PHP y el manejo de objetos en memoria

*— Eugenia Bahit agradece a **Hugo** (@huguidugui) por la **revisión ortográfica** de este artículo*

El presente es una transcripción adaptada del intercambio de ideas sobre objetos y referencias en la estructura foreach, surgidas en la clase de PHP avanzado con Xabi Pico Urisabel.

El tratamiento que PHP hace de los objetos a bajo a nivel, es realmente sorprendente y necesita sin dudas un apartado especial.

Si bien en la documentación oficial de PHP se habla sobre el manejo que el lenguaje hace de los objetos en la memoria aclarando que éstos en realidad, no son pasados por referencia [ver: http://php.net/manual/es/language.oop5.references.php], considero necesario dar una explicación más a fondo con ejemplos concretos puesto que no es algo fácil de relacionar.

De hecho, fue lo que nos sucedió no hace mucho, a **Xabi Pico Urisabel (@xabipic)** y a mí en medio de una clase de PHP y que nos llevó a hacer una investigación más profunda sobre los objetos basándonos en ejemplos concretos.

Cuando se crea un objeto -asignándolo a una variable de tipo `$a = new A();`- la variable $a se almacena en memoria pero el valor de $a no es el objeto en sí mismo, sino un puntero hacia el objeto A.

Gráficamente estaría sucediendo algo como esto:

```
--------+-----------
   D1   |     D2
--------+-----------
  obj A |  $a = D1
--------+-----------
```

(la línea de arriba representa una dirección de memoria ficticia y la de abajo, el dato almacenado en ella)

No interesa si los objetos creados se asignan solo a variables o éstas, además, se introducen en un *array*; los elementos del *array* continuarán apuntando a la dirección en memoria del objeto, ya que la variable lo hace:

```
$c = array($a);
```

```
--------+-----------+-------------
   D1   |     D2    |      D3
--------+-----------+-------------
  obj A |  $a = D1  | $c[0] = D1
--------+-----------+-------------
```

Entonces, si yo creo un alias de `$c[0]` (como el alias generado en la estructura `foreach` mediante el constructor `as`), ese alias también estaría apuntando a la dirección de memoria del objeto. Porque si bien el `as` crea variables al vuelo, no les otorga un valor a las mismas en caso de ser objetos sino que como valor, le otorga un puntero hacia la dirección de memoria del objeto.

Por este motivo, **los objetos pueden modificarse mediante `foreach` sin necesidad de ser pasados por referencia.**

Esto podría explicar además, que si bien lo siguiente agrega la propiedad `foo` al objeto A:

```
php > class A { }
php > $a = new A();
php > $b = new A();
php > $c = array($a, $b);
php > $c[0]->foo = 'bar';
php > print_r($a);
A Object
(
    [foo] => bar
)
```

Al destruir la variable `$a` igual se conserve el objeto:

```
php > unset($a);
php > print isset($a) ? 'True' : 'False';
False
php > print_r($c);
Array
(
    [0] => A Object
        (
            [foo] => bar
        )

    [1] => A Object
        (
        )

)
```

Porque en memoria, estaría pasando esto:

```
--------+-----------+-------------
   D1   |     D2    |      D3
--------+-----------+-------------
  obj A | LIBERADA  | $c[0] = D1
--------+-----------+-------------
```

Por lo tanto, siempre que se esté haciendo referencia al objeto, será éste quien sea modificado y no la variable que lo contiene:

```
php > class A { }
php > $a = new A();
php > $a->foo = 'bar';
php > print_r($a);
A Object
(
    [foo] => bar
)
php > $b = array($a);
php > foreach($b as $obj) $obj->foo = strtoupper($obj->foo);
php > print_r($a);
A Object
(
    [foo] => BAR
)
```

Por ejemplo, en el caso anterior, la estructura `foreach` itera sobre un *array*, construyendo el alias sobre el objeto propiamente dicho. Por lo tanto, éste será un puntero hacia la dirección en memoria del objeto y por ese motivo, es que el `print_r()` final lo muestra modificado.

Sin embargo, si en vez de iterar sobre el *array*, se iterase directamente sobre el objeto, el alias se estaría construyendo sobre las propiedades del objeto y no sobre el objeto mismo. Eso explicaría porqué en ese caso, solo podría modificarse el objeto si el valor de la propiedad es pasado por referencia:

```
php > # sin pasar el valor de la propiedad por referencia
php > foreach($a as $property=>$value) $value = strtolower($value);
php > print_r($a);
A Object
(
    [foo] => BAR
)


php > # ahora, pasando el valor de la propiedad por referencia
php > foreach($a as $property=>&$value) $value = strtolower($value);
php > print_r($a);
A Object
(
    [foo] => bar
)
```

Es también de hacer notar, que **el ámbito de los objetos es independiente al ámbito de las variables**,

puesto que el mismo, jamás deja de ser puntero hacia la memoria **incluso aún dentro de una función**. Es decir, si creo por ejemplo un *array* `$c` con dos objetos `$a` y `$b` y los paso a una función `foo()` como parámetro `$d`, si `$a` se encuentra en *D2* apuntando a *D1*, el primer elemento de `$c`, se encontraría en *D3* apuntando a *D1* y si lo paso como `$d`, `$d` estaría en *D4* apuntando también a *D1*. Esto supone poder modificar *«al array»* indirectamente si lo que se modifica es el objeto, sin necesidad de ser pasado por referencia.

```
php > class A { }
php > $a = new A();
php > $b = new A();
php > $c = array($a, $b);
php >
php > function foo($d) { $d[0]->bar = 123; }
php > foo($c);
php >
php > print_r($a);
A Object
(
    [bar] => 123
)
```

Para ampliar la información sobre las referencias, recomiendo leer la siguiente sección del manual de PHP: http://php.net/manual/es/language.references.php