Introducción

La gestión de errores es la parte de la programación que se encarga de prever o tratar los inevitables errores que ocurren cuando se desarrolla una aplicación. En PHP, unas pocas funciones facilitan esta tarea.

> **Amplía esta información**
>
> Recuerda que dispones de toda la información respecto a este y otros temas en la documentación oficial de PHP.

La función `die()`

Por defecto, cuando ocurre un error en PHP este muestra el mensaje propio del intérprete y su línea de error. Por ejemplo:

```php
<?php
    $a=5;
    $b=0;
    echo "<p>Voy a intentar dividir por cero</p>";
    $resultado=$a/$b;
    echo resultado;
?>
```

Genera el siguiente error:

```
Voy a intentar dividir por cero
Warning: Division by zero in C:\xampp\htdocs\index.php on line 5
```

Sin embargo, si se añade una simple comprobación previa, la función `die()` permite establecer el mensaje de error y, acto seguido, **detener la ejecución del script**:

```php
<?php
    $a=5;
    $b=0;
    echo "<p>Voy a intentar dividir por cero.</p>";
    if($b!=0) {
        $resultado=$a/$b;
        echo $resultado;
    } else {
        die("<p>No es posible dividir por cero</p>");
    }
    echo "<p>Esto no se ejecutará nunca</p>";
?>
```

Salida obtenida:

```
Voy a intentar dividir por cero.

No es posible dividir por cero
```

Manejo de errores

PHP proporciona un conjunto de funciones que permiten simplificar y mejorar la gestión de errores, personalizando los mensajes de error e incluso enviando correos electrónicos o mensajes a otras máquinas.

Constantes predefinidas

Las siguientes constantes son utilizadas para la gestión de errores:

Valor	Constante	Descripción	Nota
1	E-ERROR (integer)	Errores Fatales en tiempo de ejecución. Éstos indican errores que no se pueden recuperar, tales como un problema de asignación de memoria. La ejecución del script se interrumpe.	
2	E-WARNING (integer)	Advertencias en tiempo de ejecución (errores no fatales). La ejecución del script no se interrumpe.	
4	E-PARSE (integer)	Errores de análisis en tiempo de compilación. Los errores de análisis deberían ser generados únicamente por el analizador.	
8	E-NOTICE (integer)	Avisos en tiempo de ejecución. Indican que el script encontró algo que podría señalar un error, pero que también podría ocurrir en el curso normal al ejecutar un script.	

Valor	Constante	Descripción	Nota
16	E-CORE-ERROR (integer)	Errores fatales que ocurren durante el arranque incial de PHP. Son como un E-ERROR, excepto que son generados por el núcleo de PHP.	
32	E-CORE-WARNING (integer)	Advertencias (errores no fatales) que ocurren durante el arranque inicial de PHP. Son como un E-WARNING, excepto que son generados por el núcleo de PHP.	
64	E-COMPILE-ERROR (integer)	Errores fatales en tiempo de compilación. Son como un E-ERROR, excepto que son generados por Motor de Script Zend.	
128	E-COMPILE-WARNING (integer)	Advertencias en tiempo de compilación (errores no fatales). Son como un E-WARNING, excepto que son generados por Motor de Script Zend.	
256	E-USER-ERROR (integer)	Mensaje de error generado por el usuario. Es como un E-ERROR, excepto que es generado por código de PHP mediante el uso de la función de PHP trigger-error().	
512	E-USER-WARNING (integer)	Mensaje de advertencia generado por el usuario. Es como un E-WARNING, excepto que es generado por código de PHP mediante el uso de la función de PHP trigger-error().	
1024	E-USER-NOTICE (integer)	Mensaje de aviso generado por el usuario. Es como un E-NOTICE, excepto que es generado por código de PHP mediante el uso de la función de PHP trigger-error().	

Valor	Constante	Descripción	Nota
2048	E-STRICT (integer)	Habilítelo para que PHP sugiera cambios en su código, lo que asegurará la mejor interoperabilidad y compatibilidad con versiones posteriores de PHP de su código.	Desde PHP 5 pero no incluidoen E-ALL hasta PHP 5.4.0
4096	E-RECOVERABLE-ERROR (integer)	Error fatal capturable. Indica que ocurrió un error probablemente peligroso, pero no dejó al Motor en un estado inestable. Si no se captura el error mediante un gestor definido por el usuario (vea también set-error-handler()), la aplicación se abortará como si fuera un E-ERROR.	Desde PHP 5.2.0
8192	E-DEPRECATED (integer)	Avisos en tiempo de ejecución. Habilítelo para recibir avisos sobre código que no funcionará en futuras versiones.	Desde PHP 5.3.0
16384	E-USER-DEPRECATED (integer)	Mensajes de advertencia generados por el usuario. Son como un E-DEPRECATED, excepto que es generado por código de PHP mediante el uso de la función de PHP trigger-error().	Desde PHP 5.3.0
32767	E-ALL (integer)	Todos los errores y advertencias soportados, excepto del nivel E-STRICT antes de PHP 5.4.0.	32767 en PHP 5.4.x, 30719 en PHP 5.3.x, 6143 en PHP 5.2.x, 2047 anteriormente

Así, puedes usar la instrucción `switch` para evaluar el valor de una variable y actuar en consecuencia:

```
1  <?php
2  function printStringError($n_error,$string_error){
```

```
    switch($n_error){
        case E_ERROR:
            echo "Erro genérico: ".$string_error;
        case E_WARNING:
            echo "Advertencia: ".$string_error;
        default:
            echo "Otro tipo de error: ".$string_error;
    }
}
?>
```

La función de gestión de errores

Por defecto, PHP dispone de una función de gestión de errores propia. Con la función `set_error_handle` es posible especificar una función nueva. Para el siguiente ejemplo se utiliza la función descrita en el anterior apartado:

```
<?php
function printStringError($n_error,$string_error){
        switch($n_error){
            case E_ERROR:
                echo "Erro genérico: ".$string_error;
            case E_WARNING:
                echo "Advertencia: ".$string_error;
            default:
                echo "Otro tipo de error: ".$string_error;
        }
}

set_error_handler("printStringError");
?>
```

Así, cada vez que se genere un error, se gestionará a través de la función `printStringError`.

> **No todos los tipos de errores pueden ser gestionados por una función propia**
>
> Los siguientes tipos de errores no pueden ser manejados con una función definida por el usuario: E_ERROR, E_PARSE, E_CORE_ERROR, E_CORE_WARNING, E_COMPILE_ERROR, E_COMPILE_WARNING, y la mayoría de E_STRICT ocasionados en el archivo desde donde se llamó a set_error_handler()[1].

La función `set_error_handler` puede recibir un segundo parámetro, que será un entero representado por alguna de las constantes vistas anteriormente. Así, se podrá definir una función de gestión de errores u otra en función del tipo de error:

```php
<?php
function printStringError_E_WARNING($n_error,$string_error){
    echo "Estoy recibiendo un warning";
}

function printStringError_E_ERROR($n_error,$string_error){
    echo "Estoy recibiendo un error";
}

set_error_handler("printStringError_E_WARNING",E_WARNING);

set_error_handler("printStringError_E_ERROR",E_ERROR);
?>
```

El disparador de errores

El programador puede definir cuándo se dispara un error. Es decir: si un usuario intenta, por ejemplo, dividir por cero, el programador puede generar un error en dicho momento mediante la función `trigger_error`:

```php
<?php
function dividir($a, $b){
    if($b==0) {
        trigger_error("No es posible dividir por cero");
        die("Finalizando la ejecución");
    }
    return($a/$b);
}

dividir(5,0);
?>
```

Generará la siguiente salida:

```
Notice: No es posible dividir por cero in C:\xampp\htdocs\index.php on line 4
Finalizando la ejecución
```

Registro de errores

Por defecto, PHP registra los errores bien en el sistema de registro del servidor, bien en un archivo, según se defina en el archivo `php.ini` (directiva `error_log`).

La función `error_log()` permite enviar un mensaje de error al registro de errores del servidor, a un fichero, o a una dirección de correo electrónico[2]:

```php
<?php
// Enviar una notificación al registro del servidor si no podemos
// conectarnos a la base de datos.
if (!Ora_Logon($username, $password)) {
    error_log("¡La base de datos de Oracle no está disponible!", 0);
}

// Notificar al administrador mediante un email si agotamos FOO
if (!($foo = allocate_new_foo())) {
    error_log("Problema serio, nos hemos quedado sin FOOs!", 1,
              "operator@example.com");
}

// otra manera de llamar a error_log():
error_log("¡Lo echaste a perder!", 3, "/var/tmp/my-errors.log");
?>
```

Definición de `error_log()`

Esta es la documentación completa de `error_log()`:

`error_log ( string $message , int $message_type = 0 , string $destination = ? , string $extra_headers = ? ) : bool`

Parámetros

`message`

El mensaje de error que debería ser registrado.

`message_type`

Indica dónde debería ir el error. Los tipos de mensaje posibles son:

Tipos de registro de error_log():

0: `message` es enviado al registro del sistema de PHP, usando el mecanismo de registro del Sistema Operativo o un fichero, dependiendo de qué directiva de configuración esté establecida en error_log. Esta opción es la predeterminada.

1: `message` es enviado por email a la dirección del parámetro destination. Este es el único tipo de mensaje donde se usa el cuarto parámetro `extra_headers`.

2: Ya no es una opción (descontinuado).

3: `message` es añadido al final del fichero destination. No se añade automáticamente una nueva línea al final del string message.

4: `message` es enviado directamente al gestor de registro de la SAPI.

`destination`

El destino. Su significado depende del parámetro message_type tal como se describió arriba.

`extra_headers`

Las cabeceras extra. Se usa cuando el parámetro message_type está establecido a 1. Este tipo de mensaje usa la misma función interna que mail().

Valores devueltos

La función devuelve `true` si se ha ejecutado correctamente y `false` en caso contrario.

> ⚠ **`error_log()` no es seguro a nivel binario**
>
> `error_log()` no es seguro a nivel binario, ya que `message` será truncado por un carácter con valor `null`. Es responsabilidad del programador impedir que un usuario final pueda introducir un valor `null` en esta función.

---

1. Documentación de `set_error_handler`: https://www.php.net/manual/es/function.set-error-handler.php (consultado el 2 de febrero de 2021). ↩
2. Documentación oficial de `error_log`: https://www.php.net/manual/es/function.error-log.php (consultado el 2 de febrero de 2021). ↩

¿Qué son las excepciones?

Las excepciones son un modo de lanzar y capturar errores a través de la clase `Exception`, que puede ser redefinida por el usuario. Consulta la documentación oficial de PHP para ver la estructura original de esta clase.

Errores contra excepciones

> **Cambio de errores por excepciones**[1]
>
> PHP 7 cambia la mayoría de los errores notificados por PHP. En lugar de notificar errores a través del mecanismo de notificación de errores tradicional de PHP 5, la mayoría de los errores ahora son notificados lanzando excepciones Error.
>
> Al igual que las excepciones normales, las excepciones Error se propagarán hasta alcanzar el primer bloque catch coincidente. Si no hay bloques coincidentes, será invocado cualquier manejador de excepciones predeterminado instalado con set_exception_handler(), y si no hubiera ningún manejador de excepciones predeterminado, la excepción será convertida en un error fatal y será manejada como un error tradicional.
>
> Debido a que la jerarquía de Error no hereda de Exception, el código que emplee bloques catch (Exception $e) { … } para manejar excepciones no capturadas en PHP 5 encontrará que estos Errores no son capturados por dichos bloques. Se requiere, por tanto, un bloque catch (Error $e) { … } o un manejador set_exception_handler().

Manejo de excepciones

Al igual que ocurre en otros lenguajes, PHP permite gestionar (lanzar y atrapar) excepciones.

> **Excepciones en PHP**[2]
>
> Una excepción puede ser lanzada (“ `thrown` ”), y atrapada (“ `catched` ”) dentro de PHP. El código puede estar dentro de un bloque `try` para facilitar la captura de excepciones potenciales. Cada bloque `try` debe tener al menos un bloque `catch` o `finally` correspondiente.

Así, un ejemplo básico de control de excepciones sería:

```php
<?php

    function dividir($a, $b){
        if($b==0)
            throw new Exception('No es posible dividir por cero.');
        return $a/$b;
    }

    $a=5;
    $b=0;
    echo "<p>Voy a intentar dividir por cero.</p>";

    try {
        dividir($a, $b);
    } catch (Exception $e) {
        echo 'Se ha capturado la excepción $e, con el siguiente mensaje de
error: ', $e->getMessage();
    }

    echo "<p>Sin embargo, puedo continuar ejecutándome..."
?>
```

La línea 14 intenta dividir por cero; sin embargo, en la función `dividir` se ha establecido que, si el segundo parámetro es igual a cero, se lanzará una nueva excepción (línea 5). En otras palabras, se lanzará — `throw` — un nuevo — `new` — objeto de la clase `Exception` que recibe como parámetro el mensaje de error que se quiera establecer.

Al ejecutarse el código `throw new Exception`, la ejecución pasa directamente a la línea 15 (bloque `catch`), mostrando el mensaje que el programador ha establecido junto al mensaje establecido en la excepcion (`$e->getMessage()`). Por tanto, el resultado de esta ejecución será:

```
Voy a intentar dividir por cero.

Se ha capturado la excepción $e, con el siguiente mensaje de error: No es posible
dividir por cero.
Sin embargo, puedo continuar ejecutándome...
```

El bloque `finally`

El bloque `finally` se posiciona después del bloque `catch`, y se ejecuta **siempre** (salte o no una excepción), justo antes de continuar con el flujo habitual del programa.

Así, el siguiente código:

```
1
```

```php
<?php

    function dividir($a, $b){
        echo "<p>Intentando dividir ".$a." por ".$b."...</p>";
        if($b==0)
            throw new Exception('No es posible dividir por cero.');
        return $a/$b;
    }

    $a=5;
    $b=0;

    try {
        dividir($a, $b);
    } catch (Exception $e) {
        echo '<p>Se ha capturado la excepción $e, con el siguiente mensaje de error: '. $e->getMessage()."</p>";
    } finally{
        echo "Me ejecuto siempre, dividas o no por cero.";
    }

    echo "<p>Sin embargo, puedo continuar ejecutándome..."
?>
```

Mostrará la siguiente salida:

```
Intentando dividir 5 por 0...

Se ha capturado la excepción $e, con el siguiente mensaje de error: No es posible dividir por cero.

Me ejecuto siempre, dividas o no por cero.
Sin embargo, puedo continuar ejecutándome...
```

Y, si se modifica el parámetro `$b` por un entero distinto de cero:

```
Intentando dividir 5 por 2...

Me ejecuto siempre, dividas o no por cero.
Sin embargo, puedo continuar ejecutándome...
```

---

1. Documentación oficial de PHP. https://www.php.net/manual/es/language.errors.php7.php (consultada el 2 de febrero de 2021). ↩
2. Documentación oficial de PHP. https://www.php.net/manual/es/language.exceptions.php (consultada el 1 de febrero de 2021). ↩