Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Tutorial de PHP y librerías de Bulma **(187903 lectures)**
Per **Antonio Tejada Lacaci**, ***Wildfred*** ()
Creado el 26/01/2000 00:00 modificado el 26/01/2000 00:00

Vuestro querido webmíster os presenta un tutorial de iniciación al PHP y a las librerías de desarrollo creadas para el sitio web de Bulma.

Introducción al PHP y a las librerías de Bulma

Antonio Tejada Lacaci wildfred@teleline.es(1)

v0.3, 26 de Enero 1999

Introducción al lenguaje interpretado PHP de generación dinámica de páginas web y a las librerías desarrolladas para el sitio web de Bulma(2).

---

1. Introducción al PHP

PHP(3) es un lenguaje interpretado que sirve principalmente para realizar páginas html dinámicas, aunque sus autores aseguran emplearlo incluso para hacer shell-scripts, ya que hay intérpetes php en línea de comandos.

A pesar del ímpetu de sus desarrolladores, aquí nos centraremos especialmente en la generación de páginas dinámicas con php (versión 3), apache y el módulo mod_php.

PHP es muy parecido al C, si ya sabes C, se puede decir que ya sabes el 90% del lenguaje PHP, únicamente se diferencian en que:

- PHP no es *case sensitive* (no distingue mayúsculas de minúsculas), salvo bugs en el tratamiento de objetos.
- en PHP no se declaran las variables y no tienen un tipo fijo, sino que una misma variable puede almacenar a lo largo de su vida valores de todo tipo (números, cadenas...).

1.1 PHP y HTML

Para escribir código PHP dentro de una página html, tenemos varias alternativas:

- Incluir el código entre `<?` y `?>`
- Incluir el código entre `<?PHP` y `?>`
- Incluir el código entre bloques `<SCRIPT LANGUAGE="php">` y `</SCRIPT>`

El resultado de la salida estándar de ese código será escrito en esa misma posición de la página html.

Ejemplo:

```
<HTML>
<BODY>
<?PHP
echo("¡Hola Mundo!<BR>");
?>
</BODY>
```

```
</HTML>
```

La página anterior, si la salvamos como `ejemplo.phtml` y la cargamos con el navegador, produce como resultado una página HTML con el texto "Hola Mundo". Huelga decir que para que funcione, es necesario tener instalado un servidor web con soporte para PHP y asociar la interpretación de PHP a la extensión `phtml`.

En el proyecto web[4] de Bulma[2] tienes información de cómo instalar todo ello, incluyendo el código fuente del web, los datos de la base de datos y scripts para la instalación del web de bulma en cualquier ordenador.

En los sucesivos ejemplos, obviaremos las marcas <?PHP .. ?>, HTML y BODY

1.2 Comentarios

Los comentarios en PHP se escriben:

- Con `//` o `#` para comentarios de una sóla línea.
- Entre `/*` y `*/` para comentarios de una o más líneas.

Ejemplo:

```
/*  Título: Mi Primera página PHP
    Autor: Yo
*/

// Saludamos
echo("¡Hola Mundo!<BR>");
```

Produce la misma salida que el ejemplo anterior.

1.3 Variables

Todas las variables en PHP empiezan con el caracter dólar "$".

Declaración

Las variables se declaran simplemente inicializándolas:

```
$strCadena = "Hola Mundo";
echo($strCadena);
```

Si intentamos acceder a una variable no inicializada, PHP se quejará. Para evitar ello, existe la función `error_reporting(mask)`.

Tipos

Los tipos básicos de PHP son Integer, Double, String, Array y Object. Las variables booleanas no existen como tales, sino que cualquier valor numérico distinto de 0 o cualquier cadena no vacía se considera `TRUE`

Las variables PHP no tienen un tipo fijo, dependiendo de la última asignación realizada, tienen uno u otro tipo. La función `gettype(nombrevar)` permite obtener el tipo de esa variable en forma de cadena:

```
$variable = "Una cadena";
echo(gettype($variable));
$variable = 0;
echo(gettype($variable));
```

El ejemplo anterior escribe "String" e "Integer" por pantalla.

Las funciones `Is_Double($varname)`, `Is_Array($varname)`, `Is_String($varname)` y `Is_Object($varname)` también nos permiten saber el tipo de una variable.

Cadenas

Las cadenas en PHP se especifican rodeadas por comillas simples o por comillas dobles:

```
$strCadena1 = "Hola Mundo<BR>";
echo($strCadena1);
$strCadena2 = 'Hola Inmundo<BR>';
echo($strCadena2);
```

Hay un matiz de diferencia entre una y otra, que podemos comprobar con este ejemplo:

```
$strMessage = "Hola Mundo";
$strMsgInABottle  = "$strMessage<BR>";
echo($strMsgInABottle);
$strMsgInABottle = '$strMessage<BR>';
echo($strMsgInABottle);
```

Produce una página con el texto

```
Hola mundo
$strMessage
```

Es decir, cuando usamos comillas dobles, las expresiones del estilo `$varname` se sustituyen por el valor de la variable `$varname`, mientras que cuando usamos comillas simples, la cadena no se evalúa y se deja como está.

El operador para concatenar cadenas es el punto ".":

```
$strCadena = "Hola";
$strCadena = $strCadena . "Mundo";
echo($strCadena);
```

Las comillas pueden abarcar más de una línea sin ningún problema:

```
$strConsulta = '
   SELECT *
   FROM
     bul_tbl_noticias
   WHERE
     nombre_autor = \'Alberto\';
';
```

Como vemos, podemos escapar las comillas con la combinación `\'`. De la misma manera, `\n`, `\t` y otros tienen el mismo significado que en C.

Arrays

Los arrays en PHP son bastante potentes y flexibles:

```
$arrValores[0] = 1;
$arrValores[1] = "Una cadena";
echo("En \$arrValores[0] está $arrValores[0] y en " .
   "\$arrValores[1] está $arrValores[1]<BR>");
```

El no poner el subíndice del elemento, hace que el valor asignado se asigne a las siguiente posición libre del array. Los arrays en PHP comienzan en la posición 0, por lo que el anterior código podría escribirse más fácilmente así:

```
$arrValores[] = 1;
$arrValores[] = "Una cadena";
echo("En \$arrValores[0] está $arrValores[0] y en " .
   "\$arrValores[1] está $arrValores[1]<BR>");
```

Otra forma de crear arrays es mediante la construcción `Array()`:

```
$arrValores = Array(1, "Una cadena");
```

```
echo("En \$arrValores[0] está $arrValores[0] y en " .
   "\$arrValores[1] está $arrValores[1]<BR>");
```

Una forma muy conveniente de direccionar elementos de un array es *asociativamente*. En el caso de los arrays *asociativos*, en vez de accederse por índice, se accede por clave o *key* (las claves sí son *case sensitive*, no es lo mismo `$arrValores["a"]` que `$arrValores["A"]`, no es lo mismo):

```
$arrValores["nombre"] = "Tancredo";
$arrValores["Apellidos"] = array("Gómez", "Jiménez");
echo("En \$arrValores[\"nombre\"] está $arrValores[nombre] y " .
   "en \$arrValores[\"Apellidos\"] está " .
   $arrValores["Apellidos"][0] . " y  " .
   $arrValores[Apellidos][1] . "<BR>");
```

Como vemos, manejar arrays multidimensionales es trivial en PHP, basta con añadir los corchetes y el subíndice deseado.

La construcción `Array()` también puede usarse con arrays asociativos:

```
$arrValores=array(
  "nombre" => "Tancredo",
  "Apellidos" => array("Gómez", "Jiménez")
);
```

La construcción `List()` nos permite asignar los valores de un array a una serie de variables de una sola vez:

```
$arrValores=Array(1, "Una cadena", 1.2);
List($nNumber, $strCadena, $fNumber) = $arrValores;
echo("\$nNumber vale $nNumber, \$strCadena vale " .
   "'$strCadena' y \$fNumber vale $fNumber");
```

saca como resultado:

```
$nNumber vale 1, $strCadena vale 'Una cadena' y $fNumber vale 1.2
```

Conversiones

Para convertir una variable de un tipo a otro se emplea el *casting* mediante paréntesis:

```
$strVariable = "5";
$valor = (integer) $strVariable;
```

`$valor` contiene el valor numérico de la variable `$strVariable`.

También podemos emplear la función `SetType($varname, "vartype")` para forzar que la variable `$varname` sea del tipo `vartype`.

De todas formas, PHP es bastante consecuente en cuanto a los tipos, de manera que si sumamos un número a una cadena, esa cadena se convierte en un número:

```
$strCadena="5";
echo('$strCadena es de tipo ' . GetType($strCadena) .
   " y vale $strCadena<BR>");
$strCadena = $strCadena + 5;
echo('$strCadena es de tipo ' . GetType($strCadena) .
   " y vale $strCadena<BR>");
```

produce como resultado

```
$strCadena es de tipo string y vale 5
$strCadena es de tipo integer y vale 10
```

En caso de que concatenemos una cadena con un número, PHP realiza la conversión del número a cadena automáticamente:

```
echo("El número es " . 5 . "<BR>");
```

Produce la salida esperada

```
El número es 5
```

Variables predeclaradas HTTP

PHP tiene toda una serie de variables predeclaradas que tienen que ver con HTML, como:

- `$PHP_AUTH_USER`: Usuario de la autenticación.
- `$PHP_AUTH_TYPE`: Tipo de autorización.
- `$PHP_AUTH_PW`: Contraseña con la que se autenticó el usuario.
- `$HTTP_POST_VARS`: Array con las variables de un form pasadas por el método POST.
- `$HTTP_PUT_VARS`: Array con las variables de un form pasadas por el método PUT.

Aparte de los arrays `$HTTP_PUT_VARS` y `$HTTP_POST_VARS`, podemos acceder a las variables provenientes de forms HTML como $nombrevariable, supongamos el siguiente form:

```
<HTML>
<BODY>
<FORM ACTION="tratar_form.phtml">
<INPUT TYPE="TEXT" NAME="Nombre">
<INPUT TYPE="TEXT" NAME="Apellido[0]">
<INPUT TYPE="TEXT" NAME="Apellido[1]">
<INPUT TYPE="SUBMIT" NAME="btnAceptar" VALUE="Aceptar">
</FORM>
</BODY>
</HTML>
```

Mientras que en la página `tratar_form.phtml` podemos acceder a las variables del form con:

```
Echo("Nombre: $Nombre <BR>
  Apellido1: $Apellido[0] <BR>
  Apellido2: $Apellido[1] <BR>
");
```

Comprobación de declaración

En ocasiones es necesario saber si una variable ha sido inicializada ya (sobre todo si proviene de un form html, por ejemplo), para ello tenemos la función `IsSet($variable)` que nos permite saber si esa variable fue ya inicializada.

Esta función debe ser usada conjuntamente con un `error_reporting(~8)` para que el intérprete no capture el error de intento de acceso a variable no inicializada.

1.4 Constantes

Las constantes en PHP son literales que no comienzan por "$" y que se inicializan con la construcción `define(nomconst)`:

```
define("MAX_CLIENTS", 25);
Echo(MAX_CLIENTS);
```

Las constantes predefinidas `__FILE__` y `__LINE__` nos dan el nombre del fichero y el número de línea actual.

1.5 Operaciones

booleanas, concatenación de cadenas, aritméticas.

1.6 Sentencias de control

Las estructuras de control de PHP son iguales que las de C, con algún que otro añadido.

Condicionales

La estructura de los condicionales es igual que en C:

```
if ($usuario == "Wildfred") {

} elseif ($usuario == "Winifred") {

} else {

}
```

Switch

```
switch ($usuario) {
  case "Wildfred":
  break;
  case "Winifred":
  break;
  default:
}
```

La expresión de selección de rama del case tiene que ser escalar (no objeto o array).

Bucles

En los bucles pueden usarse las instrucciones `break` y `continue` para salir del bucle actual o para avanzar hasta la próxima iteración.

For

```
for ($i=0;i<40;i++) {
   Echo("\$i vale $i<BR>");
}
```

While

```
$bDoExit = 0;
while (!$bDoExit) {
  Echo("Iterando<BR>");
  $bDoExit = 1;
}
do {
  Echo("Iterando<BR>");
  $bDoExit = 1;
} while (!$bDoExit);
```

list, Each (arrays)

Existen dos funciones que combinadas nos permiten iterar fácilmente por todos los elementos de un array:

```
$arrApellidos = array("Pepe" => "Pérez", "Paco" => "Gómez");
while ( list($strNombre, $strApellidos) = each($arrApellidos)) {
   Echo("$strNombre se apellida $strApellidos.<BR>");
```

```
}
```

Todos los arrays mantienen un contador interno (accesible mediante las funciones `current`, `reset`, `next` y `prev`, y la función `each` se encarga de devolver `current` y llamar a `next`. Con la función `list` asignamos la clave y el elemento a las variables `$strNombre` `$strApellidos`, hasta que no queda ningún elemento (`each` devuelve null).

Array_Walk

`Array_Walk` es una función que toma como parámetros un array y una función y aplica esa función a cada elemento del array:

```
function dumpit($elem) {
   Echo("$elem<BR>");
}
$arr = array("Elem1", "Elem2", "Elem3");
Array_Walk($arr,"dumpit");
```

1.7 Funciones

Declaración

Un esqueleto típico de una función es:

```
function outputcol($strCadena, $strColor) {
   // Saca una cadena con el color deseado
   Echo("<FONT COLOR=\"#$strColor\">$strCadena</FONT>");
}
```

Y se llama con:

```
outputcol("Rojo", "FF0000");
outputcol("Verde", "00FF00");
```

Dando el resultado:

```
<FONT COLOR="#FF0000">Rojo</FONT>
<FONT COLOR="#00FF00">Verde</FONT>
```

Parámetros

Parámetros por defecto

Si deseamos que la función por defecto ponga el texto en color azul, por ejemplo, la redefiniríamos de la siguiente manera:

```
function outputcol($strCadena, $strColor="0000FF") {
   // Saca una cadena con el color deseado
   Echo("<FONT COLOR=\"#$strColor\">$strCadena</FONT>");
}
```

y la podríamos llamar con:

```
outputcol("Defecto");
outputcol("Verde", "00FF00");
```

Dando el resultado:

```
<FONT COLOR="#0000FF">Defecto</FONT>
<FONT COLOR="#00FF00">Verde</FONT>
```

Evidentemente, siempre que una función tenga *n* parámetros por defecto, éstos deberán ser los *n* últimos parámetros

declarados.

Parámetros por referencia

En PHP por defecto los parámetros se pasan por valor, es decir, que si los modificamos dentro de la función, a la vuelta de la función las variables pasadas como parámetro no se modificaron (a dichas variables se les denomina *parámetros actuales* de la función).

Si deseamos que las variables del llamante sean modificadas (los mencionados *parámetros actuales*), hay que pasar los parámetros por referencia:

```
function Concatena(&$strDest, $strSrc) {
   // Esta función concatena dos cadenas y las devuelve
   // en la primera cadena pasada
   // $strDest se le pasa como parámetro por referencia.
   $strDest = $strDest . $strSrc;
   // Como $strSrc no se pasa por referencia, la siguiente
   // instrucción no afecta al parámetro actual
   $strSrc = "";
}
$strOrigen = "Mundo";
$strDestino = "Hola ";
Echo("Origen es $strOrigen y destino es $strDestino<BR>");
Concatena($strDestino, $strOrigen);
Echo("Origen es $strOrigen y destino es $strDestino<BR>");
```

Que ofrece el resultado:

```
Origen es Mundo y destino es Hola
Origen es Mundo y destino es Hola Mundo
```

Como se ve, para pasar un parámetro por referencia, basta con poner "&" delante del nombre del parámetro en la declaración de la función, es decir, poner "&" delante del nombre del *parámetro formal*.

También se puede pasar un parámetro por referencia aunque en la función no esté declarado como tal, anteponiendo el ampersand "&" al parámetro actual (al invocar la función).

Variables en funciones

Variables locales

Para definir una variable local, simplemente se asigna un valor a la variable:

```
function Iva($fValue) {
   // $fIVA es una variable local
   $fIVA = $fValue * 0.16;
   Echo("El IVA de $fValue es $fIVA<BR>");
}
Iva(2350);
```

Variables estáticas

Si queremos que la variable local conserve el valor de invocación a invocación de la función, basta declararla como estática:

```
function Counter() {
  static $count = 0;
  $count2 = 0;
  $count2++;
  Echo("Count vale $count y count2 vale $count2<BR>");
  $count++;
}
Counter();
```

```
Counter();
Counter();
```

muestra en la página:

```
Count vale 0 y count2 vale 1
Count vale 1 y count2 vale 1
Count vale 2 y count2 vale 1
```

La inicialización de la variable sólo tiene lugar la primera vez que `Counter` es invocada.

Acceso a variables globales

Este es uno de los puntos en los que PHP se diferencia de C y es un punto importante y causa de bastantes quebraderos de cabeza para el primerizo en PHP.

Para acceder a una variable global desde dentro de una función es **imprescindible** declararla dentro de la función como `global $variable`, ya que de otro modo PHP pensará que se desea hacer referencia a una variable local:

```
function TouchGlobal() {
   global $strCadena;
   $strCadena = "¡Tocada!";
   $nValue = 7;

   Echo("Dentro de TouchGlobal ahora \$strCadena vale " .
      $strCadena " . "y \$nValue vale $nValue<BR>");
}
$strCadena = "Hola mundo";
$nValue = 4;
Echo("\$strCadena vale $strCadena y \$nValue vale $nValue<BR>");
TouchGlobal();
Echo("\$strCadena ahora vale $strCadena y \$nValue sigue ".
   "valiendo $nValue<BR>");
```

Como se ve, no es necesario que la variable global esté en el fichero *físicamente* delante de la función, basta con que haya sido inicializada antes de llamar a la función. Los cambios realizados a una variable global dentro de una función, permanecen cuando se vuelve de la función.

Otra forma de acceder a las variables globales es mediante una indexación asociativa del array `$GLOBALS`.

Devolución de un valor

Para devolver un valor se emplea la cláusula `return`.

```
function Factorial($nValue) {
   if ($nValue <= 1) {
      return 1;
   } else {
      return Factorial($nValue-1)*$nValue;
   }
}
$nNumber = 7;
Echo("El factorial de $nNumber es " . Factorial($nNumber));
```

En PHP las funciones pueden llamarse a sí mismas (recursivas), e incluso se pueden declarar funciones dentro de funciones o clases dentro de funciones.

1.8 Funciones interesantes

Printf, sprintf

El formato de llamada de estas funciones es exactamente igual que en C.

Echo

`Echo` en realidad no es una función sino una construcción del lenguaje, por ello se puede poner tanto con paréntesis como sin paréntesis.

Evaluación de variables con `Eval`

La función `Eval($strExpr)` permite evaluar la expresión `$strExpr`, de manera que si contiene código PHP válido, éste será interpretado. Esto permite cosas muy flexibles como por ejemplo callbacks:

```
function mycallback($strParam) {
   Echo("Dentro del callback<BR> con parámetro $strParam");
}

function myfunc($fnCallback) {
   // Creamos una tabla
   Echo("<TABLE><TR><TD>Callback1:</TD><TD>");
   // Llamamos al callback
   Eval($fnCallback);
   // Cerramos la tabla
   Echo("</TD></TR><TABLE>");
}
$strCode = 'global $strParam; mycallback($strParam);';
$strParam = "Soy el parámetro del callback";
myfunc($strCode);
```

Podemos emplear otra técnica para hacer callbacks más sencillos con:

```
function mycallback() {
  Echo("Dentro del callback.<BR>");
}
$strCallback = "mycallback";
// Llamamos al callback
$strCallback();
```

Variables variables

No, el título no está equivocado, las *variables variables* son formas de *indireccionamiento* a la hora de referirse a variables.

Si tenemos dos variables `$strVarName` y `$nValue` y hacemos que `$strVarName` contenga la cadena `"nValue"`, al referirnos a `$$strVarName` (nótese el doble dólar `"$$"`) nos estamos refiriendo a la variable que tiene como nombre el contenido de `$strVarName`, es decir, a `$nValue`.

Lo anterior plasmado en un ejemplo sería:

```
$nValue = 5;
$strVarName = "nValue";
Echo("Mostrando el valor de $strVarName: $$strVarName.<BR>");
$$strVarName = 5;
Echo("Ahora $strVarName vale $$strVarName.<BR>");
```

En los casos en los que haya ambigüedad, puede emplearse el agrupador `{}`, por ejemplo, para `$$myarray[0])`, podríamos referirnos:

1. Al primer elemento del array que tiene como nombre el contenido de la variable `$myarray`.
2. A la variable que tiene como nombre el contenido de `$myarray[0]`.

Si empleásemos el operador de agrupación para evitar la ambigüedad, el primer caso lo escribiríamos `${$myArray}[0]` y `${$myArray[0]}` en el segundo.

Control de errores Error_Reporting

Mediante la función `Error_Reporting(mask)` se pueden limitar los errores que captura el intérprete de PHP y ante los que aborta la ejecución del programa de entre los siguientes:

- E_ERROR (1)
- E_WARNING (2)
- E_PARSE (4)
- E_NOTICE (8)
- E_CORE_ERROR (16)
- E_CORE_WARNING (32)

Por ejemplo, con:

```
Error_Reporting(E_NOTICE | E_WARNING);
```

Se hará que el intérprete de PHP no capture los errores distintos de NOTICEs o de WARNINGs, para que podamos tratarlos nosotros.

Es habitual deshabilitar `E_NOTICE` cuando se emplea la función `IsSet` para comprobar si se inicializó una variable, ya que si no se hace así, y la variable no fue inicializada dará un error y el intérprete abortará la ejecución:

```
// Deshabilitamos notices
$nOldMask = Error_Reporting(~E_NOTICE);
if (IsSet($btnAlta)) {
   // Pulsó el botón de alta en el form
   Echo("Dando de alta el elemento solicitado.<BR>");
} elseif (IsSet($btnBaja)) {
   // Pulsó el botón de baja en el form
   Echo("Dando de baja el elemento solicitado.<BR>");
}
// Restauramos la máscara de error antigua
Error_Reporting($nOldMask);
```

Opcionalmente puede deshabilitarse la detección de errores para una sola sentencia anteponiendo la arroba "@" a la sentencia.

Die, exit

Se emplean para terminar la ejecución del script abruptamente.

- `Die($msg)` muestra el mensaje `$msg` antes de salir.
- `Exit()` sale de la ejecución del script.

1.9 Librerías

Include

Incluye el fichero cada vez, por si se desea meter esta instrucción en un bucle:

```
// Array con los nombres de las librerías
$arrLibraries = Array("include1.php3", "include2.php3",
  "include4.php3");
// Iteramos por cada nombre de librería
while (list($nIndex, $strLibname) = each(arrLibraries)) {
   // Incluímos esta librería
   include($strLibname);
}
```

Require

Se reemplaza la instrucción `Require("nomfich")` por el fichero. Es lo que se suele usar normalmente para incluir librerías externas.

1.10 Clases (Objetos)

Para declarar un objeto en PHP usamos la construcción `class`, las variables de instancia se declaran anteponiendo `var` al nombre y las funciones miembro se definen dentro del bloque de la clase como funciones normales y corrientes. Un constructor se define como una función con el mismo nombre que el objeto.

Para referirnos al propio objeto o a variables miembro, empleamos `$this->`.

Un ejemplo de todo lo anterior sería:

```
class MyObject {
   var $nCount = 0;
   var $strName;
   function MyObject($strName="Nonamed") {
      $this->strName = $strName;
   }
   function AddCount($nDelta) {
      $this->nCount += $nDelta;
   }
   function GetCount() {
      return $this->nCount;
   }
}
```

Como vemos, en las funciones miembro podemos emplear parámetros por defecto y cualquier cosa que emplearíamos en una función normal.

Para usar este objeto, haríamos:

```
$myObj = new MyObject("Pepito");
Echo("Count vale " . $myObj->GetCount() . "<BR>");
$myObj->AddCount(5);
Echo("Ahora count vale " . $myObj->GetCount());
```

Mucho cuidado con las mayúsculas y minúsculas en los nombres de las variables de tipo objeto, porque el PHP tiende a ser *case sensitive* con éstas.

2. Librerías de Bulma

A lo largo de todos los ficheros PHP de Bulma, se sigue la notación húngara (bueno, técnicamente es un derivado denominado notación checoslovaca, creo), es decir, anteponer el prefijo de tipo al nombre de la variable.

Por ejemplo, la variable de tipo cadena que almacene el nombre de usuario, podría ser

---

`strUser`

(

`str`

---

es el prefijo de variables de tipo cadena). Otros prefijos habituales son:

- `n` para enteros: `nCounter`, `nIndex`...
- `lrc` para instancias de la clase `LoginRec`.
- `rs` para instancias de la clase `RecordSet`: `rsUsuarios`, `rsNoticias`...

- `con` para instancias de la clase `Connection`: `conBulma`...

2.1 recordset.php3

Este fichero contiene objetos para abstraer de la base de datos en concreto a usar. De esta manera, si en algún punto del desarrollo se desea cambiar de base de datos (mySQL...), se podrá hacer cambiando únicamente este fichero.

Actualmente este fichero implementa las funciones de acceso únicamente para la fantástica base de datos PostgreSQL[5].

Connection

Connection($strDBName, $bPersistent)

Crea la conexión con la base de datos dada.

Parámetros

- *$strDBName* Nombre de la base de datos a la que conectar.
- *$bPersistent* Si usar conexiones persistentes o no (reciclar una conexión anterior, cosas del PostgreSQL).

Notas

Es importante cerrar la conexión llamando a `Close()` cuando se haya acabado de emplear.

Close()

Cierra la conexión previamente abierta.

Exec($strQuery)

Ejecuta la consulta `$strQuery` sobre esta conexión.

Parámetros

- *$strQuery* Cadena cno la consulta SQL a ejecutar.

Valor de retorno

Devuelve un `RecordSet` con el resultado de la consulta.

Notas

Es importante llamar al método `Close` del `RecordSet` cuando se haya acabado de usar éste.

RecordSet

Los `RecordSet` almacenan *conjuntos de registros*, resultado de una consulta a una base de datos.

RecordSet($con)

Parámetros

- *$con* `Connection` sobre la que operará este `RecordSet`

Notas

Cuando se acabe de usar el `RecordSet` se debe llamar a `Close` para liberar los recursos de este `RecordSet`.

Open($strQuery)

Es equivalente al método `Exec()`.

GetError()

Devuelve la cadena del último error ocurrido en la última llamada a `Exec()` u `Open()`, o 0 si no se produjo ningún error.

Exec($strQuery)

Ejecuta la consulta SQL dada.

Parámetros

- *$strQuery* Consulta SQL a ejecutar.

Notas

Antes de llamar al próximo `Exec()` u `Open` es necesario cerrar el `RecordSet` actualmente abierto.

MoveNext()

Recupera los datos del siguiente registro de este `RecordSet`.

MoveTo($nRow)

Se posiciona sobre un registro concreto del RecordSet.

Parámetros

- *$nRow* Número de fila del `RecordSet` sobre la que posicionarse.

Eof()

Valor de Retorno

Devuelve TRUE si se intentó avanzar más allá de la última fila del `RecordSet`.

Notas

Un `RecordSet` vacío devuelve `Eof()` a TRUE nada más ejecutarse.

GetActRow()

Devuelve el número de fila actual, es decir, el número de registro que está siendo visualizado actualmente en el campo `fields`.

GetNumRows()

Devuelve el número de registros de este `RecordSet`.

Valor de Retorno

- -1 Si el `RecordSet` no ha sido ejecutado aún o la última consulta devolvió error.
- 0 Si el `RecordSet` está vacío.
- Mayor que 0 Si el `RecordSet` es válido y tiene registros (en cuyo caso devuelve el número de registros del `RecordSet`).

Close()

Cierra el `RecordSet` y desaloja los recursos asignados.

Notas

Después de cada llamada con éxito a `Open()` o a `Exec()` tiene que haber una llamada correspondiente a `Close()`. Después de llamar a `Close()`, pueden realizarse nuevas llamadas a `Exec()` y `Open()` sobre este `RecordSet`.

fields

Array asociativo que se indexa con los nombres de los campos de la consulta realizada y devuelve los valores para el registro actual del `RecordSet`.

2.2 loginrec.php3

Esta librería contiene el objeto que soporta la autenticación HTTP mediante autenticación básica y contraste de la contraseña frente a una base de datos.

LoginRec

LoginRec()

Inicializa el objeto `LoginRec`, tomando las variables HTTP de información de login.

Authenticate($con)

Fuerza la autenticación de la conexión HTTP consultando la base de datos asociada al objeto `Connection` pasado como parámetro.Si la autenticación no es correcta, devuelve una página de error.

Matches($strUser, $strPassword)

Devuelve TRUE si el objeto `LoginRec` coincide con el usuario `$strUser` y la contraseña `$strPassword`.

GetUser

Devuelve el usuario HTTP.

GetIdUser

Devuelve el identificador del usuario en la base de datos.

GetPassword

Devuelve la contraseña HTTP.

IsLogged

Devuelve TRUE si se ha introducido información de usuario y contraseña.

IsAuthenticated

Devuelve TRUE si se ha introducido información de usuario y coontraseña y estos se han validado contra la base de datos, mediante una llamada anterior a `Authenticate()`.

$IrcLoginRec

Objeto de tipo `LoginRec` creado al incluir esta librería, que contiene la información de login de la conexión HTTP actual.

Lista de enlaces de este artículo:

1. mailto:wildfred@teleline.es
2. http://bulma.lug.net
3. http://www.php.net/
4. http://bulma.lug.net/body.phtml?nIdNoticia=14
5. http://www.postgresql.org

E-mail del autor: wildfred _ARROBA_ teleline.es
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=215