado esperado al hacer click es: <<La capacidad del depósito es 200 litros y faltan 180 litros para llenarlo>>

Pero supongamos que es una aplicación web y que al cabo de un par de años, cuando esta aplicación tiene miles de líneas, otro programador introduce otra función que tiene el mismo nombre que la que nosotros habíamos definido antes. El código suponemos que hubiera quedado así:

```
<!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 capacidadDeposito = 200;
//Aquí muchas líneas
function obtenerNecesidades(contenidoActual){
alert('La capacidad del depósito es ' + capacidadDeposito + ' litros y faltan ' + (capacidadDeposito-contenidoActual) +
' litros para llenarlo');
}
// Aquí muchas líneas
function obtenerNecesidades(contenidoActual) {
var capacidadDeposito = 300;
alert('La capacidad del depósito es ' + capacidadDeposito +
' litros y faltan ' + (capacidadDeposito-contenidoActual) + ' litros para llenarlo');
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="obtenerNecesidades(20)"> Probar </div>
</body></html>
```

Ahora el resultado obtenido al hacer click es: << La capacidad del depósito es 300 litros y faltan 280 litros para llenarlo>>

Quizás la capacidad del depósito de un nuevo modelo de vehículo sea de 300 litros, pero nuestro código referido a otro modelo de vehículo ha dejado de funcionar debido a una colisión de nombres: la función que se invoca al hacer click es la última que el intérprete lee dentro del código, y al haber dos funciones con el mismo nombre una queda "tapada" por la otra.

Una forma de tratar de mitigar estos efectos indeseados es definir spacenames o espacios de nombres.

Los espacios de nombres son objetos que actúan a modo de contenedor para envolver todo un conjunto de propiedades, funciones, etc. de forma que su identificación sea más segura y la probabilidad de conflicto de nombres sea baja.

Para crear un espacio de nombres podemos usar esta sintaxis, que ya habíamos explicado como forma de crear objetos en JavaScript:

```
    var nombreObjetoCreado = {
        propiedad1: valorPropiedad1,
        propiedad2: valorPropiedad2,
        propiedadN: valorPropiedadN,
        ...
        método1: function () { ... código ... }
        método2: function (par1, par2, ..., parN) { ... código ... }
        métodoN: function () { ... código ... }
}
```

En nuestro caso, nombreObjetoCreado será el nombre del espacio de nombres. Este nombre será "un prefijo" que habrá que aplicar para invocar cualquier variable (propiedad) o método de ese espacio de nombres.

El prefijo puede ser relativo a la funcionalidad que tiene el código, o puede ser un prefijo que usemos nosotros para identificarnos como programadores, un prefijo creado para desarrollar una aplicación, etc.

Ejecuta este código y comprueba los 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">

var modeloFordSpace = {
capacidadDeposito: 200,
obtenerNecesidades: function (contenidoActual){
alert('La capacidad del depósito es ' + modeloFordSpace.capacidadDeposito + ' litros y faltan ' +
 (modeloFordSpace.capacidadDeposito-contenidoActual) +
' litros para llenarlo');}
}
```

```
var modeloToyotaSpace = {
obtenerNecesidades: function(contenidoActual) {
var capacidadDeposito = 300;
alert('La capacidad del depósito es ' + capacidadDeposito +
' litros y faltan ' + (capacidadDeposito-contenidoActual) + ' litros para llenarlo');}
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="modeloFordSpace.obtenerNecesidades(20)"> Probar (ford) </div>
<div style="color:blue;" id ="pulsador" onclick="modeloToyotaSpace.obtenerNecesidades(20)"> Probar (toyota) </div>
</body></html>
```

Con esta alternativa de diseño usamos a modo de variables propiedades de un objeto y a modo de funciones métodos de un objeto, sirviéndonos dicho objeto para crear el espacio de nombres. En este ejemplo tenemos dos funciones con el mismo nombre, pero al estar en distintos espacios de nombres podemos usar uno u otro sin problemas, simplemente empleando el prefijo adecuado.

Una sintaxis que puede resultar interesante es usar var espacioDeNombres = {}; para crear el espacio de nombres y posteriormente definir sus propiedades y métodos por separado. Ejecuta este ejemplo 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">
var modeloToyotaSpace = {};
modeloToyotaSpace.cocheToyota = function(modelo, capacidadDeposito) {
this.capacidadDeposito = capacidadDeposito;
this.modelo = modelo;
alert('Creado coche Toyota modelo ' + this.modelo + ' con capacidad '+this.capacidadDeposito);
this.obtenerVelocidad = function(){ if (capacidadDeposito<200) {return '150 km/h';} else {return '240 km/h';}}
this.fabrica = 'Ken-Zhuan';
}
function ejemplo() {
var coche1 = new modeloToyotaSpace.cocheToyota('Auris', 175);
alert('La velocidad máxima de este vehículo es ' + coche1.obtenerVelocidad());
alert('Vehiculo fabricado en la fábrica de ' + coche1.fabrica);
}
</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>
```

LA PALABRA CLAVE THIS

Ya hemos hablado sobre el significado de la palabra clave this, que en JavaScript no siempre es el mismo. Vamos a tratar de resumir a qué puede hacer referencia la palabra clave this y repasar este concepto:

1. this para referirnos a un nodo del DOM (ya estudiado)

Un ejemplo lo veríamos aquí:

```
<h2 onclick="alert(this.nodeName)">Haz click aquí para ver el nodeName de this</h2>
<h2>Haz click <span onclick="alert(this.nodeName)" style="color:red;">aquí para ver el nodeName de this</span></h2>
```

En este caso al hacer click en el primer caso obtenemos H2 y en el segundo SPAN porque this alude al nodo dentro del cual estamos haciendo uso de esta palabra clave.

Aquí this (usada dentro de un manejador de evento onclick) nos devuelve el nodo de tipo Element definido por las etiquetas HTML donde se define el evento.

Generalizando, podemos decir que dentro de la función de respuesta a un evento, la palabra clave this hace referencia al elemento que es quien recibe (objetivo o target) el evento.

Este código refleja lo que hemos comentado:

```
window.onload = function () {
var elemsH = document.querySelectorAll("h1, h2, h3, h4, h5, h6");
for (var i=0;i<elemsH.length;i++) {
elemsH[i].addEventListener("mouseover", cambiarColor1);
elemsH[i].addEventListener("mouseout", cambiarColor2);}
function cambiarColor1() { this.style.color = 'orange';}
function cambiarColor2() { this.style.color = 'brown';}
}
```

Aquí las funciones de respuesta al evento son cambiarColor1 y cambiarColor2. Y dentro de estas funciones, this hace referencia al nodo del DOM (elemento) que recibe el evento.

2. this para referirnos a propiedades y métodos de un objeto (ya estudiado)

Lo aplicamos cuando construimos objetos de esta forma:

```
function nombreDelTipoDeObjeto (par1, par2, ..., parN) {
this.nombrePropiedad1 = valorPropiedad1;
this.nombrePropiedad2 = valorPropiedad2;
this.método1 = function () { ... código .... }
this.método2 = function (param1, param2, ..., paramN) { ... código ...}
}
```

Aquí this sirve para aludir al objeto del cual estamos definiendo propiedades y métodos. This nos permite desambiguar los nombres y por ejemplo podemos tener como nombre de parámetro edad y establecer this.edad = edad; quedando así claro que this.edad alude a la propiedad del objeto y edad al parámetro recibido.

3. this para referirnos a un objeto especificado cuando usamos call y apply (ya estudiado)

La función call permite llamar a cualquier función JavaScript indicándole el objeto que actuará como this dentro de la función llamada, así como los parámetros adicionales que sean necesarios.

Lo vemos en este ejemplo:

```
function Profesor (nombre) { this.nombre = nombre || 'Nombre desconocido'; this.salarioBase = 1200; }
function saludar() { alert ('Hola, soy ' + this.nombre); }
function ejemploObjetos() {
var unProfesor = new Profesor('Carlos');
saludar.call(unProfesor);
}
```

Aquí al invocar call sobre la función saludar, se indica que el objeto que actuará como this será el objeto unProfesor. Pero podría haber sido otro objeto, el que nosotros hubiéramos querido.

Las funciones call y apply son muy similares, difieren tan solo en cómo pasan los parámetros a la función invocada (como una lista de items separados por comas con call ó como un array con apply).

4. this por defecto

Si no usamos this en ninguno de los contextos anteriores, ¿qué es this? This por defecto es el objeto dentro del cual se invoca. Si no se invoca dentro de ningún objeto, el objeto por defecto es el objeto global window.

Ejecuta 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() {
alert ('Aquí this es ' + this);
var miObjeto = {};
miObjeto.habla = function() {alert('Aquí ahora this es ' + this);}
miObjeto.habla();
}
</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 <<Aquí this es [object Window]>> y <<Aquí ahora this es[object Object]>>

Ten en cuenta que dentro de ejemplo this no es el nodo del DOM porque ejemplo no es la función de respuesta al evento. En realidad la función de respuesta al evento podemos decir que es una función cuyo contenido es ejemplo(), es decir, una función que da lugar a la ejecución de la función ejemplo, pero esta función no es la función de respuesta al evento.

PERDER EL THIS

Toda función define un ámbito y una función dentro de otra función puede hacer que this no se refiera a lo mismo según dónde lo usemos. Por ejemplo, si tenemos una función anónima dentro de otra, this en la función externa puede estar haciendo referencia al objeto envolvente mientras que this en la función anónima puede estar haciendo referencia al objeto global window.

La solución para mantener una referencia a this en una función anónima interna puede estar en crear un closure. Definiríamos como variable local a la función externa var that = this;, y luego en la función anónima interna haríamos referencia a that, variable auxiliar que nos sirve para mantener la referencia deseada.

En el ejercicio propuesto a continuación veremos un ejemplo de esta situación.

RESUMEN SOBRE THIS

Hemos visto distintas acepciones de this en JavaScript y posiblemente todavía nos falten otras acepciones y nos resulte difícil manejar este concepto y su importancia. No te preocupes ahora de entender todo lo relacionado con este concepto. Continúa avanzando con el curso, irás adquiriendo destreza en el manejo de estas ideas a medida que desarrolles más código y te enfrentes a situaciones variadas.

EJERCICIO

Analiza el siguiente código 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">
function ejemplo() {
var cabecera = document.querySelector('#cabecera');
var respuestaCabecera = function () {
  alert(this);
  setTimeout(function () { alert(this);}, 2000);
};
cabecera.addEventListener('click', respuestaCabecera, false);
}
</script>
```

```
</head>
<body onload="ejemplo()"><div id="cabecera"><h2>Cursos aprenderaprogramar.com HAZ CLICK
AQUÍ</h2><h3>Ejemplos JavaScript</h3></div>
</body>
</html>
```

a) ¿Por qué se muestran diferentes mensajes si en ambos alert estamos invocando this?

b) Modifica el código para que el mensaje que se muestre con retardo muestre lo mismo que el mensaje que se muestra sin retardo. Para ello, haz que en la función anónima sea conocida la referencia a this que existe en la funcion externa.

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

Próxima entrega: CU01169E

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