APRENDERAPROGRAMAR.COM

CLASS JAVASCRIPT ¿CLASES? OBJETOS PREDEFINIDOS. WINDOW, OBJETO GLOBAL. NUMBER, MATH, DATE, REGEXP, ERROR. (CU01144E)

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

Fecha revisión: 2029

Resumen: Entrega nº44 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

CLASS JAVASCRIPT

Las personas que están aprendiendo JavaScript y han programado previamente en otros lenguajes "tradicionales" como Java, C++, Visual Basic, etc. usando programación orientada a objetos, buscan con frecuencia las clases (class) en JavaScript. Pero tal y como indica la especificación oficial, JavaScript no usa clases como lo hacen C++, SmallItalk o Java, aunque sí permita la creación de objetos.

Al igual que muchas personas buscan class en JavaScript, también buscan que el comportamiento de JavaScript se asemeje al de otros lenguajes, y la verdad es que JavaScript tiene en gran medida un comportamiento distinto al de lenguajes tradicionales populares como Java. El uso de términos comunes entre JavaScript y otros lenguajes, y el intento por programar en JavaScript como se haría en otros lenguajes, ha provocado que exista cierta confusión terminológica y con frecuencia errores de diseño o de concepto a la hora de concebir la programación con JavaScript. Por tanto si has programado previamente con lenguajes como C++ ó Java, te daremos la recomendación de que no supongas que JavaScript se va a comportar igual que estos lenguajes.

JavaScript se dice que es un lenguaje orientado a objetos que sigue un paradigma diferente de otros lenguajes como Java, al paradigma de JavaScript se le llama programación basada en prototipos o programación basada en instancias. En la programación basada en prototipos no existen las clases tal y como existen en otros lenguajes, ni existe la herencia tal y como existe en otros lenguajes, aunque sí podemos encontrar ciertas similitudes.

Algunos expertos consideran que JavaScript es el lenguaje más incomprendido entre todos los existentes, porque pocos programadores conocen bien su filosofía y potencialidad. Nosotros a lo largo del curso estamos tratando de comprender JavaScript, y seguiremos en ello a lo largo de los próximos apartados.

DEFINICIÓN DE OBJETO

Ya hemos hablado sobre objetos y tenemos una idea bastante clara de lo que son. Vamos ahora a dar una definición un poco más académica:

<<Un objeto es una estructura de datos que posee un nombre y que está formado por contenedores de información que pueden albergar otros objetos, valores primitivos o funciones. Un objeto pertenece al tipo de dato Object de JavaScript.>>

A los objetos y valores primitivos que alberga un objeto los solemos llamar propiedades, mientras que a las funciones que alberga un objeto las llamamos métodos, de ahí que muchas veces se diga que un objeto es una colección de propiedades y métodos.

Casi todo en JavaScript son objetos. Pero no todo. No son objetos los datos que son tipos primitivos: String, Number, Boolean, Null, Undefined. No obstante, existen tipos objeto equivalentes a los tipos primitivos. Por ejemplo, podemos crear un String como tipo primitivo o como objeto usando:

var cadena = 'aprenderaprogramar.com'; // Tipo primitivo

var cadenaOb = new String ('aprenderaprogramar.com'); // Objeto

Un tipo primitivo se considera un tipo de dato simple: contiene una información simple. En cambio un objeto es un tipo de datos complejo, que puede albergar gran cantidad de información de distinta naturaleza.

OBJETOS EN EL ENTORNO DE JAVASCRIPT Y OBJETOS DE USUARIO

Hemos visto cómo crear un objeto donde nosotros definimos sus propiedades y métodos. Pero existen objetos que son predefinidos por JavaScript y por tanto el navegador los reconoce sin necesidad de que nosotros los creemos.

Objeto predefinido	Utilidad
Objeto global	Objeto único y global, existente antes de que comience la ejecución de código. window suele considerarse que es el objeto global.
Object	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con objetos
Function	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con funciones
Array	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con arrays
String	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con strings
Boolean	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con booleanos
Number	Objeto que permite acceder a propiedades y hacer operaciones relacionadas con números
Math	Facilita el uso de funciones y posibilidades matemáticas
Date	Facilita el trabajo con fechas
RegExp	Permite trabajar con expresiones regulares para reconocer fragmentos de cadenas o patrones presentes en cadenas
JSON	Permite trabajar con el formato de datos JSON
Error, EvalError, RangeError, ReferenceError, SyntaxError, TypeError, URIError	Objetos que permiten controlar y obtener información sobre errores

Escribe por ejemplo alert ('El valor de number max value es ' + Number.MAX_VALUE);

Comprobarás que te devuelve un resultado del tipo "El valor de number max value es 1.7976931348623157e+308". ¿Por qué podemos invocar Number.MAX_VALUE si no hemos definido el objeto Number ni sus propiedades y métodos? Porque este objeto, al igual que otros, son objetos predefinidos de JavaScript.

Los objetos predefinidos de JavaScript nos resultarán muy útiles para distintos cometidos. Por ejemplo el objeto Math nos permite realizar cálculos matemáticos u obtener números pseudoaleatorios. Prueba por ejemplo a hacer que se muestren varios números aleatorios con alert(Math.random());

Todo objeto JavaScript (incluido las funciones) se considera que son instancias del objeto Object, y debido a ello heredan propiedades y métodos de Object. Si no tienes claro qué significa que exista herencia no te preocupes pues lo iremos viendo a lo largo del curso. Te recomendamos que leas esto: herencia en POO, sólo para tener una idea de lo que significa herencia.

window, el objeto global, tiene sus propiedades y métodos. La función que venimos usando para mostrar mensajes por pantalla, alert, es un método de window. Cuando escribimos alert('Hola'); en realidad estamos invocando window.alert('Hola');

Es decir, alert y window.alert son lo mismo para el navegador. El hecho de que no sea necesario escribir window cuando escribimos alert obedece a que si se invoca una función que no ha sido definida de otra manera, se considera que es un método del objeto global.

Prueba 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 ejemploObjetos() {
var texto = 'Aprende a programar';
window.alert('alert es un método del objeto global window');
window.alert(texto);
}
</script>
</head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body>
</html>
```

JavaScript permite definir tipos de objetos y crear instancias de objetos de diferentes maneras y esto ha creado algo de confusión entre los programadores. Otros lenguajes tienen una sola manera de hacer estas cosas, en cambio JavaScript tiene varias. Esto resulta molesto para muchos programadores, sobre todo cuando tienen que revisar código creado por otras personas. La ventaja de esta existencia de múltiples formas sintácticas para hacer lo mismo es que cada programador puede elegir la que le resulte más cómoda o conveniente, y el inconveniente es la falta de uniformidad en el código desarrollado por distintos programadores y la necesidad de conocer esas múltiples formas para poder revisar código no creado por uno mismo.

Ya conocemos una de las formas de definir un tipo de objeto:

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

Como alternativa tenemos esta otra forma:

```
function nombreDelTipoDeObjeto (par1, par2, ..., parN) {
this.nombrePropiedad1 = valorPropiedad1;
this.nombrePropiedad2 = valorPropiedad2;
this.método1 = nombreFuncion1;
this.método2 = nombreFuncion2;
}
function nombreFuncion1 (par1, par2, ..., parN) { ... código ... }
function nombreFuncion2 (par1, par2, ..., parN) { ... código ... }
```

En esta forma de definición lo que antes eran funciones anónimas internas, ahora se han convertido en funciones con nombre externas.

Fíjate que this.método1 = nombreFuncion1; únicamente define el nombre de la función que se ejecutará cuando se llame al método. En el nombre de la función no se incluyen paréntesis, únicamente el nombre.

La función externa reconoce a que se invoca una propiedad del objeto si se incluye this.nombrePropiedad dentro de su código. Igualmente para los métodos.

Prueba este código y comprueba sus resultados (que deben ser 14, 14 y 10).

```
<!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 objetoSerie(unArray) {
        this.nombre = 'Nombre del objeto';
        this.contenidoArray = unArray;
        this.suma = obtenerSuma;
}
```

```
function obtenerSuma() {
                        var suma=0;
                        for (var i=0; i<this.contenidoArray.length; i++){ suma = suma + this.contenidoArray[i];   }
                        return suma;
}

function ejemploObjetos() {
var suma = 0; var serie = [3, 2, 9];
for (var i=0; i<serie.length; i++){ suma = suma + serie[i]; }
alert ('La suma de los números visto como array simple es: ' + suma);
serieOb = new objetoSerie(serie);
alert ('La suma de los números visto como objeto es: ' + serieOb.suma());
otroObjeto = new objetoSerie([2,1,3,4]); alert ('La suma para otro objeto es: ' + otroObjeto.suma());
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body></html>
```

El problema de usar este método basado en funciones auxiliares definidas externamente es que dichas funciones son accesibles no sólo desde objetos del tipo definido, sino también como funciones "normales". Además, al crearse muchos métodos basados en funciones externas normalmente se termina creando un conflicto de nombres: más de una función auxiliar externa con el mismo nombre, de modo que el navegador no sabe qué función es la que debe tomar entre las dos (o más) con el mismo nombre. Por ello recomendamos seguir la definición de tipos de objeto basada en funciones internas antes que este otro método (que también es muy popular).

EJERCICIO

Define un tipo de objeto Medico en JavaScript que tenga como propiedades: nombre (String), personasCuradas (número entero), especialidad (String) y como métodos un método denominado curarPersona y otro método denominado mostrarDatos. El método curarPersona deberá añadir una unidad al valor de la propiedad personasCuradas y el método mostrarDatos deberá mostrar los datos el médico. Por ejemplo, "El médico se llama Juan Eslava, su especialidad es traumatología y lleva curadas 8 personas". Crea dos objetos del tipo definido, e invoca sus métodos para comprobar que funcionan correctamente.

Crea las dos alternativas de código: métodos con funciones internas anónimas o métodos con referencia a funciones externas.

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

Próxima entrega: CU01145E

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