Manejo de errores y parámetros del servidor © ADR Infor SL

Indice

Manejo de errores y parámetros del servidor

Al finalizar esta unidad el alumno será capaz de identificar los diferentes tipos de errores y corregirlos. Del mismo modo podrá realizar un manejo de excepciones y conocer los diferentes parámetros que nos ofrece el servidor con información sobre el usuario y la petición realizada.

Errores en PHP

Seguramente a estas alturas ya hayas tropezado con un error de PHP al intentar ejecutar un script erróneo. Estos errores nos notifican fallos en nuestros scripts o posibles mejoras y nos ayudan a la depuración de código.

La apariencia de un error en PHP es similar a la siguiente:

Warning: Division by zero in **C:\xampp\htdocs\index.php** on line 2

En esta unidad vamos a profundizar sobre la compresión de estos errores y cómo tratarlos. Todas las notificaciones de errores tienen la misma estructura:

tipo_error: descripción_error in **script** on line linea

tipo_error	Indica el tipo de error notificado. Existen diferentes tipos de errores. Algunos de ellos impiden que el script continúe su ejecución y otros son simples advertencias o sugerencias.
descripción_error	La descripción del error nos indicará cual es el problema que se nos está notificando.
script	El archivo PHP en el que se ha producido el error.
linea	La línea del archivo PHP donde se ha producido el error.

Con esta información resulta muy fácil entender dónde está el problema y cual es para solventarlo con sencillez.

Tipos de error

Vamos a ver los diferentes tipos de error que nos puede notificar PHP:

Fatal error

El tipo **Fatal error** se refiere a errores que impiden que el script continúe ejecutándose. Este es el tipo de error más severo y debe ser corregido ineludiblemente para que nuestro código funcione.

Si llamamos a una función que no existe, PHP no sabrá cómo continuar y devolverá el siguiente error:

Fatal error: Uncaught Error: Call to undefined function mifuncion() in C:\xampp\htdocs\index.php:3 Stack trace: #0 {main} thrown in **C:\xampp\htdocs\index.php** on line 3

Warning

El tipo **Warning** se refiere a un error que permite que el script continúe. Aunque la ejecución no se interrumpa, es muy probable que el script produzca resultados inesperados, por lo que es muy recomendable depurar este tipo de errores.

Si intentamos dividir un número entre cero, PHP producirá un Warning ya que, el resultado de la operación sería infinito y no podría manejarse. En este caso nos devolvería el siguiente error:

Warning: Division by zero in **C:\xampp\htdocs\index.php** on line 5

Notice

Una notificación o **Notice**, nos advierte de sucesos que pueden haber sido causados por un resultado inesperado, pero que también podrían formar parte de la ejecución normal de nuestro programa. No es necesario resolver estos errores, pero conviene tenerlos en cuenta para vigilar que nuestro código no realiza acciones inesperadas.

Si intentamos mostrar por pantalla el valor de una variable que nunca hemos utilizado, y por lo tanto, no tiene ningún valor, nos devolverá el siguiente error:

Notice: Undefined variable: a in **C:\xampp\htdocs\index.php** on line 3

Parse error

Un error de tipo **Parse error** indica que existe un fallo en la sintaxis de nuestro script. Cuando sucede esta situación, PHP no es capaz de entender el código que le estamos indicando y ni siquiera comienza la ejecución del script. Como sucede con un Fatal error, es imprescindible solventar la situación para que nuestro código funcione.

Si en un bucle for, olvidamos poner la tercera instrucción de su estructura, que se encarga de incrementar el contador, se producirá el siguiente error:

Parse error: syntax error, unexpected ')', expecting ';' in **C:\xampp\htdocs\index.php** on line 3

Deprecated

Una notificación de tipo **Deprecated** simboliza una sección de código que está utilizando una sintaxis obsoleta. Nuestro script funcionará con normalidad, pero a medida que avancen las versiones de PHP se irán eliminando las sintaxis obsoletas. Es recomendable, aunque no urgente, utilizar la sintaxis actualizada para evitar que nuestro código deje de funcionar cuando se actualice el servidor.

Si utilizamos una función que ha quedado obsoleta y que será eliminada en futuras versiones de PHP se nos notificará con una advertencia como la siguiente:

Deprecated: Function mcrypt_create_iv() is deprecated in **C:\xampp\htdocs\index.php** on line 4

Versión anterior: Strict standards

Si estás utilizando una versión de PHP anterior a la 7.0 puedes encontrar un tipo de error denominado **Strict standards**.

Este tipo de error indicaba ciertas notificaciones sobre formas de codificar que hacían un uso del lenguaje poco estricto.

A partir de PHP 7.0 se distribuyeron todos los casos en los que aparecía este tipo de error entre el resto de tipos y se eliminó la clasificación.

Tipos de error en PHP

Configurar qué errores se muestran

Resulta de vital importancia configurar adecuadamente qué errores debe mostrar el servidor y cuales debe ocultar.

Si se trata del servidor que utilizamos **para desarrollar**, suele ser conveniente mostrar **casi todos los tipos de errores** con la finalidad de depurar nuestro código e identificar todas las posibilidades de fallo.

Sin embargo, cuando nuestro código se ejecute **en un entorno real** desde nuestro servidor de producción, seguramente, no queramos mostrar **ningún tipo de notificación**. Los errores suelen dar bastante información sobre la arquitectura de nuestra aplicación que puede ser utilizada por usuarios malintencionados para atacar nuestro código.

Existen una serie de directivas de PHP que podemos especificar en el archivo **php.ini** para gestionar cómo se mostrarán los errores:

display_errors

La directiva **display_errors**, controla si PHP debe mostrar los errores por pantalla.

Puede tomar los valores **On** u **Off**.

En el caso de establecer **display_errors=Off** no se mostrará ningún error por pantalla cuando se produzcan.

Es recomendable desactivar la visualización de errores en el servidor de producción y mantenerla habilitada en el servidor de desarrollo.

error_reporting

La directiva **error_reporting** especifica qué clase de errores debe registrar PHP.

Existen una serie de valores que pueden indicarse a esta directiva para especificar los errores a registrar. Los mas comunes son:

Valor	Significado
E_ALL	Todos los errores
E_ERROR	Errores de tipo **Fatal error**
E_WARNING	Advertencias de tipo **Warning**
E_PARSE	Errores de tipo **Parse error**
E_NOTICE	Advertencias de tipo **Notice**
E_DEPRECATED	Advertencias de tipo **Deprecated**

Para indicar los errores que queremos mostrar indicaremos los valores separados por ampersands (**&**).

error_reporting = E_ERROR & E_WARNING & E_PARSE

La siguiente directiva hará que PHP registre los errores de tipo **Fatal error**, **Warning** y **Parse error**.

También podemos especificar que un error no se mostrará utilizando una virgulilla (~) antes del valor indicado.

error_reporting = E_ALL & ~E_DEPRECATED

La siguiente directiva hará que PHP registre todos los errores, excepto los de tipo **Deprecated**.

Un valor habitual para entornos de producción sería:

error_reporting = E_ALL & ~E_NOTICE & ~E_DEPRECATED

Sin embargo, en entornos de desarrollo es más frecuente utilizar un valor similar al siguiente:

error_reporting = E_ALL & ~E_NOTICE

log_errors

La directiva **log_errors**, controla si PHP debe guardar en un archivo de log, los errores que se produzcan. Esto suele ser una opción muy interesante, ya que nos aporta mucha información sobre lo sucedido en nuestra web, sobretodo en entornos de producción.

Puede tomar los valores **On** u **Off**.

En el caso de establecer **log_errors=On** se almacenarán los errores especificados en **error_reporting**, en el archivo indicado en la directiva **error_log**.

```
error_reporting = E_ERROR & E_WARNING
error_log = C:\phperrors.log
log_errors = On
```

Las siguientes directivas, almacenan en el archivo **phperrors.log** todos los **Fatal error** y **Warning** que se produzcan durante la ejecución de nuestros scripts.

Truco: Directivas de errores en tiempo de ejecución

También podemos establecer el valor de estas directivas en tiempo de ejecución, en nuestros propios scripts PHP, utilizando la función **ini_set**.

EJEMPLO:

```
ini_set('display_errors', 'On');
```

Configurar qué errores se muestran

Capturar errores

En muchas ocasiones, necesitaremos controlar cuando se produce un error para realizar una serie de acciones en tal caso. PHP nos provee estructuras para realizar un tratamiento de errores completo y poder gestionar todo tipo de situaciones que se den en nuestro código.

try

Para capturar un error de tipo **Fatal error** y tratarlo, tenemos a nuestra disposición la estructura **try** con la siguiente sintaxis:

```
try {
    // Bloque de código a testear
} catch (Throwable $t) {
    // Acciones a realizar si se produce un error
}
```

Si se produce un error en alguna de las instrucciones del bloque inicial **try**, la ejecución del script saltará inmediatamente al bloque **catch** y ejecutará las instrucciones contenidas en él.

Esta estructura solo captura los errores de tipo Fatal error. Los errores de tipo Warning, Notice o Deprecated seguirán siendo mostrados aunque se produzcan dentro de la estructura **try**.

Podemos obtener información del error producido utilizando la variable **$t** que hemos definido en el apartado **catch**. Las siguientes instrucciones nos indican estos datos:

Instrucción	Valor obtenido
$t - > getMessage()	Obtiene la descripción del error capturado.
$t -> getFile()	Obtiene la ruta del script en el que se ha producido el error capturado.
$t -> getLine()	Obtiene la línea del script en el que se ha producido el error capturado.

En el siguiente script no tenemos definida la función "mifuncion"

```
<?php
    try {
        mifuncion();
    catch (Throwable $t) {
        echo "No existe la función\n";
    }
    echo 'Mi script continúa';
?>
```

En lugar de producirse un error y detener el script, producirá la siguiente salida:

```
No existe la función
Mi script continúa
```

Versión anterior: Estructura try en PHP 5 o anterior.

El manejo de errores en PHP 7 ha variado con respecto a versiones anteriores. Si estás utilizando PHP 5 o anterior, la estructura try tendrá la siguiente sintaxis:

```
try {
    // Bloque de código a testear
} catch (Exception $e) {
    // Acciones a realizar si se produce un error
}
```

set_error_handler

Como hemos visto, la estructura **try** no permite capturar los errores que no finalizan la ejecución del script (**Warning**, **Notice** o **Deprecated**).

Para capturar y tratar esta clase de errores, tendremos que utilizar un **manejador de errores** personalizado.

Para hacer esto, debemos crear una función que se encargará de realizar las acciones correspondientes cuando se produzcan estos errores. Su sintaxis será la siguiente:

```
function manejador_errores($codigo,$mensaje,$archivo,$linea) {
    // Acciones a realizar cuando se produce un error Warning, Notice o Deprecated
}
```

Una vez creada la función debemos indicar de que se encargará de tratar los errores con la siguiente instrucción:

```
set_error_handler('manejador_errores');
```

Esta instrucción establece la función que indiquemos como manejador de errores. Cuando se produce un error, le envía los parametros $codigo, $mensaje, $archivo y $línea que representan el **código del error**, la **descripción**, la **ruta del script** y la **línea** donde se ha producido el error respectivamente.

Todos los errores no fatales producidos después de la asignación del manejador de errores, no se mostrarán por pantalla. Si queremos deshabilitar nuestro manejador de errores y dejar que PHP siga tratando los errores no fatales como habitualmente, podemos hacerlo con la siguiente instrucción:

```
restore_error_handler();
```

Tenemos el siguiente script de PHP

```
<?php
    function manejador_errores($codigo,$mensaje,$archivo,$linea) {
        global $mensaje;
        $mensaje = 'El resultado es infinito';
    }
    set_error_handler('manejador_errores');
    $resultado = 48758 / ((int)$_GET['divisor']);
    restore_error_handler();
    if ($mensaje == '') {
        echo "El resultado es $resultado";
    }
    echo $mensaje;
?>
```

Si el usuario nos envía 0 en el parámetro "divisor" se producirá un error y el manejador de errores lo capturará produciendo la siguiente salida:

```
El resultado es infinito
```

Capturar errores

Parámetros del servidor

En muchas ocasiones, necesitamos conocer cierta **información relativa al usuario** que está solicitando una página en nuestro servidor. Esta información, puede resultar interesante para identificar al usuario y su entorno.

Algunos datos que podemos obtener desde PHP del entorno del usuario son:

1
Dirección IP del usuario.
2
Navegador que utiliza.
3
Idioma predeterminado de su navegador.

También nos puede ser muy interesante conocer **información de nuestro propio servidor**. Es muy probable que nuestro código tenga que ser migrado en algún momento de un servidor a otro. Conocer el entorno en el que nuestro código se ejecuta y adaptarse a él nos evitará muchos quebraderos de cabeza en el futuro.

Algunos datos que podemos obtener desde PHP del entorno del servidor son:

1
Nombre del servidor.
2
Software utilizado para servir las páginas.
3
Puerto por el que el servidor está sirviendo.

El array $_SERVER

Toda la información anteriormente citada se encuentra en un array predefinido por PHP. Su sintaxis es la siguiente:

$_SERVER[parámetro]

parámetro es el nombre del parámetro del servidor que queremos obtener.

A continuación tienes un ejemplo de los principales parámetros que puedes encontrar en $_SERVER:

Parámetro	Descripción	Ejemplo de valor devuelto
HTTP_HOST	Nombre del servidor	www.example.com
HTTP_USER_AGENT	Cadena de texto que identifica al navegador del usuario.	Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36
HTTP_REFERER	Página desde la cual se ha llamado a la URL actual al pulsar un enlace.	http://www.example.com/item.php
HTTP_ACCEPT_LANGUAGE	Idiomas configurados en el navegador del usuario.	es-ES,es;q=0.8
SERVER_SOFTWARE	Software utilizado para servir las páginas.	Apache/2.4.25 (Win32) OpenSSL/1.0.2j PHP/7.1.1
SERVER_NAME	Nombre del servidor.	www.example.com
SERVER_ADDR	Dirección IP del servidor.	74.125.128.94
SERVER_PORT	Puerto por el cual se sirven las páginas desde el servidor.	80
REMOTE_ADDR	Dirección IP del usuario.	79.148.106.54
DOCUMENT_ROOT	Dirección absoluta de la carpeta pública del servidor.	C:/xampp/htdocs
SCRIPT_FILENAME	Dirección absoluta del script que se está ejecutando.	C:/xampp/htdocs/index.php
SERVER_PROTOCOL	Versión del protocolo HTTP	HTTP/1.1
REQUEST_URI	URL relativa a la que se está llamando.	/
PHP_SELF	Dirección relativa del script que se está ejecutando.	/index.php
REQUEST_TIME	Timestamp de la solicitud.	1496843010

Vamos a ver un ejemplo de su uso:

Tenemos el siguiente script de PHP

```
Tu dirección IP es <?php echo $_SERVER['REMOTE_ADDR']; ?>
```

Al llamarlo desde el navegador mostrará lo siguiente:

Tu dirección IP es 79.148.106.54

El array $_SERVER

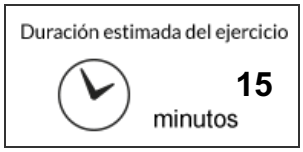
Hemos aprendido

En esta unidad hemos aprendido:

- Cuando se produce un error, PHP nos indicará el tipo de error que es, una descripción del mismo, el archivo en el que se produce y la línea del error.
- Existen varios tipos de error con diferentes significados: **Fatal error**, **Parse error**, **Warning**, **Notice** y **Deprecated**.
- Podemos configurar qué errores mostrar y cómo con las directivas de php.ini: **display_errors**, **error_reporting**, **log_errors** y **error_log**.
- Con la estructura **try** capturaremos los errores fatales.
- Con la instrucción **set_error_handler** construiremos un manejador de errores para capturar los errores no fatales.
- Podemos obtener información del usuario y del servidor con el array predefinido de PHP **$_SERVER**.

Ejercicios

Ejercicio 1: Recuperar el contenido de una URL



Para finalizar correctamente este ejercicio deberás leer el contenido de una URL y mostrarlo por pantalla.

Será necesario realizar un control de errores para mostrar un aviso en el caso de que la URL sea incorrecta o no se pueda recuperar su contenido.

Lo necesario para comenzar

Descarga esta página y sírvela a través de tu servidor web. Deberás descomprimir los archivos que contiene el fichero ZIP en el directorio público de tu servidor. Te servirá como base para desarrollar el ejercicio correctamente.

ejercicio1.zip

Plantilla web.

Pasos a seguir

1. Recupera con la función **file_get_contents** el contenido de la URL enviada en el parámetro.
2. Muestra el contenido recuperado por pantalla.
3. Realiza un **manejador de errores** para controlar el **Warning** que se produce si la función no puede recuperar el contenido.
4. En el caso de que la función file_get_contents falle, mostraremos una notificación por pantalla indicando que no se puede recuperar el contenido del archivo.

Recuerda: allow_url_fopen

Para que la función **file_get_contents** funcione con **URLs externas** a nuestro servidor, necesitamos que la directiva de php.ini **allow_url_fopen** esté establecida a **On.**

Solución

Solución al ejercicio

Ejercicio 2: Utilizar el idioma del navegador

Duración estimada del ejercicio

25 minutos

Para finalizar correctamente este ejercicio, deberás mostrar un saludo por pantalla en el idioma del usuario.

Lo necesario para comenzar

Descarga esta página y sírvela a través de tu servidor web. Deberás descomprimir los archivos que contiene el fichero ZIP en el directorio público de tu servidor. Te servirá como base para desarrollar el ejercicio correctamente.

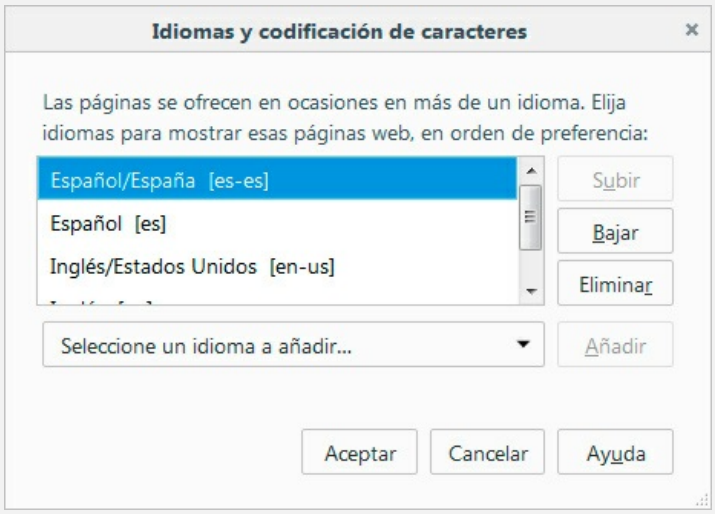
ejercicio2.zip

Plantilla web.

Pasos a seguir

1. Obtén los idiomas que están **configurados en el navegador** del usuario.
2. Vamos a saludar al usuario en **español**, **inglés** o **francés**. Guardaremos como idioma por defecto el que **más prioridad** tenga según el navegador del usuario, dentro de estos tres. Si no tuviese ninguno de los tres idiomas, el idioma por defecto sería el inglés.
3. Muestra al usuario **Hola**, **Hello** o **Bonjour** en función del idioma que hemos establecido.

Puedes cambiar los idiomas por defecto desde la configuración de tu navegador.



Solución

Solución al ejercicio

Recursos

Preguntas Frecuentes

1. **¿Existe una estructura para capturar los errores de tipoParse error?**

   No. Un **Parse error** se produce por un fallo de sintaxis de PHP que impide que el script empiece a procesarse. Debido a que ninguna instrucción de las que indiquemos será entendida por PHP, no tiene sentido tener una estructura para capturar estos errores ya que nunca podría ser entendida.

   Los errores de tipo Parse error no dependen de la ejecución del script ni de las circunstancias que le rodean. Lo único que debemos hacer es solventar nuestro fallo de sintaxis y nunca se volverá a producir bajo ninguna circunstancia.

2. **Las componentes del array $_SERVER ¿son siempre las mismas en todos los servidores?**

   No. Las componentes de **$_SERVER** pueden variar de un servidor a otro. Depende de la configuración, de las versiones o del entorno en el que se ejecute el servidor tendremos algunas componentes adicionales opcionales.

   No obstante las componentes principales son comunes en todos los casos y podemos utilizarlas sin miedo a que no estén disponibles.

3. **¿Puedo establecer un manejador de errores sólo para un tipo de error? Por ejemplo, sólo para Warnings.**

   Si, es posible especificar, en un parámetro adicional de **set_error_handler**, qué errores manejará nuestra función:

   set_error_handler('mifunción',E_WARNING); // Captura los errores de tipo Warning

   set_error_handler('mifunción',E_NOTICE); // Captura los errores de tipo Notice

   set_error_handler('mifunción',E_DEPRECATED); // Captura los errores de tipo Deprecated

Glosario.

- **Log**: Un log es un historial de sucesos. Por lo general suele tratarse de un archivo de texto plano al cual se añaden líneas cuando suceden ciertos eventos.

- **Manejador de errores**: Estructura de código encargada de tomar las decisiones y realizar las acciones oportunas cuando se produce un error en nuestro código.

- **Servidor de desarrollo**: Es el servidor utilizado para desarrollar nuestra web y realizar las pruebas necesarias. Suele tener pocos recursos ya que sólo es utilizado por nosotros y nuestro equipo de trabajo y gran cantidad de notificaciones para realizar el seguimiento y depurar los errores que surjan durante el desarrollo.

- **Servidor de producción**: Es el servidor utilizado para servir nuestra web de forma definitiva, cuando el desarrollo está finalizado y la página publicada. Por lo general tiene medidas de seguridad más elevadas que un servidor de desarrollo y una mayor cantidad de recursos para soportar la carga de trabajo.