Validación de datos en PHP

Validación de datos en PHP

La validación de datos es una de las tareas más oscuras (menos llamativas) pero, al mismo tiempo de una importancia capital, cuando escribimos código PHP. Si tenemos un formulario que debe ser rellenado por un usuario, es muy posible que los datos estén mal escritos (direcciones de correo electrónico sin el signo arroba, por ejemplo). Si esos datos van a ser almacenados o procesados de algún modo, o leídos y empleados posteriormente (para eso se rellena el formulario, después de todo), es obvio que, antes de grabarlos o procesarlos de algún modo es imprescindible hacer una comprobación de que se ajusten al formato esperado. PHP cuenta, desde la versión 5.2 con un recurso muy interesante, nativo del propio PHP (es decir, que no necesitamos añadir ninguna extensión ni librería) que va a hacer nuestra vida más fácil, y que vamos a conocer en este artículo.

Tradicionalmente, en PHP los datos se han validado con expresiones regulares. Este sistema funciona, pero es un engorro emplearlo. Al menos, yo siempre lo he encontrado extremadamente tedioso. Afortunadamente, los desarrolladores de PHP pensaron lo mismo que yo (y que un montón de gente) y nos dieron un recurso que nos facilita enormemente la validación de datos sin recurrir a las expresiones regulares: el uso de **filtros**.

LA FUNCIÓN filter_var()]

Esta función permite comparar un dato contra un filtro tipificado en el propio PHP, identificado por una constante del lenguaje. Recibe dos argumentos (más, en algunos casos, un tercero opcional, del que hablaremos enseguida). El primero es el dato que queremos comprobar y el segundo es la constante que representa el filtro con el que queremos validar el dato. Si el dato se ajusta al filtro, la función nos devolverá el mismo dato que le hemos pasado en el primer argumento. Si no, nos devolverá un false lógico o, según el caso, un valor null.

Veamos un ejemplo de uso muy típico, para entender como opera esta función: la comprobación del formato de una dirección de correo electrónico. Supongamos que tenemos una variable que contiene lo que haya llegado de un formulario, o una cadena obtenida de otra entrada, y que, esperamos, se ajuste a un formato correcto de dirección de e-mail.

```
$correo = "webmaster@eldesvandejose.com";
```

A continuación vamos a comprobar si se ajusta a un filtro de correo electrónico correcto:

```
$resultado = filter_var($correo, FILTER_VALIDATE_EMAIL);
```

Si la dirección es correcta, en la variable $resultado se almacena la misma dirección. Si no lo es, se almacena false. Por lo tanto, la comprobación es muy simple:

```
if ($resultado == $correo){
    echo "La dirección es correcta.";
 } else {
    echo "Compruebe la dirección, por favor.";
 }
```

Y ya está.

Otro ejemplo muy útil es la aplicación de un filtro para determinar si un dato booleano es realmente booleano (o puede asimilarse como tal) o no, como vemos a continuación:

```
$dato = true;
 $resultado = filter_var($dato, FILTER_VALIDATE_BOOLEAN);
```

Sin embargo, este ejemplo, que parece extremadamente simple, tiene "gato encerrado". Si el dato es true, la función nos devolverá true. Sin embargo, si el dato es false, la función nos devuelve false y no tenemos, a priori, manera de saber si el false que nos devuelve es porque se trata del dato original, o porque la validación ha fallado. Aquí es donde entra en juego el tercer argumento. En este caso, la validación correcta la haremos así:

```
$resultado = filter_var($dato, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE);
```

De este modo, si obtenemos un valor true o false, sabremos que es el dato original y, por tanto, la validación se ha efectuado correctamente. En el caso de que la variable contuviera un dato que no pueda evaluarse como booleano, la función devolvería null.

ATENCIÓN. A la hora de efectuar la validación de un dato booleano, debes saber que PHP asimilia ciertos valores no booleanos a este tipo de dato. Así pues, datos como "on", "yes", "1" y 1 se evalúan como true y datos como "off", "no", "", "0" y 0 se evalúan como false (esto sólo funciona correctamente, en el caso de los filtros booleanos, si está presente el parámetro FILTER_NULL_ON_FAILURE).

Cómo puedes ver, el uso de esta función, desconocida por muchos desarrolladores, es extremadamente simple y útil. No obstante, te aconsejo que le eches un vistazo rápido a la mejor fuente de sabiduría: la documentación oficial. La de esta función la puedes encontrar en http://php.net/manual/es/function.filter-var.php.
Además, debes saber que lo expuesto aquí son sólo unos ejemplos, para mostrarte lo simple y útil que resulta este recurso. La lista completa de filtros y de opciones útiles para el tercer argumento puedes encontrarla en http://php.net/manual/es/filter.filters.php.
Como verás, con la lista de filtros y opciones que contempla PHP en la actualidad (algunos se han incorporado en la versión 7 del lenguaje), es muy difícil que no encuentres el que necesitas.

BUSCANDO UN DATO EN UNA CADENA]

En ocasiones tenemos que buscar un dato en una cadena. Por ejemplo, a lo mejor necesitamos comprobar si una frase contiene una dirección de correo electrónico. Imagina la siguiente declaración:

```
$frase = "Mi email es correo@correo.com. Enviáme algo.";
```

Si usamos la función filter_var() "a pelo" no nos sirve. Imagina que haces lo siguiente:

```
$resultado = filter_var($frase, FILTER_VALIDATE_EMAIL);
```

Esto devolverá false, porque filter_var() no busca que exista el filtro, sino que el dato que le pasamos cómo argumento coincida con el filtro. Esto es un problema porque, una vez más, nos aboca al uso de expresiones regulares. Sin embargo, existe una manera de solucionarlo, usando esta función:

```
<?php
 // Declaramos una frase.
 $frase = "Mi email es correo@correo.com. Enviáme algo.";
 $contieneUnCorreo = false; // Por defecto, contamos con que no contenga un correo válido
 $cadenaPartida = explode(" ",$frase); // Fragmentamos la frase "rompiéndola" por los espacios en blanco.
 /* Iteramos sobre cada uno de los fragmentos para procesarlo. */
 foreach ($cadenaPartida as $fragmento){
  $fragmento = trim ($fragmento, " tnrx0B."); // Eliminamos espacios en blanco, puntos, y cualquier carácter que queramos depurar, al principio y al final de cada fragmento.
  if ($fragmento == filter_var($fragmento, FILTER_VALIDATE_EMAIL)) $contieneUnCorreo = true; // Comprobamos cada fragmento para ver si coincide con el filtro.
 }
 /* Visualizamos si alguno de los fragmentos ha coincidido con el filtro. */
 if ($contieneUnCorreo){
  echo "Hay un correo";
 } else {
  echo "No hay correo";
```

```
 }
?>
```

Esto sí funciona. Y si eres aficionado a las buenas prácticas de programación, puedes desacoplar el proceso, sacándolo a una función externa o, incluso, si lo consideras adecuado, a un método de una clase. Aquí te lo he puesto para que te resulte lo más claro y legible posible.