JavaScript como Orientación a Objetos

Gustavo Lacoste (gustavo@lacosox.org)

October 2012

Resumen

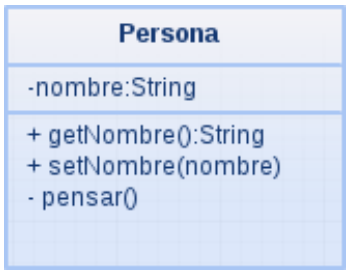
El objetivo de las siguientes notas es generar una estructura en JavaScript que nos permita reutilizar de manera limpia las funciones creadas en otros ficheros JavaScript, para esto reorganizaremos el código intentando aproximarnos al concepto de clase tomando como base su implementación en el lenguaje de programación orientado a objetos PHP.

Palabras Clave: JavaScript, Orientación a Objetos, JSON, Funciones anónimas, Funciones autoejecutables, PHP.

JavaScript NO es un lenguaje Orientado a Objetos, es en realidad el único lenguaje de programación basado en herencia de prototipos [2]. A pesar que encontramos el uso de los operadores **new** y **this** dentro de códigos JavaScript, estos operadores no hacen exactamente lo mismo que en los lenguajes Orientados a Objetos. Por una parte **this** en realidad cambia su comportamiento dependiendo del contexto y por otra parte **new** genera una instancia del objeto referido aunque la instancia puede ser extendida en el futuro mediante el uso de **prototype**, estos operadores actúan como una fachada que hace parecer el lenguaje como orientando a objetos aún cuando su funcionamiento es diferente al de los lenguajes Orientado a Objetos tradicionales.

La razón por la cual JavaScript intenta esconder su verdadera naturaleza detrás de una fachada clásica de POO es la complejidad que representa la comprensión de un nuevo paradigma particular para este lenguaje. La idea de disfrazar el uso de JavaScript bajo la apariencia típica del paradigma Orientado a Objetos no es nueva, por un lado la academia clásica enseña a los nuevos programadores todos los conceptos clásicos de Orientación a Objetos y modelado de software (mediante UML [3]), dejando poco espacio para la discusión de otros paradigmas; parece lógico entonces intentar disfrazar JavaScript con la apariencia de POO en post de una comprensión más rápida y práctica del lenguaje.

A continuación intentaremos mostrar una sintáxis que se asimile lo más posible a la clásica orientación a objetos utilizando los elementos existentes en JavaScript. En UML (Unified Modeling Language) una forma básica para representar la clase Persona sería:



Según este diagrama una clase (molde) para construir objetos Personas se caracteriza entre otras cosas por:

- El atributo nombre de tipo String y de acceso privado (es decir sólo puede ser accedido y modificado por métodos de la propio objeto que se cree a partir de la clase) de la misma forma el método pensar() sólo puede ser llamado desde métodos de la clase.
- Los métodos setNombre y getNombre son públicos es decir pueden ser llamados desde fuera de la clase, en la práctica dichos métodos que serán parte del objeto que instanciemos a partir de la clase podrán ser visibles desde otros objetos pero no así aquellos atributos o métodos privados.

Para realizar una comparación realista vamos a escribir la clase con uno de los lenguajes Orientados a Objetos más utilizados por los desarrolladores web: PHP (*acrónimo de PHP: Hypertext Preprocessor*). PHP es básicamente un lenguaje de scripting diseñado para generar código HTML y enviarlo al navegador web del cliente, es por tanto un lenguaje de programación que se utiliza de lado del servidor.

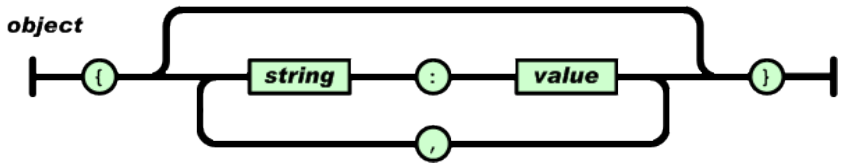
El caso de PHP es diferente, dado que el lenguaje fué originalmente diseñado para realizar scripts estructurados, en PHP es posible programar tanto de manera estructurada como Orientada a Objetos. En siguiente código representa la declaración de la clase Persona en PHP:

Listing 1: persona.php

```
1  <?php
2  class Persona {
3      private $nombre;
4      public function getNombre(){
5          $this->pensar('Entrego mi nombre<br/>');
6          return $this->nombre;
7      }
8      public function setNombre($nombre){
9          $this->pensar('Cambio mi nombre<br/>');
10         $this->nombre=$nombre;
11     }
12     private function pensar($pensamiento){
13         echo $pensamiento;
14     }
15 }
```

```
16
17 $gustavo = new Persona();
18 $gustavo->setNombre("Gustavo");
19 echo 'Mi nombre es '.$gustavo->getNombre().'<br/>';
20
21 $juanelo = new Persona();
22 $juanelo->setNombre("Juan");
23 echo 'Mi nombre es '.$juanelo->getNombre().'<br/>';
24 ?>
```

en esta implementación mediante PHP podemos diferenciar claramente la declación de las variables y métodos. A diferencia en el caso de JavaScript los objetos pueden contener sus propios métodos sin necesidad de clases. Quizá una de las formas más simples de generar una estructura similar a la anterior en JavaScript es hacer uso de la notación literal de declaración de objetos en la que un objeto es declarado como un conjunto desordenado de pares nombre/valor. En esta forma de declaración de un objeto a menudo conocida como **JSON**[1], un objeto comienza con (llave de apertura) y termina con (llave de cierre). Cada nombre es seguido por : (dos puntos) y los pares nombre/valor están separados por , (coma). El flujo se ve representado en el siguiente diagrama:



Siguiendo esta lógica podríamos declarar un objeto de forma literal en JavaScript.

Además una particularidad de JavaScript es la de asignar una función al valor de una variable, puediendo esta ser incluso anónima por lo tanto podríamos tomar esta estructura y asignar a las variables que deseamos ver como métodos una función en lugar de un valor fijo (String, entero, etc) de esta forma podríamos llamar a esta función anónima llamando en realidad al nombre de la variable a la cual le asignamos la función anónima. Otra característica interesante del uso de funciones en JavaScript es que nos permite pasar funciones como parámetros a otras funciones, y obtener funciones como resultados de la ejecución de una función.

La implementación quedaría como sigue:

Listing 2: persona.js

```
1
2  var Persona = {
3      nombre: "",
4      getNombre: function(){
5          this.pensar('Entrego mi nombre<br/>');
6          return this.nombre;
7      },
8      setNombre: function(nombre){
9          this.pensar('Cambio mi nombre<br/>');
10         this.nombre=nombre;
11     },
12     pensar: function(pensamiento){
13         document.write(pensamiento);
14     }
15 };
16
17 Persona.setNombre("Gustavo");
18 document.write('Mi nombre es '+Persona.getNombre());
```

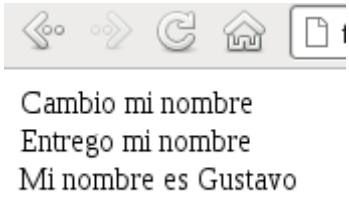
Listing 3: persona_adv.php

```
1  <?php
2  class Persona {
3      private $nombre;
4      public function getNombre(){
5          $this->pensar('Entrego mi nombre<br/>');
6          return $this->nombre;
7      }
8      public function setNombre($nombre){
9          $this->pensar('Cambio mi nombre<br/>');
10         $this->nombre=$nombre;
11     }
12     private function pensar($pensamiento){
13         echo $pensamiento;
14     }
15 }
16
17 $gustavo = new Persona();
18 $gustavo->setNombre("Gustavo");
19 echo 'Mi nombre es '.$gustavo->getNombre().'<br/>';
20
21 $juanelo = new Persona();
22 $juanelo->setNombre("Juan");
23 echo 'Mi nombre es '.$juanelo->getNombre().'<br/>';
24 ?>
```

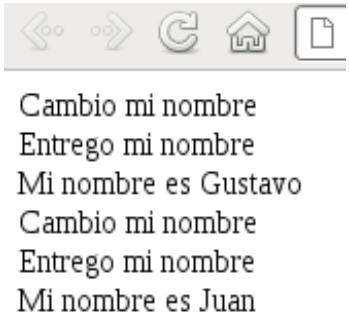
Listing 4: index.html

```
1 <html>
2   <head>
3     <script type="text/javascript" src="persona.js"></script>
4   </head>
5   <body>
6   </body>
7 </html>
```

Resultado de la ejecución con JavaScript:



Resultado de la ejecución con PHP:



Si bien con ambas implementaciones (en PHP y en JavaScript) se obtiene el mismo resultado (para un único objeto) la implementación en JavaScript es sólo una declaración de un objeto. Esto significa que no existe un molde desde el cual generamos los objetos sino que simplemente delcaramos un objeto inmeditamente, por otra parte también observamos que todos los métodos son de hecho públicos, necesitamos que al igual que en PHP la implementación del método pensar sea accesible sólo por el propio objeto y no de forma pública.

Si bien la forma de declaración de objetos de JavaScript es la más simple y fácil de comprender sobre todo por su relación directa con el formato JSON no es la forma más conveniente de hacerlo para este caso dado que los nombres de los atributos son siempre públicos. Una forma simple de acernos aún más es hacer uso de las funciones autoejecutables de JavaScript `(function() ...)();`, este tipo de funciones se ejecutan automáticamente al ser interpretadas y nos servirán para hacer uso del operador new de JavaScript de tal forma que la función autoejecutable retorne en realidad un objeto con los métodos públicos que queremos que sean visibles solamente, manteniendo en su interior los métodos privados. La implementación basado en esta idea quedaría así:

Listing 5: persona_adv.js

```
window.Persona = window.Persona || {};
window.Persona = (function () {
  "use strict";

  function Persona() {
  }
  var nombre = "";

  Persona.prototype.getNombre = function () {
    pensar('Entrego mi nombre<br/>');
    return this.nombre;
  };

  Persona.prototype.setNombre = function (nombre) {
    pensar('Cambio mi nombre<br/>');
    this.nombre=nombre;
  };

  function pensar(pensamiento){
    document.write(pensamiento);
  }

  return Persona;
}());

var gustavo = new Persona();
gustavo.setNombre("Gustavo");
document.write('Mi nombre es '+gustavo.getNombre()+'<br/>');

var juanelo = new Persona();
juanelo.setNombre("Juan");
document.write('Mi nombre es '+juanelo.getNombre()+'<br/>');
```

Listing 6: persona_adv.php

```
<?php
class Persona {
    private $nombre;
    public function getNombre(){
        $this->pensar('Entrego mi nombre<br/>');
        return $this->nombre;
    }
    public function setNombre($nombre){
        $this->pensar('Cambio mi nombre<br/>');
        $this->nombre=$nombre;
    }
    private function pensar($pensamiento){
        echo $pensamiento;
    }
}

$gustavo = new Persona();
$gustavo->setNombre("Gustavo");
echo 'Mi nombre es '.$gustavo->getNombre().'<br/>';

$juanelo = new Persona();
$juanelo->setNombre("Juan");
echo 'Mi nombre es '.$juanelo->getNombre().'<br/>';
?>
```

Esta nueva versión presentada es bastante más compleja que la anterior e introduce algunos conceptos que vale la pena destacar como son:

- **Funciones anónimas:** son funciones sin nombre que se pueden asignar a una variable directamente y ser llamar a estas variables para invocar a la función.
- **Extención de un objeto dinámicamente:** se puede extender un objeto en tiempo de ejecución mediante prototype.

si observamos la primera línea `window.Persona` es básicamente la declaración de un objeto que luego mediante la asignación de una función anónima implementa en su interior un nuevo objeto del mismo nombre que se va extendiendo hasta ser retornado. Este efecto produce que al llamar a la función `window.Persona` este método funcione en realidad como una fábrica de objetos lo que finalmente usaremos para generar 2 objetos en memoria que con valores diferentes en sus atributos. La particularidad que tiene esta forma de escribir una similitud a una clase es que el método pensar efectivamente sólo puede ser llamado desde métodos internos dentro de la fábrica, o *entre comillas* dentro de la clase lo cual es bastante similar a los métodos privados que observamos en la implementación mediante PHP.

Referencias

[1] D Crockford. The application/json media type for javascript object notation (json). 2006.

[2] S Ecma. ECMA-262 ECMAScript Language Specification, 2009.

[3] Object Management Group. OMG Unified Modeling Language (TM) Superstructure Version 2.3, 2010.