Parte VII

Autoload - Serialización

Métodos Mágicos - Clonado

Type Hinting - Overloading

Autoload (o autocarga de clases)

Hasta ahora, cada clase que queremos utilizar (la cual está guardada en un archivo independiente), hemos tenido que añadirla con includes. La pregunta que nos hacemos es ¿no hay alguna manera de automatizar el proceso?

Lo que queremos es evitar encontrarnos en cada archivo con un montón de includes sólo por el hecho de usar varias clases simultáneamente. Nos interesaría que las clases que vamos a usar se incluyeran de forma lo más transparente y automática para nosotros.

PHP dispone de varios sistemas de autoload o autocarga de clases.

__autoload()

Uno de los métodos más habituales es usar la función __autoload(), que por llevar dos guiones bajos delante se parece a los "métodos mágicos" como __construct() o __destruct() pero es una función propia de PHP y no pertenece a ninguna clase en particular.

La función __autoload() es automáticamente invocada en caso de que estemos intentando utilizar una clase/interfaz que todavía no haya sido definida.

Un ejemplo simple de uso podría ser el siguiente:

```
function __autoload($nombreClase){
 $archivo = "./" . $nombreClase . ".class.php";
 include_once($archivo);
}
```

http://www.php.net/manual/es/function.autoload.php

__autoload()

La función __autoload() se suele escribir en el archivo inicial como index.php o algún otro del estilo.

Aunque la función de la diapositiva anterior es funcional, es más completa la siguiente, ya que podemos tener clases en diferentes carpetas. Y se puede hacer tan compleja como sea necesario.

```
function __autoload($clase){
    $ruta = "./clases/$clase.php";
    if (file_exists($ruta)){
        include_once $ruta;
        return true;
    }
    $ruta = "./otras_clases/$clase.php";
    if (file_exists($ruta)){
        include_once $ruta;
        return true;
    }
}
```

spl_autoload_register()

Esta función también permite hacer autocarga de clases. Y según la documentación de PHP "proporciona una alternativa más flexible para la carga automática de clases. Por esta razón, el uso de __autoload() no se recomienda y puede quedar obsoleta o ser eliminada en el futuro".

spl_autoload_register() permite registrar varias funciones de autocarga de clases.

http://php.net/manual/es/function.spl-autoload-register.php

Esta función recibe como parámetro una función de autocarga de clases. Así, el ejemplo anterior, que teníamos la posibilidad de acceder a varias rutas, era porque __autoload() sólo podía haber una, pero ahora, cada una de esas rutas podría ir en una función diferente y con spl_autoload_register() las vamos registrando en el sistema, de tal modo que al usar una clase, recorremos una a una las funciones registradas hasta encontrarla y así poder hacer el include (o generar un error).

spl_autoload_register()

Por tanto, el código de la función __autoload() puede ser sustituido por el siguiente:

```
function autocarga1($clase){
  $ruta = "./clases/$clase.php";
  if (file_exists($ruta)){
    include_once $ruta;
}
function autocarga2($clase){
  $ruta = “./otras_clases/$clase.php";
  if (file_exists($ruta)){
    include_once $ruta;
}
spl_autoload_register(“autocarga1”);
spl_autoload_register(“autocarga2”);
```

spl_autoload_register()

Pero si tenemos en cuenta que desde la versión 5.3.0 de PHP están disponibles las funciones anónimas (funciones que no tienen nombre, ya que son de un solo uso), el código podría quedar algo más compacto de la siguiente forma:

```
spl_autoload_register(function($clase){
  $ruta = "./clases/$clase.php";
  if (file_exists($ruta)){
    include_once $ruta;
  }
});
spl_autoload_register(function($clase){
  $ruta = "./otras_clases/$clase.php";
  if (file_exists($ruta)){
    include_once $ruta;
  }
});
```

Podéis mirar otras funciones SPL, que contienen utilidades interesantes para trabajar con clases y objetos:

http://www.php.net/manual/es/ref.spl.php

Serialización

La serialización es el mecanismo por el cual convertimos una estructura de datos (array, objeto, etc) en una cadena.

Las funciones PHP **serialize()** y **unserialize()** implementan mecanismos de serialización, que convierten objetos en cadenas de texto que podemos almacenar o transmitir.

- Almacenar un objeto en la BD.
- Transmitir un objeto entre páginas o entre sistemas.
- Almacenar un objeto en una cookie.
- Si almacenamos un objeto en una variable de sesión se hace de manera transparente.

Ejemplo de Serialización

```
$x = new Objeto($datos, $para, $construir);

$cadena = serialize($x);
$y = unserialize($cadena);
```

IMPORTANTE: Antes de hacer **unserialize()** debemos haber cargado la declaración de la clase del objeto sobre el que vamos a operar, sino no sabrá hacerlo.

Como el hecho de almacenar un objeto en una variable de sesión PHP lo serializa internamente, también debemos haber cargado la clase del objeto antes de hacer el session_start().

Comparación de Objetos en PHP

Operador ==

Dos objetos son iguales si tienen los mismos atributos con los mismos valores.

Operador ===

Dos objetos son iguales si tienen la misma referencia (apuntan a la misma instancia).

Cuando serializamos un objeto y luego deshacemos la operación, si comparamos el objeto antes y después, con == nos dará verdadero, ya que son iguales. Pero al compararlos con === nos da falso, ya que no son la misma referencia, dado que uno es copia del otro en posiciones de memoria diferentes.

Métodos Mágicos

Los métodos que comienzan por "__" (dos guiones bajos) se denominan métodos mágicos.

De momento ya hemos visto dos, que son __construct() y __destruct().

PHP dispone de una serie de funciones "mágicas" que implementan mecanismos de control cuando ocurren ciertas circunstancias con los objetos.

Nos podemos encontrar con una buena cantidad de ellos, como:

__set(), __get(), __isset(), __unset(), __sleep(), __wakeup(), __call(), __callStatic(), __toString(), __invoke(), __set_state(), __clone()

Más en: http://www.php.net/manual/es/language.oop5.magic.php

PHP desaconseja que creemos métodos propios que comiencen con doble guión bajo, ya que es algo reservado para este tipo de métodos.

Métodos Mágicos

Los métodos __sleep() y __wakeup() afectan a la serialización de objetos.

- sleep(): Se llama, si existe, antes de serializar un objeto. Debe devolver un array con los nombres de los campos (atributos) que deseas que sean serializados. Los demás se ignoran.

- __wakeup(): Se llama, si existe, después de des-serializar un objeto. El uso para el que está destinado es restablecer las conexiones de BF que se puedan haber perdido durante la serialización y realizar otras tareas de reinicialización.

Clonado de objetos

La palabra clave **clone** nos sirve para clonar objetos.

```
$f2 = clone $f1;
```

Cuando se realiza una clonación se invoca, si existe, al método mágico **__clone()** sobre el objeto clonado.

Por tanto, dentro de **__clone()** la palabra **$this** es una referencia al clon que se acaba de crear y no al objeto original.

Type Hinting

Aunque sin nombrarlo de esa manera, esto ya lo hemos utilizado antes. Consiste en especificar el tipo del objeto que estamos pasando como parámetro a una función. Así la función no acepta otra cosa diferente (que en PHP sería posible al ser un lenguaje débilmente tipado).

Vimos la clase Jaula, e hicimos el siguiente método:

```
public function meter(Mamifero $animal){
  $this->contenido = $animal;
}
```

Si no escribiéramos el nombre de la clase antes del parámetro, PHP aceptaría una llamada como las siguientes, aunque no fueran correctas y no tuvieran sentido:

```
$jaula->meter("métete tú");
$jaula->meter(27);
```

Type Hinting

También funciona con interfaces, pero no con traits (los traits los veremos al final de esta parte).

Vimos la clase Carrito, e hicimos el siguiente método:

```
public function meter(iEnCarrito $elemento){
    $this->productos[] = $elemento;
}
```

Es importante tener en cuenta que el Type Hinting no funciona con tipos primitivos (enteros, reales, cadenas) sino sólo con objetos.

Sobrecarga en PHP

La documentación de PHP hace uso del término sobrecarga (overloading) para referirse a un concepto diferente de la sobrecarga entendida en la POO tradicional.

Para PHP "overloading" significa acceder a métodos o propiedades "inaccesibles", ya sea:

- Porque no existan.
- Porque no haya visibilidad.

Cuando hablamos de métodos o propiedades inaccesibles nos referimos a aquellos que no se han declarado o que no son visibles en el ámbito en el que nos encontramos.

Sobrecarga de Propiedades

Existen diversos métodos mágicos que podemos definir para establecer comportamientos cuando ocurra la "sobrecarga" en PHP. Todos estos métodos deben definirse como **public**.

- **__set()** se ejecuta al escribir datos sobre propiedades inaccesibles.
- **__get()** se utiliza al consultar datos a partir de propiedades inaccesibles.
- **__isset()** se lanza al llamar a isset() o empty() sobre propiedades inaccesibles.
- **__unset()** se invoca cuando se usa unset() sobre propiedades inaccesibles.

http://php.net/manual/es/language.oop5.overloading.php

Ejemplo de Sobrecarga

El comportamiento normal al acceder a una propiedad inaccesible puede variar al trabajar con los métodos mágicos.

```
$f1 = new Freelance("Pepe", 25);
echo $f1->precioHora;  //Siendo precioHora private
echo $f1->mote;     //Siendo mote una propiedad inexistente
```

Este código, desde fuera de la clase, normalmente dará un error fatal, ya que no puedo acceder a esas propiedades, y se pararía la ejecución. Para evitar que ocurra es para lo que tenemos los métodos mágicos vistos en la diapositiva anterior.

Si en la clase tenemos definido el método __get() no dará error fatal y a cambio, se ejecutará ese método.

```
public function __get($name){
  echo "Estas intentando acceder a $name y no deberías";
}
```

Sobrecarga de métodos

- __call() es lanzado al invocar un método inaccesible en un contexto de objeto.
- __callStatic() es lanzado al invocar un método inaccesible en un contexto estático.

Ambos métodos llevan dos parámetros, el primero corresponde al nombre del método que se está llamando, y el segundo es un array enumerado que contiene los parámetros que se le han pasado al método invocado.

```
public function __call($nombre, $arrayParam){
 echo "La llamada al método $nombre con ";
 echo count($arrarParam) . " parámetros no es correcta";
}
```