JAVASCRIPT V: OBJETOS Y CLASES

1. Objetos

Al igual que sucede con otros lenguajes de programación, los **objetos** se emplean en JavaScript para **organizar** el **código fuente** de una **forma** más **clara** y para **encapsular métodos y funciones comunes**. La forma más sencilla de crear un objeto es mediante la **palabra reservada** `new` seguida del nombre de la clase que se quiere instanciar:

```
var unObjeto = new Object();
var unaCadena = new String();
```

El objeto `unaCadena` creado mediante el **objeto nativo** String permite almacenar una **cadena de texto** y **aprovechar todas las herramientas y utilidades que proporciona JavaScript** para su manejo. Por otra parte, la variable `unObjeto` almacena un **objeto genérico de JavaScript,** al que se pueden **añadir propiedades y métodos propios** para definir su comportamiento.

Definición de un objeto

Técnicamente, un **objeto de JavaScript** es un **array asociativo** formado por las **propiedades** y los **métodos** del objeto. Así, la forma más directa para definir las propiedades y métodos de un objeto es mediante la **notación de puntos de los arrays asociativos**.

Un **array asociativo** es aquel en el que **cada elemento** no está asociado a su posición numérica dentro del array, sino que está **asociado a otro valor específico**. Los valores de los **arrays normales** se asocian a **índices** que siempre son **numéricos**. Los valores de los **arrays asociativos** se asocian a **claves** que siempre son **cadenas de texto**.

La **forma tradicional de definir** los **arrays asociativos** es mediante la **clase Array**:

```
var unArray = new Array();
unArray['primero'] = 1;
unArray['segundo'] = 2;

alert(unArray['primero']);
alert(unArray[0]);
```

El primer `alert()` muestra el valor 1 correspondiente al valor asociado con la clave primero. El segundo `alert()` muestra `undefined`, ya que como no se trata de un array normal, **sus elementos no se pueden acceder mediante su posición numérica**.

Afortunadamente, existen **métodos alternativos abreviados** para **crear array asociativos**. El **ejemplo anterior** se puede rehacer de la siguiente forma:

```
var unArray = new Array();
unArray.primero = 1;
unArray.segundo = 2;

alert(unArray['primero']);
alert(unArray.primero);
alert(unArray[0]);
```

El método seguido en el ejemplo anterior para **crear el array asociativo** se denomina **"notación de puntos"**. Para acceder y/o establecer cada valor, se indica el nombre del array seguido de un punto y seguido del nombre de cada clave. De forma genérica, la notación de puntos tiene el siguiente formato:

```
nombreArray.nombreClave = valor;
```

Para **acceder a un determinado valor**, también se puede utilizar la **notación de puntos** en vez de la tradicional notación de los arrays, de forma que las dos instrucciones siguientes son equivalentes:

```
unArray['primero'];
unArray.primero;
```

Más adelante se muestra **otra forma aún más abreviada y directa** de establecer el valor tanto de los **arrays "normales"** como de los **arrays asociativos**.

Propiedades

Como los **objetos son** en realidad **arrays asociativos** que almacenan sus propiedades y métodos, la forma más directa para **definir** esas propiedades y métodos es la **notación de puntos**:

```
unObjeto.id = "10";
unObjeto.nombre = "Objeto de prueba";
```

Al contrario de lo que sucede en otros lenguajes orientados a objetos, como por ejemplo Java, para asignar el valor de una propiedad **no es necesario que la clase tenga definida previamente esa propiedad**.

También es posible utilizar la **notación tradicional** de los arrays para definir el valor de las propiedades:

```
unObjeto['id'] = "10";
unObjeto['nombre'] = "Objeto de prueba";
```

Métodos

Además de las propiedades, los **métodos** de los objetos también se pueden **definir** mediante la **notación de puntos**:

```
//definición de un método con una función anónima
unObjeto.muestraId = function() {
 alert("El ID del objeto es " + this.id);
}
```

Los **métodos** se definen **asignando funciones al objeto**. Si la función **no está definida previamente**, es posible **crear una función anónima** para asignarla al nuevo método del objeto, tal y como muestra el ejemplo anterior.

Uno de los aspectos más importantes del ejemplo anterior es el **uso de la palabra reservada `this`**. La palabra `this` se suele utilizar habitualmente dentro de los métodos de un objeto y siempre **hace referencia al objeto que está llamado a ese método**.

De esta forma, en el **ejemplo** anterior:

```
var unObjeto = new Object();
unObjeto.id = "10";
unObjeto.muestraId = function() {
 alert("El ID del objeto es " + this.id);
}
```

Dentro del método, `this` **apunta al objeto que llama a ese método**. En este caso, this hace referencia a `unObjeto`. Por tanto, la instrucción del método `muestraId` es **equivalente a** indicar:

```
alert("El ID del objeto es " + unObjeto.id);
```

El uso de `this` es imprescindible para crear aplicaciones reales. El motivo es que **nunca se puede suponer el nombre que tendrá la variable (el objeto) que incluye ese método**. Como los programadores pueden elegir libremente el nombre de cada objeto, **no hay forma de asegurar** que la siguiente instrucción **funcione siempre** correctamente:

```
alert("El ID del objeto es " + unObjeto.id);
```

Si el objeto **se llamara `otroObjeto`**, **el código anterior no funcionaría correctamente**. Sin embargo, **utilizando** la palabra reservada `this`, el método **funciona siempre** bien independientemente del nombre del objeto.

Además, la palabra `this` se debe **utilizar** siempre que se quiera **acceder** a una **propiedad** de un **objeto**, ya que en otro caso, no se está accediendo correctamente a la propiedad:

```
var unObjeto = new Object();
unObjeto.id = "10";
unObjeto.muestraId = function() {
 alert("El ID del objeto es "+ id);
}
```

Si se ejecuta el **ejemplo** anterior, se muestra el error **"id is not defined"** (la variable id no está definida).

Además de las funciones anónimas, también es posible **asignar** a los **métodos** de un objeto **funciones definidas con anterioridad**:

```
function obtieneId() {
 return this.id;
}

unObjeto.obtieneId = obtieneId;
```

Para asignar una función externa al método de un objeto, sólo se debe **indicar** el **nombre** de la **función** externa **sin paréntesis**. Si se utilizaran los paréntesis:

```
function obtieneId() {
 return this.id;
}

unObjeto.obtieneId = obtieneId();
```

En el **ejemplo anterior**, se ejecuta la función **`obtieneId()`** y el resultado de la ejecución se asigna a la propiedad **`obtieneId`** del objeto. Así, el **objeto no tendría un método llamado obtieneId, sino una propiedad con ese nombre** y con un **valor** igual al **resultado devuelto** por la **función** externa.

Por otra parte, **no es obligatorio** que el **método** del objeto **se llame igual** que la **función externa**, aunque es posible que así sea.

A continuación se muestra un **objeto completo** formado por varias propiedades y métodos y creado con la notación de puntos:

```
var unObjeto = new Object();
unObjeto.id = "10";
unObjeto.nombre = "Objeto de prueba";
unObjeto.muestraId = function() {
 alert("El ID del objeto es "+ this.id);
}

unObjeto.muestraNombre = function() {
 alert(this.nombre);
}
```

Siguiendo este mismo procedimiento, es posible crear **objetos complejos que contengan otros objetos**:

```
var Aplicacion = new Object();

Aplicacion.Modulos = new Array();
Aplicacion.Modulos[0] = new Object();
Aplicacion.Modulos[0].titulo = "Lector RSS";

var inicial = new Object();
inicial.estado = 1;
inicial.publico = 0;
inicial.nombre = "Modulo_RSS";
inicial.datos = new Object();

Aplicacion.Modulos[0].objetoInicial = inicial;
```

En el **ejemplo anterior**, se define un **objeto principal** llamado **Aplicacion** que a su vez **contiene varios objetos**. La propiedad **Modulos** de la aplicación es un **array** en el que cada elemento es un objeto que representa a un módulo. A su vez, cada **objeto Modulo** tiene una **propiedad** llamada **titulo** y otra llamada **objetoInicial que también es un objeto** con las propiedades y valores iniciales del módulo.

La notación tradicional de JavaScript puede llegar a ser tediosa cuando se desarrollan aplicaciones complejas con objetos que contienen otros muchos objetos y arrays. Por este motivo, JavaScript define un **método alternativo de notación** llamado **JSON (JavaScript Object Notation)** y que se verá más adelante.

Notación JSON

JSON (JavaScript Object Notation) es un **formato sencillo para el intercambio de información**. El formato JSON permite **representar estructuras de datos (arrays) y objetos (arrays asociativos) en forma de texto**. La notación de objetos mediante JSON es una de las **características principales de JavaScript** y es un mecanismo definido en los fundamentos básicos del lenguaje.

En los últimos años, JSON se ha convertido en una **alternativa al formato XML**, ya que es **más fácil de leer y escribir**, además de ser mucho más conciso. No obstante, **XML es superior técnicamente porque es un lenguaje de marcado**, mientras que **JSON es simplemente un formato para intercambiar datos**.

La especificación completa de JSON se puede consultar en RFC 4627 y su tipo **MIME** oficial es **application/json**.

Como ya se sabe, la **notación tradicional** de los arrays es **tediosa** cuando existen muchos elementos:

```
var modulos = new Array();
modulos[0] = "Lector RSS";
```

```
modulos[1] = "Gestor email";
modulos[2] = "Agenda";
modulos[3] = "Buscador";
modulos[4] = "Enlaces";
```

Para crear un array normal mediante JSON, se indican sus valores separados por comas y encerrados entre corchetes. Por lo tanto, el ejemplo anterior se puede reescribir de la siguiente manera utilizando la notación JSON:

```
var modulos =["Lector RSS","Gestor email","Agenda","Buscador","Enlaces"];
```

Por su parte, la notación tradicional de los arrays asociativos es igual de tediosa que la de los arrays normales:

```
var modulos = new Array();
modulos.titulos = new Array();
modulos.titulos['rss'] = "Lector RSS";
modulos.titulos['email'] = "Gestor de email";
modulos.titulos['agenda'] = "Agenda";
```

En este caso, se puede utilizar la notación de puntos para abreviar ligeramente su definición:

```
var modulos = new Array();
modulos.titulos = new Array();
modulos.titulos.rss = "Lector RSS";
modulos.titulos.email = "Gestor de email";
modulos.titulos.agenda = "Agenda";
```

En cualquier caso, la notación JSON permite definir los arrays asociativos de una forma mucho más concisa. De hecho, el ejemplo anterior se puede reescribir de la siguiente manera utilizando la notación JSON:

```
var modulos = new Array();
modulos.titulos = {rss: "Lector RSS", email: "Gestor de
email",  agenda: "Agenda"};
```

La notación JSON para los arrays asociativos se compone de tres partes:

- Los contenidos del array asociativo se encierran entre llaves ({ y }) •
  Los elementos del array se separan mediante una coma (,)
- La clave y el valor de cada elemento se separan mediante dos puntos (:)

Si la clave no contiene espacios en blanco, es posible prescindir de las comillas que encierran sus contenidos. Sin embargo, las comillas son obligatorias cuando las claves pueden contener espacios en blanco:

```
var titulosModulos = {"Lector RSS": "rss", "Gestor de email": "email",
```

```
"Agenda": "agenda"};
```

Como **JavaScript ignora los espacios en blanco sobrantes**, es posible **reordenar** las **claves** y **valores** para que se muestren **más claramente** en el código fuente de la aplicación. El **ejemplo anterior** se puede rehacer de la siguiente manera añadiendo nuevas líneas para separar los elementos y añadiendo espacios en blanco para tabular las claves y para alinear los valores:

```
var titulos = {
 rss: "Lector RSS",
 email: "Gestor de email",
 agenda: "Agenda"
};
```

Combinando la notación de los **arrays simples** y **asociativos**, es posible construir objetos muy complejos de forma sencilla. Con la **notación tradicional**, un **objeto complejo** se puede crear de la siguiente manera:

```
var modulo = new Object();
modulo.titulo = "Lector RSS";
modulo.objetoInicial = new Object();
modulo.objetoInicial.estado = 1;
modulo.objetoInicial.publico = 0;
modulo.objetoInicial.nombre = "Modulo_RSS";
modulo.objetoInicial.datos = new Object();
```

Utilizando **JSON**, es posible reescribir el **ejemplo** anterior de forma mucho más concisa:

```
var modulo = {
 titulo : "Lector RSS",
 objetoInicial : { estado:1, publico:0, nombre:"Modulo RSS", datos:{} }
};
```

Los **objetos** se pueden definir en forma de **pares clave/valor** separados por **comas** y encerrados **entre llaves**. Para crear objetos vacíos, se utilizan un **par de llaves sin contenido en su interior {}**.

A continuación se muestra la **notación JSON** genérica para crear arrays y objetos:

- **Arrays**

```
var array = [valor1, valor2, valor3, ..., valorN];
```

- **Objetos**

```
var objeto = { clave1:valor1, clave2:valor2, clave3:valor3, ...,
                        claveN:valorN };
```

La notación abreviada **se puede combinar** para crear arrays de objetos, objetos con arrays, objetos con objetos y arrays, etc. A continuación se muestran un ejemplo de aplicación web real que crea objetos mediante esta notación.

La **forma habitual para definir los objetos en JavaScript** se basa en el siguiente modelo creado con la notación JSON:

```
var objeto = {
 "propiedad1": valor_simple_1,
 "propiedad2": valor_simple_2,
 "propiedad3": [array1_valor1, array1_valor2],
 "propiedad4": { "propiedad anidada": valor },
 "metodo1": nombre_funcion_externa,
 "metodo2": function() { ... },
 "metodo3": function() { ... },
 "metodo4": function() { ... }
};
```

En un mismo objeto se puede **utilizar** de **forma simultánea** la **notación tradicional** de **JavaScript** y la **notación JSON**:

```
var libro = new Object();
libro.numeroPaginas = 150;
libro.autores = [ {id: 50}, {id: 67} ];
```

El **ejemplo anterior** se puede reescribir utilizando **solamente la notación tradicional**:

```
var libro = { numeroPaginas: 150 };
libro.autores = new Array();
libro.autores[0] = new Object();
libro.autores[0].id = 50;
libro.autores[1] = new Object();
libro.autores[1].id = 67;
```

El **ejemplo anterior** también se puede reescribir utilizando **exclusivamente la notación JSON**:

```
var libro = { numeroPaginas: 150, autores: [{id: 50}, {id: 67}] };
```

2. Clases

Los **objetos** que se han visto hasta ahora son una **simple colección** de **propiedades** y **métodos** que se definen **para cada objeto individual**. Sin embargo, en la **programación orientada a objetos**, el concepto fundamental es el de **clase**.

La forma habitual de trabajo consiste en **definir clases a partir de las cuales se crean los objetos** con los que trabajan las aplicaciones.

```
class Rectangulo {

  constructor (alto, ancho) {

    this.alto = alto;

    this.ancho = ancho;

  }

  // Getter

  get area() {

     return this.calcArea();

   }

  // Método

  calcArea () {

    return this.alto * this.ancho;

  }

}


const cuadrado = new Rectangulo(10, 10);


console.log(cuadrado.area); // 100
```