Parte 3
Aproximación a la POO en JavaScript

Capítulo 3-1
Enfoque orientado a "objetos" en JavaScript

1. Introducción

Aunque la implementación del modelo de programación orientada a objetos (POO) no esté tan completa en JavaScript como en C++ o Java, JavaScript ofrece los mecanismos principales gestionados por estos lenguajes.

Recordemos los conceptos más importantes de la POO:

- Encapsulación: reunión de un conjunto de propiedades (parte de tratamiento de datos) y de funciones, también llamadas métodos (parte de procesamientos), dentro de un objeto tipo (quizás es más correcto hablar de clase), con la posibilidad de crear (instanciar) objetos a partir de esta clase.
- Herencia: posibilidad de "fabricar" una nueva clase a partir de una clase existente; esta nueva clase hereda las propiedades y métodos de la clase padre (se pueden añadir nuevas propiedades/métodos a la nueva clase).
- Polimorfismo: un método del mismo nombre asociado a varias clases puede tener comportamientos diferentes para algunas de estas clases.

2. Programación orientada a objetos a través de ejemplos

JavaScript siempre ha sido una pieza esencial en los desarrollos web, principalmente para la programación del lado "cliente", es decir, del lado del navegador. Habitualmente, sin sumergirse de lleno en el lenguaje, los desarrolladores producen código JavaScript de calidad mediocre, contentándose con adaptar el código fuente recuperado de sitios web y manipulando los conceptos POO lo menos posible.

En paralelo, han aparecido un gran número de librerías JavaScript y su uso permite producir aplicaciones de mejor calidad. El dominio de estas librerías supone tener conocimientos básicos de POO en JavaScript.

Por tanto, el objetivo de la exposición que sigue es presentarle lo que hay que saber sobre este asunto. Los conceptos se van a explicar a través de una serie de ejemplos.

Algunos lectores que ya tengan una importante experiencia en otros lenguajes POO (PHP 5, Java, C++...) al principio se pueden sentir incómodos con los aspectos específicos de la POO en JavaScript, la POO por prototipado.

2.1 Secuencia 1: Declaración de los objetos JavaScript de manera "Inline"

Se trata de la manera más sencilla de declarar un objeto en JavaScript.

```
/* Declaración inline de un objeto JavaScript */
/* NB: Esta técnica no permite la herencia a partir del objeto
más adelante */
var Adicion = {
    x: 5,
    y: 10,
    calculo: function()
    {
        return this.x + this.y;
    }
};

/* Uso del objeto Adicion */
document.write("Suma: " + Adicion.calculo());
```

El resultado obtenido de la ejecución será el siguiente:

```
Suma = 15
```

En este tipo de declaración de objeto, es posible prever la especificación de atributos (propiedades) y también de métodos.

La palabra clave `this` sirve para indicar que se está haciendo referencia a los atributos del objeto en sí mismo.

Está claro que con este tipo de declaración no será posible reutilizar este tipo de definición para crear un objeto de las mismas características (o parecidas). Por tanto, este método se utilizará poco (o nada) porque no permite la herencia. No se preocupe, porque volveremos sobre esto más en detalle, a lo largo de esta exposición.

2.2 Secuencia 2: Creación de objetos JavaScript con un constructor

También es posible crear nuestros objetos JavaScript con un constructor (concepto bien conocido en los lenguajes POO). En JavaScript, será suficiente con escribir una función y llamarla posteriormente con la palabra clave `new`. Por tanto, la función jugará el papel de clase sin serlo realmente.

Veamos con un ejemplo el desarrollo que es necesario seguir:

```
/* Definición de una función constructor, de nombre Coche */
var Coche = function()
{
    /* Atributo(s) del objeto */
    this.tieneMotor = true;
    /* Método(s) del objeto */
    this.avanzar = function()
    {
        document.write("avanza");
    }
}

/* Instanciación de un objeto simca1100 a través del constructor Coche */
var simca1100 = new Coche();

/* Visualización del atributo tieneMotor del objeto simca1100 */
```

```
if (simca1100.tieneMotor)
{
    document.write("El coche simca1100 tiene un motor<br />");
}
else
{
    document.write("El coche simca1100 no tiene motor !<br />");
}

/* Llamada al método avanzar del objeto simca1100 */
document.write("El coche simca1100 ");
simca1100.avanzar();
```

En nuestro ejemplo, se define en primer lugar una función `Coche`. Integra un atributo booleano que indica que los coches tienen un motor y un método (función) de nombre `avanzar`, que mostrará "`avanza`" cuando se pida, a partir de un objeto de tipo `Coche` (se entenderá rápidamente).

Posteriormente, se construye un objeto de nombre `simca1100` a partir del constructor `Coche` (siento que el término "constructor" pueda ser confuso en un ejemplo basado en coches):

```
/* Instanciación de un objeto simca1100 a través del constructor Coche */
var simca1100 = new Coche();
```

Ahora, la propiedad (atributo) `tieneMotor` se puede consultar para el objeto `simca1100` instanciado y también se puede ejecutar el método `avanzar`. Cuando se ejecute, tendremos:

```
El coche simca1100 tiene un motor
El coche simca1100 avanza
```

2.3 Secuencia 3: Variables privadas en una instancia de objeto

En el ejemplo de la secuencia anterior, habrá observado que las propiedades (atributos) llevan como prefijo la palabra clave `this`. Esto es lo que las hace usables desde el exterior (caso de la propiedad `tieneMotor`). Por el contrario, por necesidades locales del constructor (cálculo interno), puede declarar variables no expuestas, usando como prefijo la palabra clave `var`.

Veamos un ejemplo concreto:

```
/* Definición de una función constructor de nombre Coche */
var Coche = function()
{
    /* Variable(s) local(s) no accesible(s) desde el exterior del objeto */
    var numeroRuedas = 4;
    /* Método(s) del objeto */
    this.avanzar = function()
    {
    document.write("avanza");
    }
}

/* Instanciación de un objeto simca1100 a través del constructor Coche */
var simca1100 = new Coche();

/* Llamada al método avanzar del objeto simca1100 */
document.write("El coche simca1100 ");
simca1100.avanzar();

/* Intento de visualización de la variable local del constructor Coche */
document.write("<br />");
document.write("El coche simca1100 tiene " + simca1100.numeroRuedas +
" ruedas");
```

Se usa una variable local `numeroRuedas` en el constructor con el prefijo `var`. Esto solo es accesible, como estaba previsto, desde dentro del constructor, como se muestra en la ejecución de este script:

```
El coche simca1100 avanza
El coche simca1100 tiene undefined ruedas
```

2.4 Secuencia 4: Paso de argumento(s) a un constructor

En el ejemplo siguiente, vamos ver que es posible pasar uno o varios argumentos (lo habíamos visto ya para las funciones clásicas) a un constructor:

```
/* Definición de una función constructor de nombre Coche */
var Coche = function(modelo)
{
    /* Atributo(s) del objeto informado durante la instanciación */
    this.modelo = modelo;
}
```

```
/* Instanciación de un objeto simca1100 a través del constructor Coche
con paso de argumento */
var miCoche = new Coche("simca1100");

/* Visualización del atributo modelo del objeto miCoche */
document.write("Tengo un " + miCoche.modelo);
```

El argumento `modelo` está en los paréntesis que siguen al nombre del constructor:

```
var Coche = function(modelo)
```

y está disponible en el cuerpo del constructor por:

```
    this.modelo = modelo;
```

A continuación, es suficiente a nivel de la instanciación del objeto `miCoche` con pasar como argumento un valor ("`simca1100`" en nuestro caso):

```
var miCoche = new Coche("simca1100");
```

La propiedad (atributo) `modelo` del objeto se mostrará por:

```
document.write("Tengo un " + miCoche.modelo);
```

2.5 Secuencia 5: No compartición de los métodos por las instancias de objetos

Dado que los métodos se declaran durante la instanciación de los objetos, sus definiciones están duplicadas en memoria.

En un ejemplo pequeño, el impacto es bajo.

Por el contrario, si su aplicación manipula muchos objetos con métodos múltiples y complejos en los constructores, esto se convierte en inmanejable.

El siguiente ejemplo destaca el problema que acabamos de comentar:

```
/* Definición de una función constructor de nombre Coche */
var Coche = function()
{
    /* Atributo(s) del objeto */
    this.tieneMotor = true;
    /* Método(s) del objeto */
    this.avanzar = function()
```

```
    {
        document.write("avanza");
    }
    this.retroceder = function()
    {
        document.write("retrocede");
    }
}

/* Instanciación de un objeto simca1100 a través del constructor Coche */
var simca1100 = new Coche();

/* Instanciación de un objeto renault12 a través del constructor Coche */
var renault12 = new Coche();

/* Comprobación de la igualdad de métodos avanzar de los objetos simca1100
y renault12 */
if (simca1100.avanzar == renault.avanzar)
{
   document.write("Método avanzar compartido por los objetos simca1100 y
Renault12<br />");
}
else
{
    document.write("Método avanzar no compartido por los objetos simca1100
y Renault12<br />");
}
```

La ejecución confirma que el método `avanzar` no está factorizado:

```
Método avanzar no compartido por los objetos simca1100 y Renault12
```

La noción de prototipo que vamos a descubrir a continuación va a resolver este problema.

2.6 Secuencia 6: Noción de prototipo

Un prototipo es un conjunto de elementos (atributos/propiedades y métodos) que se va a asociar a un constructor (sin "almacenamiento" en el constructor en sí mismo). Durante la ejecución, cuando una propiedad de objeto solicitada en el código no se encuentre en el constructor del objeto en cuestión, se realizará una búsqueda en esta lista "adicional".

Capítulo 2-10
Los formularios

1. La presencia de formularios en las páginas web

Los formularios forman parte integrante de los sitios web actuales. Muy habitualmente habrá tenido que rellenar un formulario, tanto para reservar un viaje como un concierto, inscribirse a una conferencia, pagar sus compras en línea o publicar un comentario en las páginas de las redes sociales, etc.

La integración de un formulario en una página web normalmente hace necesaria la integración de varias competencias. Podemos tener una ergonomía o un diseño de interfaz para la parte de experiencia de usuario, un integrador que crea el formulario en HTML/CSS y un desarrollador que diseña el script que va a recuperar los datos introducidos, los va a validar y los gestiona de manera segura para el tratamiento (reserva, compra, inscripción, etc.). El desarrollador utilizará el lenguaje servidor usado en el proyecto general del sitio web (PHP, Ruby, C#, Java, etc.).

Observación

Observe que puede enviar los datos del formulario por mail, con `action="mailto:direcciones@mail.org"`*. En este caso, no hay ningún tratamiento y los datos se envían y reciben en un formato de texto en bruto.*

2. La estructura de los formularios

2.1 El formulario

Los formularios se insertan en el elemento `<form>`. En este elemento, encontraremos todos los campos útiles y los botones de validación y anulación.

El elemento `<form>` acepta varios atributos:

- `action`: indica la URL del script que va a hacerse cargo de los datos introducidos en el formulario.
- `method`: especifica si los datos se enviarán usando HTTP, con el método `get` o `post`.
- `name`: asigna un nombre al formulario.
- `enctype`: indica el tipo MIME de los datos enviados. El tipo MIME, de *Multipurpose Internet Mail Extension*, permite indicar la naturaleza y el formato de un documento enviado por medio de un formulario. El valor `application/x-www-form-urlencoded` es el valor por defecto. Estos datos se codifican como una pareja clave-valor. Para el envío del archivo, el tipo debe ser `multipart/form-data`, formato adaptado para los datos binarios.

El elemento `<form>` es de tipo `block` y solo tiene definido un valor por defecto para su margen superior.

```
form {
  display: block;
  margin-top: 0em;
}
```

A continuación se indica un sencillo ejemplo:

```
<form method="post" action="mi-script.php"
enctype="application/x-www-form-urlencoded" name="inscripcion">
   ...
</form>
```

2.2 Las etiquetas

El elemento `<label>` permite asociar una etiqueta a un campo. Esta etiqueta se va a mostrar delante del campo y permite nombrarlo. Esto facilitará su utilización y ofrecerá una mejor accesibilidad a las personas discapacitadas. De hecho, será necesario pulsar la etiqueta de un campo para que se active, por ejemplo con un destello en este campo.

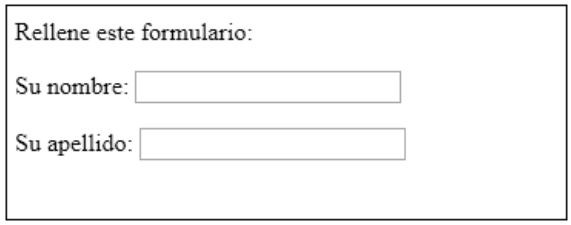
Es el método más aconsejable para nombrar los campos. Es mejor evitar insertar un texto no semántico delante del campo.

A continuación se muestra un sencillo ejemplo con dos campos de entrada:

```
<!doctype html>
<html>
<head>
   <meta charset="utf-8">
   <title>Mi página web</title>
</head>
<body>
   <p>Rellene este formulario:</p>
   <form method="post" action="mi-script.php"
enctype="application/x-www-form-urlencoded" name="inscripcion">
       <p>
           <label for="nombre">Su nombre: </label>
           <input type="text" id="nombre" name="nombre"/>
       </p>
       <p>
           <label for="apellido">Su apellido: </label>
           <input type="text" id="apellido" name="apellido"/>
       </p>
   </form>
</body>
</html>
```

El enlace entre la etiqueta `<label>` y el campo de entrada `<input>` se hace con los atributos `for` e `id`. Los dos valores deben ser idénticos.

A continuación se muestra la visualización obtenida:



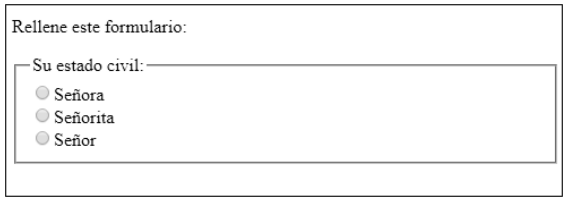
2.3 Agrupar los campos

Para una mejor visibilidad de los campos de un formulario, puede agrupar algunos con el elemento `<fieldset>`. Después puede mostrar una leyenda con el elemento `<legend>`.

A continuación se muestra un sencillo ejemplo con la agrupación de botones de radio:

```
<!doctype html>
<html>
<head>
   <meta charset="utf-8">
   <title>Mi página web</title>
</head>
<body>
   <p>Rellene este formulario:</p>
   <form method="post" action="mi-script.php"
enctype="application/x-www-form-urlencoded" name="inscripcion">
       <fieldset>
           <legend>Su estado civil: </legend>
           <input type="radio" name="estado_civil"
value="señora">Señora<br>
           <input type="radio" name="estado_civil"
value="señorita">Señorita<br>
           <input type="radio" name="estado_civil"
value="señor">Señor
       </fieldset>
   </form>
</body>
</html>
```

A continuación se muestra la visualización obtenida:



2.4 Los atributos comunes

Los elementos HTML dedicados a los formularios comparten un gran número de atributos comunes. Los estudiaremos en las siguientes secciones, en su contexto de utilización. A continuación se muestran los más usados:

- `autocomplete`: permite el autocompletado en los campos. El autocompletado sirve para encontrar entradas ya realizadas en una lista que se muestra bajo el campo.
- `autofocus`: para activar inmediatamente un campo. En un campo de texto, el punto de inserción parpadea en este.
- `checked`: indica que el botón de radio o la casilla de selección que se ha de marcar ya está seleccionada.
- `disable`: desactiva el campo y, por lo tanto, los visitantes ya no lo pueden utilizar.
- `id`: permite identificar de manera única cada elemento del formulario.
- `max` y `min`: indican los valores máximo y mínimo autorizados en los campos de texto y calendario.
- `name`: da el nombre del campo.
- `placeholder`: muestra un texto, una indicación en el campo.
- `readonly`: indica que el campo es de solo lectura.
- `required`: permite especificar que la entrada en el campo es obligatoria.
- `type`: permite definir el tipo del campo.
- `value`: indica cuál es el valor que se debe enviar al script y permite mostrar la etiqueta de los botones de acción.

3. Los campos de texto

3.1 La introducción de texto

En la mayor parte de los formularios, tiene que rellenar la información en forma de texto. Los formularios ofrecen varios campos de entrada de texto. Puede tratarse de un campo de introducción de datos sencillos, como para indicar su nombre y apellidos; campos con varias líneas, como para dejar comentarios, por ejemplo; campos para rellenar una contraseña, etc.

3.2 Los campos de texto sencillos

El elemento `<input>` es el que permite rellenar un texto sencillo. El atributo `type` determina el tipo de campo que queremos. Para un campo de texto sencillo, el tipo es `text: <input type="text">`.

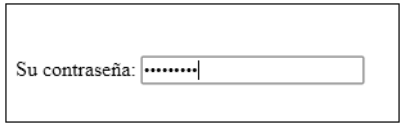
Para un sencillo ejemplo de campos de texto, vaya a la sección anterior, llamada **Las etiquetas**.

3.3 Los campos de texto para las contraseñas

Cuando el visitante necesita realizar una conexión a un espacio reservado, es más prudente que introduzca su contraseña, sin que los caracteres se muestren. En este caso, en el elemento `<input>`, el atributo `type` toma el valor `password`:

```
<p>
   <label for="contrasenia">Su contraseña: </label>
   <input type="password" id="contrasenia" name="contrasenia">
</p>
```

A continuación se muestra la visualización obtenida, con una entrada en el campo:



3.4 Los campos de texto multilínea

Si desea ofrecer a sus visitantes un campo de texto multilínea, utilice el elemento `<textarea>`. El campo de texto multilínea permite la entrada de un texto más largo, en varias líneas, para introducir un comentario o información adicional. Este elemento acepta varios atributos:

- `rows` para definir el número de líneas.
- `cols` para indicar el número de columnas expresado en longitud de caracteres.
- `wrap` permite especificar cómo se gestionan los retornos de carro en el campo y en el envío del formulario. El valor `hard` indica que se envía un carácter de retorno de carro junto con los datos del formulario, y `soft` especifica que el carácter de retorno de carro no se envía.

A continuación se muestra un sencillo ejemplo:

```
<p>
   <label for="comentario">Su comentario: </label>
   <br>
   <textarea id="comentario" rows="6" cols="100"></textarea>
</p>
```

A continuación se muestra la visualización obtenida: