Paso de parámetros por la URL

Las aplicaciones web pueden recibir parámetros a través de la propia URL de invocación del servidor.

Imagina que tenemos este link en un documento HTML:

```
<a href="pagina.php?variable1=valor1&variable2=valor2&etc…">
```

Al hacer clic en él, pediremos al servidor que ejecute el programa cuyo código fuente está en el archivo *pagina.php*, ¿verdad?

Pues bien, ese programa *pagina.php* tendrá a su disposición unas variables llamadas "variable1, **variable2*, etc, que son las que han viajado hasta el servidor en la URL.

Para acceder a esas variables, PHP utiliza un array global llamado **$_GET**, que se indexa con el nombre de las variables. Así:

```
<?php
echo "La variable 2 vale:".$_GET['variable2']."<br>";
?>
```

Observa el uso del carácter punto (.) para concatenar strings en la salida de *echo*. Esto, en Java y muchos otros lenguajes, se haría con el carácter más (+). PHP es un poquito especial en este detalle.

Entrada de datos a través de formulario

Como PHP se ejecuta dentro de HTML, sólo puede recibir datos del usuario de la aplicación a través del navegador web.

Y sólo hay una forma de introducir datos en una página web: *a través de un formulario*.

Veámoslo con un ejemplo. Supongamos que hemos definido en HTML este sencillo formulario:

```
<body>
<form method="post" action="destino.php">
Nombre<br/>
<input type="text" name="nombre"><br/>
Apellidos<br>
<input type="text" name="apellidos"><br/>
<input type="submit">
</form>
</body>
```

Al pulsar el botón "Enviar", se cargará el script *destino.php* en el servidor.

Ese script recibirá dos variables HTML llamadas *nombre* y *apellido*, con el valor que el usuario haya introducido en el formulario.

Para acceder a las variables HTML, se usa el array del sistema **$_POST**, indexándolo con el nombre de la variable:

```
<?php
     echo "La variable nombre vale".$_POST['nombre']."<br>"
?>
```

Observa que *$_POST* es una variable semejante a *$_GET*. Puedes utilizar una u otra según el valor del atributo *method* de tu formulario HTML.

La variable **$_REQUEST** sirve tanto para POST como para GET. **Por eso será la que nosotros usaremos preferentemente en nuestros programas**.

Validación y saneamiento de formularios

Los datos que llegan desde un formulario son una fuente inagotable de quebraderos de cabeza. Para empezar, casi todos los ataques a las aplicaciones web provienen de intentos de los atacantes de usar los formularios como puerta de entrada al servidor. Y no hay que menospreciar el caos que puede provocar en una aplicación un usuario bienintencionado pero torpe que envía al servidor cosas totalmente imprevistas a través de un formulario.

Por lo tanto, todo lo que los usuarios de la aplicación escriban en un formulario debe filtrarse:

1. En el cliente, usando los atributos HTML5 necesarios y, si hace falta, recurriendo a Javascript.
2. En el servidor, mediante PHP o el lenguaje de servidor que estemos usando.

Ese filtro puede ser algo muy simple, como eliminar todos los caracteres no alfabéticos que provengan del formulario, o algo más complejo, como comprobar que el usuario ha escrito una dirección de email bien formada.

Ese proceso de filtrado se denomina **validación y saneamiento**.

Para ayudarnos en esta validación, PHP proporciona la función ***filter_var()***, que limpia diferentes conjuntos de caracteres sospechosos de cualquier dato que provenga del formulario. Esta función recibe como parámetro un string y permite tanto sanearlo como validarlo.

Por ejemplo, supongamos que tenemos un sencillo formulario con dos campos, *nombre* y *email*:

```
<form action='procesa_formulario.php'>
    <input type='text' name='nombre'>
    <input type='text' name='email'>
    <button type='submit'>Enviar</button>
</form>
```

El script *procesa_formulario.php* recibirá los datos enviados por este formulario (nombre y email) en las variables *$_REQUEST[“nombre”]* y *$_REQUEST[“email”]*. Pues bien, si queremos sanear (limpiar) cualquier carácter sospechoso que pueda venir en esas variables, podemos hacerlo así:

```
if (!isset($_REQUEST["nombre"])) {
    echo "Error: el campo nombre es obligatorio";
}
if (!isset($_REQUEST["email"])) {
    echo "Error: el campo email es obligatorio";
}
$nombre = filter_var($_REQUEST["nombre"], FILTER_SANITIZE_STRING);
$email = filter_var($_REQUEST["email"], FILTER_SANITIZE_EMAIL);
```

Tras la ejecución de este código nos habremos asegurado de que el usuario ha rellenado los dos campos y que esos campos no contienen ningún carácter sospechoso de ataque.

filter_var() admite otros valores como segundo parámetro. Son estos:

- **FILTER_SANITIZE_STRING**: elimina cualquier etiqueta HTML que encuentre en el string.
- **FILTER_SANITIZE_NUMBER_INT**: elimina cualquier carácter que no sea numérico (solo respeta los caracteres “+” y “-“)
- **FILTER_SANITIZE_URL**: elimina cualquier carácter que no forme parte de una URL. El decir, solo deja las letras, los números y algunos caracteres especiales como _, : o ?
- **FILTER_SANITIZE_EMAIL**: elimina cualquier carácter que no forme parte de una dirección de email típica.

Si solo queremos validar un string procedente de un formulario, podemos cambiar los valores anteriores por FILTER_VALIDATE_STRING, FILTER_VALIDATE_NUMBER_INT, etc. Es decir, cambiaremos la palabra SANITIZE por VALIDATE. De ese modo, la función *filter_var()* no cambiará el string, sino que comprobará si pasa el filtro o no y nos devolverá *true* o *false*.

Existen otros filtros más complejos que puedes consultar en la referencia oficial del lenguaje.

Y, por supuesto, para construir validaciones más específicas, siempre puede programarlas por tu cuenta y riesgo, usando las funciones de procesamiento de strings que te ofrece PHP y currándotelo un poco.