Introducción

Si está familiarizado con otros lenguajes de programación como C o Java, ya conocerá algunos mecanismos que permiten manejar errores y excepciones. Una de las nuevas características que PHP 5 incluye, es un objeto de manejo de excepciones con una sintaxis muy similar a Java.
El manejo de excepciones es una herramienta muy potente que podrá apreciar una vez que comprenda el concepto y haga buen uso del objeto en su código. Estas funciones permiten depuíar condiciones de error, recuperar situaciones inesperadas y presentar *interfaces* limpias de errores.

Errores y Excepciones

Lo primero que tenemos que hacer es no pensar en una excepción como si fuera un error. Una excepción es una condición que se experimenta en un programa inesperadamente y no puede ser manejada por el código que hemos escrito. Generalmente no causan la parada del programa, pero sí es posible detectarla para que continúe normalmente.
Las excepciones, si son tratadas correctamente, pueden dar más fiabilidad a su aplicación y librarle de muchos dolores de cabeza. Aunque, si no son correctamente utilizadas, pueden crear más problemas que resolverlos. Debe tomar un tiempo para determinar si necesita crear sus propios métodos de manejo de errores y, si es así, verá que a la larga resulta más gratificante. Como ejemplo vamos a ver una pieza de código que simula la recepción de un identificador de usuario y su posible comprobación en una base de datos. Tal y como puede ver, se comprueba que los tres primeros caracteres sean "usr", que el tamaño en caracteres sea al menos de 9 y que exista en una base de datos:

```
<?php
//extraemos el usuario por GET
$usuario_id = $_GET['usuario_id ' ] ;
//Si existe el usuario escribimos el mensaje CORRECTO
if  (!usuario_valido($usuario_id))  {
   echo "ERROR:: El usuario $usuario_id no existe";
} else {
   echo "CORRECTO:: El usuario $usuario_id existe";
}

function usuario_valido($usuario_id)  {
```

```
    $cadena = "usr";
    //Comprobaciones de usuario
    if   ( (strstr ($usuario_id,$cadena) == false)  [[
          (strpos($usuario_id,$cadena) != 0)) {
               return false;
          }

          if (strlen ($usuario__id) < 9) {
               return false;
          }
          if (existe__base_datos ($usuario_id) ) {
               return true;
          } else {
               return false;
          }
}
?>
```

Como puede ver, cualquier condición que está dentro de la función que valida los usuarios puede dar un resultado negativo y sólo uno de ellos dará un resultado positivo; si el usuario existe en la base de datos.

Las excepciones permiten distinguir entre distintos tipos de errores y manejarlos dependiendo de su naturaleza.

La clase Exception

La clase Exception se ha creado en PHP 5 para manejar las anomalías producidas. En vez de utilizar comprobaciones que evalúan si es falso o verdadero una determinada condición, se utiliza una instancia de la clase Exception, que captura los errores y permite mostrar un mensaje de error personalizado.

```
<?php
//extraemos el usuario por GET
$usuario__id = $_GET ['usuario_id'] ;
//Si existe el usuario escribimos el mensaje CORRECTO
try {
   if ( lusuario_valido($usuario_id)) {
        echo "ERROR:: El usuario $usuario_id no existe";
   } else {
        echo "CORRECTO:: El usuario $usuario_id existe";
   }
} catch(Exception $ex) {
   //Devuelve el mensaje de error
   $mensaje = ($ex->qetMessage() ) ;
   echo $mensaje;
```

```
}


function usuario_valido($usuario_id)  {
   $cadena = "usr";
   //Comprobaciones de usuario
   if    ( (strstr ($usuario_id,$cadena)  == false)  |
         (strpos($usuario_id, $cadena)  != 0))  {
              throw   new Exception  ("$usuario_id no contiene el
prefijo");
         }

         if  (strlen($usuario_id)  <  9)  {

              throw new Exception  ("El  tamaño  de  $usuario_id es
menor de 9 caracteres");
         }
         if  (!existe_base_datos($usuario_id) )  {
              return false;
         } else {
              return false;
         }
}
```

Como puede ver, las excepciones se crean de una manera algo diferente. En el momento en el que se escribe throw new Exception, los errores que no se controlen con el flujo normal del programa son utilizados como excepciones separadas del resto de la aplicación.

Bloque Try / Catch

Las excepciones son controladas por el bloque try/catch. Todo el código que escriba y que pueda generar un error, se incluirá dentro de una estructura try. Si dentro del bloque try se encuentra un error, la ejecución del código se para y se pasa el control al bloque catch. Este bloque se consuita para encontrar *la* excepción que ha producido *el* error, de acuerdo con el código que hemos escrito.

Una de las cosas más convenientes de utilizar excepciones es que puede mostrar tanta información, o tan poca, como quiera. El objeto Except ion tiene varios métodos para manejar sus propios errores.

El código siguiente muestra cómo utilizar throw e inmediatamente capturar la excepción. Podemos capturar también algunos parámetros útiles:

```
<?php
try {
   throw new Exception("Error de sintaxis");
} catch (Exception $ex){
   //Mensaje de error
   $mensaje = ($ex->getMessage()) ;
   //Código de error configurable
   $codigo = ($ex-igetCode());
   //Fichero del error
   $fichero = ($ex->getFile() ) ;
   //Linea donde se ha producido el error
   $linea = ($ex->getLine());

   echo "Error número $codigo: $mensaje en el fichero $fichero en
   la linea $linea";
}
?>
```

El mensaje de error que se muestra es muy típico en PHP. Como es totalmente configurable, puede generar su propio formato de Mensaje de Error, que puede ver en la figura 17.2.

Aunque en este ejemplo, el código que captura la excepción sólo tiene una línea, podrá incluir dentro del bloque varias líneas o funciones dentro que actuarán de la misma forma.

Heredar de (a clase Exception

PHP 5 permite definir sus propias clases que heredan de la clase Exception. Puede hacerlo de la siguiente forma:

```
<?php
class Excepción extends Exception
{
   function___construct($mensaje)
   {
        parent::Exception($mensaje);
   }
}
?>
```

El siguiente ejemplo muestra cómo crear clases propias que manejan errores a partir de la clase Exception:

```
<?php
$usuario_id = $_GET['usuario_id'];
try {

   if f J usuario_valido ($usuario__id) ) [
        echo "ERROR:: El usuario $usuario id no existe";
```

```
   } else {
        echo "CORRECTO:: El usuario $usuario_id existe";
   }
} catch(MiExcepcion $ex) {
   $cadena = "usr";
   if   ((strpos($usuario_id,$cadena) == false) |]
        (strpos($usuario_id,$cadena) != 0)) {
             $mensaje = $ex->getMessage();
   }
}
catch(OtraExcepcion $ex) {
   if (strlen($usuario_id) < 9) {
        $mensaje = $ex->getMessage();
   }
}
//Definición de las clases que heredan de Exception
class MiExcepcion extends Excepción
{
   function___construct($mensaje)
   {
        parent::Exception($mensaje);
   }
}
class OtraExcepcion extends Excepción
{
   function___construct($mensaje)
   {
        parent::Exception($mensaje);
   }
}
echo $mensaje;
function usuario_valido($usuario_id) {
   $cadena = "usr";
   if   ((strpos($usuario_id,$cadena) == false) ||
        (strpos($usuario_id,$cadena) != 0)) {
             throw new MiExcepcion("Usuario $usuario_id no
contiene el prefijo " ) ;
        }
   if (strlen($usuario_id) < 9) {
        throw new OtraExcepcion("Usuario tiene menos de 9
caracteres");
   }
}
?>
```

Limitaciones de PHP 5

El objeto de excepciones es totalmente nuevo en PHP 5. Debido a esto, algunas funcionalidades creadas en otros lenguajes de programación, como

Java, no han sido todavía implementadas; por ejemplo el uso de los métodos finally() y throws(). Los errores comunes de PHP, que se muestran en el navegador, no son mapeados a excepciones automáticamente. Esta funcionalidad será implementada en versiones posteriores de PHP. Los errores que no pueden ser controlados por Exception pueden manejarse con otras técnicas de programación.

Control de errores sin excepciones

Si ya conoce algunas versiones de PHP, sabrá que hay multitud de funciones capaces de controlar errores, incluidos los errores nativos de PHP, controladores personalizados de error y los errores de usuario.

Errores nativos de PHP

PHP genera 3 tipos de error, dependiendo de su seriedad:

- **Notice:** No son errores severos y no crean problemas complejos. Por defecto, no se muestran en pantalla cuando suceden, a menos que digamos lo contrario en el archivo de configuración php.ini.
- **Warning:** Alguna parte del código ha causado un error, pero la ejecución continúa.
- **Fatal error:** Un problema serio que hace abortar la ejecución del programa.

Cada tipo de error es representado por una constante diferente: E_USER_NOTICE, E_USER_WARNING y E_USER_ERROR. Puede elegir el nivel de error que quiere mostrar en pantalla con la función error_reporting().

```
<?php
//Mostrar errores fatales
error_reporting(E_USER_ERROR);
//Mostrar advertencias y notificaciones
error_reporting(E_USER_WARNING | E_USER_NOTICE);
//Mostrar todos los errores
error_reporting(E__USER__ALL);
?>
```

El manejo de errores queda limitado la mayoría de veces a los del tipo warning. Los errores fatales se consideran tan críticos que no pueden controlarse y la ejecución del programa queda suspendida.

Controladores de error

Cuando suceda un error que no tenga contemplado en su aplicación, pueden aparecer mensajes en la pantalla que no interesa mostrar. La solución es crear su propia función para controlar y mostrar errores.

```
<?php
function errores($error,{mensaje,$fichero,$linea) {
   echo "<b>: : ERROR : : </bxbr>" ;
   echo "Sentimos comunicarle que se ha producido un error";
   echo "Tipo de error: $error: $mensaje en $fichero en la linea
   $linea";
}
?>
```

Con estas líneas hemos creado una función que muestra el error producido, pero con un diseño acorde a nuestra página Web. Puede mostrar tanta información como quiera. El siguiente paso es llamar a la función `set_error_handler ()` para que PHP sepa que nuestra función `errores ()` se va a encargar a partir de ahora de gestionar los posibles fallos en el flujo del programa.

```
<?php
set_error_handler("errores");
?>
```

Con el código anterior, todos los errores producidos del nivel que tenga permitido (fatal error, warning o notice) serán enviados a la función errores () para mostrar su mensaje particular.

Errores de usuario con trigger_error()

Desde PHP 4, puede usarse una función para identificar los errores producidos por los usuarios, muy parecida a la función de PHP 5 throw () . La función trigger_error () genera un error del tipo que quiera (fatal error, warning o notice) y será gestionado por PHP o por su propio gestor de errores.

```
<?php
error_reporting(E_ALL);
function errores($error,$mensaje,$fichero,$linea) {
        echo "<b>: :ERROR: :</bxbr>";
        echo "Sentimos comunicarle que se ha producido un error";
        echo "Tipo de error: $error: $mensaje en $fichero en la
        línea $ linea";
}
set_error_handler("errores");

if (1 != 0 ) {
        trigger_error(" Esto es falso ",E_USER_NOTICE);
}
?>
```

Es mejor utilizar siempre la función trigger_error ( ) con set_error_handler () . El error que creemos puede ser de cualquiera de los 3 tipos existentes en PHP. En este caso, generamos un error notice para indicar que la condición es falsa.

Depuración de errores

El manejo de errores puede ahorrar esfuerzos a la hora de depurar el funcionamiento óptimo de su código, pero también puede crear algunos dolores de cabeza si empieza a suprimir algunos fallos.

La funciónerror_log () crea un archivo donde se irán almacenando las ocurrencias de su código. Con el siguiente ejemplo podrá capturar todos los errores producidos en el programa, para después hacer un estudio de lo que ha pasado.

```
<?php
```

```
try {
   if ($usuario != "Luis") {
         throw new Exception("El usuario no existe");
   }
}
catch (Exception $ex) {
        $mensaje = ($ex->getMessage() ) ;
        $codigo = ($ex->getCode() ) ;
        $fichero = ($ex->getFile());
        $linea = ($ex->getLine());
        $log_mensaje =date("H:i d-m-Y") . " Error:: $codigo en
$fichero en la línea $linea. Error $mensaje";
   error_log($log_mensaje,3,"/Users/luis/Sites/error.log") ;
   echo $log_mensaje;
}
?>
```

Nota:

Los ficheros log *son muy utilizados en los Sistemas Operativos para mantener un registro de incidencias. Los ficheros* log *guardan desde el número de veces que ha entrado un usuario, hasta los errores que se han producido en el inicio del sistema.*

La función error_log() puede pasar como segundo parámetro un número entero que identifica el tipo de error y qué hacer con él.

- 0: Se utiliza el mecanismo del sistema operativo.
- 1: Envía el error a una dirección de correo especificada como cuarto parámetro. (Debe formar parte de una cabecera).
- 2: Envía el error a través de una sesión de depuración de PHP (Debe estar habilitado el modo depuración)
- 3: Envía el error a un fichero.

Resumen

Como puede darse cuenta, la mayoría de los programas incluidos en el libro no tienen ningún tipo de control de errores. He preferido evitarlos para ganar en claridad y no tener que empezar explicando los bloques Try / Catch desde el principio. A partir de ahora siéntase libre para modificar todos los *scripts* a su antojo para que sean más fiables y seguros.

El desarrollo de las posteriores versiones de PHP permitirá manejar los errores más fácilmente. El futuro de las excepciones no ha hecho más que empezar con la versión 5 de PHP, y se esperan mejoras suculentas con respecto a las anteriores. Pese a esto, muchas librerías ya incluyen sus propios controles de error como la de MySQL o PEAR.