PHP

La Manera Correcta.

Bienvenido

Hoy en día existe mucha información anticuada acerca de PHP que guía a nuevos programadores por mal camino, propaga las malas prácticas y código inseguro. *PHP: La Manera Correcta* es una referencia práctica y fácil de entender, los mejores métodos, estándares de código, enlaces a tutoriales autoritativos alrededor de la Web y lo que los contribuyentes consideran como las mejores prácticas en la actualidad.

No existe una manera canónica de utilizar PHP. Este sitio tiene como objetivo introducir a los nuevos desarrolladores en PHP a algunos temas que tal vez no descubran hasta que es demasiado tarde, y también ofrecer nuevas ideas a los profesionales experimentados sobre aquellos temas que han estado haciendo durante años sin reconsiderar. Este sitio no le dirá que herramientas utilizar, en su lugar le ofrecerá diferentes opciones, cuando sea posible se le explicarán las diferencias de enfoque y casos de uso.

Consideramos este sitio como un documento vivo que se continuará actualizando con más información útil y ejemplos según se hagan disponibles.

Traducciones

PHP: La Manera Correcta ha sido (o muy pronto será) traducido a diferentes lenguajes:

- Alemán
- Búlgaro
- Chino (simplificado)
- Coreano
- Esloveno
- Español
- Francés
- Indonesio
- Inglés
- Italiano
- Japones

- Polaco
- Portugués
- Rumano
- Ruso
- Thai
- Turco
- Ucraniano

¿Cómo Colaborar?

¡Ayuda a que este sitio se convierta en el mejor recurso para nuevos programadores en PHP! Colabora en GitHub

¡Corre la voz!

PHP: La Manera Correcta tiene disponibles *banners* que puedes usar en tu sitio web. ¡Apoya la causa y hazles saber a nuevos desarrolladores en PHP donde es que pueden encontrar buena información!

Ver las imágenes para *banners*

Volver al Inicio

Primeros pasos

Use la versión estable actual (5.6)

Si está dando sus primeros pasos con PHP, asegúrese de usar con la última versión estable (current stable) de PHP 5.6. En los últimos años se ha progresado mucho añadiendo nuevas y potentes características a PHP. Aunque la diferencia entre las versiones 5.2 y 5.6 aparenta ser mínima, en realidad la nueva versión representa *grandes* mejoras en el sistema. Si estás buscando alguna función en particular o su modo de uso, la documentación oficial en php.net tendrá la respuesta.

Servidor Web Integrado

Usted puede comenzar a aprender PHP sin tener que instalar y configurar un servidor web completo (se requiere la versión PHP 5.4). Para iniciar el servidor integrado se necesita correr el siguiente comando en la terminal desde el directorio raíz de su proyecto:

```
> php -S localhost:8000
```

- Aprenda más acerca del servidor web integrado

Configuración en Mac

El sistema OSX viene previamente configurado con un PHP que normalmente no está actualizado a la versión corriente. El sistema “Lion” viene configurado con PHP 5.3.6, el sistema “Mountain Lion” con la versión 5.3.10 y el sistema “Mavericks” con al versión 5.4.17.

Para actualizar la versión de PHP en el sistema OSX se puede instalar por medio de diferentes gestores de paquetes, sin embargo, recomendamos el paquete php-osx by Liip.

La otra opción disponible es que usted compile el paquete de instalación. En tal caso debe asegurarse de tener la aplicación para desarrollo *Xcode* o, como sustituto, las herramientas “Command Line Tools for Xcode” que se pueden descargar directamente del *Mac Developer Center* de Apple.

También existen paquetes tipo “todo incluido” con un control gráfico de configuración bastante simple, MAMP o XAMPP. Estos incluyen una configuración de PHP junto con el servidor web Apache y el gestor de base de datos MySQL.

Configuración en Windows

Hay un sinnúmero de maneras de configurar PHP en el sistema Windows. Usted puede descargar los binarios y recientemente el paquete de instalación estaba disponible en el formato ‘.msi’. Este paquete de instalación ya no se actualiza y la última versión fue PHP 5.3.0.

Para aprender PHP o el desarrollo local, se puede utilizar el servidor web embebido que viene con la versión PHP 5.4, así no tendrá que preocuparse por configurar un servidor por separado. Ahora bien, si le gustaría disponer de una solución tipo “todo incluido”, que le ofrece un servidor web completo junto con el gestor de base de datos MySQL, entonces herramientas como el Web Platform Installer, XAMPP y WAMP le ayudaran a configurar un entorno de desarrollo web en Windows más fácilmente y en menos tiempo. Tenga en consideración que estas herramientas son un poco diferentes a las que usara en su sistema de producción, así que tenga cuidado de las diferencias en el entorno de su aplicación si es que la desarrolla en Windows pero la despliega en Linux.

Si necesita correr un sistema de producción en Windows entonces el servidor IIS 7 le ofrecerá un entorno más estable y con el mejor rendimiento. Una herramienta como phpmanager (un complemento GUI para IIS 7) le simplificara la gestión y configuración de PHP en este servidor. IIS 7 viene configurado con FastCGI listo para su uso, solo necesita apuntar a PHP como controlador. Para más información y recursos adicionales refiérase a la área dedicada en iis.net para PHP.

Volver al Inicio

Guía de Estilo de Código

La comunidad detrás de PHP es enorme y diversa, compuesta de innumerables librerías, armazones de desarrollo (*frameworks*) y componentes. Es común que desarrolladores en PHP escojan de entre estos y los combinen en un solo proyecto. Por eso es importante que el código PHP se adhiera (lo más cerca posible) a un estilo de código común para que se facilite el trabajo de los desarrolladores al combinar una variedad de librerías para sus proyectos.

El Framework Interop Group (antes conocido como el 'PHP Standards Group') ha propuesto y aprobado una serie de recomendaciones de estilo, conocidas como PSR-0, PSR-1, PSR-2 y PSR-4. No deje que los títulos raros lo confundan, estas recomendaciones son simplemente un conjunto de normas que algunos proyectos como Drupal, Zend, CakePHP, phpBB, AWS SDK, FuelPHP, Lithium, y otros han comenzado a adoptar. Usted puede utilizar estas normas en sus propios proyectos o continuar usando su propio estilo.

Sería ideal que escribiera código PHP que se adhiere a uno o más de estos estándares. Puede ser cualquier combinación de PSR, o uno de los estándares de codificación hechos por PEAR o Zend. Esto significa que otros desarrolladores pueden fácilmente leer y trabajar con su código, y las aplicaciones que implementan los componentes puedan tener consistencia incluso cuando trabajen con una gran cantidad de código de terceros.

- Lea acerca de PSR-0
- Lea acerca de PSR-1
- Lea acerca de PSR-2
- Lea acerca de PSR-4
- Lea acerca de los Estandares de Codificación de PEAR
- Lea acerca de los Estandares de Codificación de Zend
- Lea acerca de los Estandares de Codificación de Symfony

Puede utilizar PHP_CodeSniffer para verificar que su código se apegue a estas recomendaciones, y plugins para editores de texto como Sublime Text 2 para obtener un análisis en tiempo real.

Utilice PHP Coding Standards Fixer de Fabien Potencier para modificar la sintaxis de su código automáticamente y así quede conforme a estos estándares, ahorrándole tiempo y esfuerzo que requeriría arreglar cada problema a mano.

Se prefiere la utilización del idioma Inglés para todos los nombres de símbolos e infraestructura del código. Los comentarios pueden ser escritos e un lenguaje fácilmente legible para todos autores y futuros desarrolladores que trabajaran en la base del código.

Volver al Inicio

Aspectos Destacados del Lenguaje

Paradigmas de Programación

PHP es un lenguaje flexible y dinámico que permite usar una variedad de técnicas de programación. El lenguaje ha evolucionado dramáticamente a través de los años. Se añadió un modelo de objetos (object-oriented) sólido en la versión

5.0 (2004), funciones anónimas y espacios de nombres (namespaces) en PHP 5.3, y rasgos (traits) en PHP 5.4 (2012).

Programación Orientada a Objetos

PHP tiene un conjunto muy completo de aspectos que facilitan la programación orientada a objetos (OOP) que incluye la habilidad de crear clases, clases abstractas, interfaces, herencia, constructores, clonación de objetos, excepciones y mucho más.

- Leer más acerca de PHP orientado a objetos
- Leer más acerca de Rasgos

Programación Funcional

PHP tiene la capacidad de declarar funciones de primera clase, en otras palabras, una función puede ser asignada a un variable. Las funciones definidas por el usuario, así como las funciones internas (incluidas), tiene la habilidad de ser referenciadas por un variable e invocadas dinámicamente. Las funciones pueden ser pasadas como argumentos a otras funciones (un aspecto llamado funciones de orden superior) y funciones pueden devolver otras funciones.

La recursividad es un aspecto que le permite a una función a llamarse a sí misma. El lenguaje PHP habilita este tipo de algoritmos, sin embargo, la mayoría del código PHP se enfoca en iteración.

Las funciones anónimas (con soporte para closures) están presentes en PHP desde la versión 5.3 (2009).

En PHP 5.4 se añadió la habilidad para vincular closure al ámbito de un objeto y también se mejoró el soporte de funciones de tipo *callable* para que puedan intercambiarse con funciones anónimas en casi todos los casos.

- Continúe leyendo acerca de la programación funcional en PHP
- Leer acerca de las funciones anónimas
- Leer acerca de la clase Closure
- Mas detalles definidos en el RFC de closures
- Leer acerca de funciones tipo Callable
- Leer acerca de cómo invocar funciones con `call_user_func_array`

Programación Meta

PHP soporta varias formas de programación meta por medio de mecanismos como el API de Reflexión y los Métodos Mágicos. Hay muchos Métodos Mágicos disponibles como `__get()`, `__set()`, `__clone()`, `__toString()`, `__invoke()` y más, que permiten a los desarrolladores a conectarse con el funcionamiento de la clase. A menudo desarrolladores en Ruby dicen que a PHP le falta la función de `method_missing`, sin embargo los aspectos de esta función están disponibles en `__call()` y `__callStatic()`.

- Leer acerca de los Métodos Mágicos
- Leer acerca de Reflexion

Namespaces

Como mencionamos anteriormente, la comunidad de PHP tiene muchos desarrolladores creando una gran cantidad de código fuente. Esto quiere decir que existe la posibilidad que dos librerías diferentes utilicen el mismo nombre para una clase en su código. Cuando las dos librerías se usan dentro del mismo namespace esto se denomina como una colisión y puede causar problemas.

Los *Namespaces* resuelven este problema. Como se describe en el manual de referencia de PHP, los namespaces son similares a los directorios que separan los archivos en el sistema operativo. Dos archivos con el mismo nombre pueden coexistir en directorios separados. Igualmente, dos clases de PHP con el mismo nombre pueden coexistir en namespaces separados, es tan simple como eso.

Es importante que separe su código con un namespace para que pueda ser usado por otros desarrolladores sin la preocupación de que cause conflictos con otras librerías.

Uno de los métodos recomendados para el uso de espacios de nombres se indica en el PSR-0, el cual se propone proveer una convención estándar para los archivos, clases y los namespaces, lo cual facilita el intercambio y uso del código en diferentes proyectos.

- Leer acerca de los Namespaces
- Leer acerca del PSR-0

Librería Estándar de PHP

La Librería Estándar de PHP (SPL) viene empaquetada con PHP y provee una colección de clases e interfaces compuesta principalmente de clases de estructura de datos (como stack, queue y heap) e iteradores que pueden atravesar estas estructuras de datos o sus propias clases que implementan las interfaces de la SPL.

- Leer acerca de la SPL

Interface de Línea de Comando (CLI)

PHP fue creado principalmente para desarrollar aplicaciones web, pero también es muy útil para implementar programas que corren en la interface de línea de comando (CLI). Los programas de línea de comando en PHP pueden ayudarle a automatizar tareas comunes como pruebas, despliegues y la administración de aplicaciones.

Los programas CLI en PHP son muy potentes porque el código de la aplicación se puede utilizar directamente sin tener que crear o asegurar un GUI web para su uso. Por esta razón, ¡asegúrese de no colocar sus programas CLI en su directorio raíz público!

Intente correr PHP desde la línea de comando:

```
> php -i
```

La opción `-i` imprimirá la configuración de PHP, como sucede con la función `phpinfo`. La opción `-a` habilita una consola interactiva muy similar al IRB de Ruby o a la consola interactiva de Python. Existen varias opciones de línea de comando que resultan muy útiles. Vamos a escribir un programa simple que imprima "Hola, $nombre" a la línea de comando. Para empezar, vamos a crear un archive llamad `hola.php` como se muestra enseguida:

```
<?php
if($argc != 2) {
    echo "Uso: php hola.php [nombre].\n";
    exit(1);
}
$nombre = $argv[1];
echo "Hola, $nombre\n";
```

PHP hace disponibles dos variables especiales basados en los argumentos que recibe el programa el ser ejecutado. El variable de tipo *entero* `$argc` contiene el *count* o número de argumentos y el variable de tipo *array* `$argv` contiene el *value* o valor de cada uno de los argumentos que se pasaron durante la ejecución. El primer argumento siempre es el nombre del archivo del programa PHP, que en este caso es `hola.php` .

La expresión `exit()` se puede usar con un número que no es cero para dejarle saber a la consola que el comando ha fallado. Aquí puede encontrar los códigos de salida más comúnmente usados.

Para ejecutar el programa desde la línea de comando:

```
> php hola.php
Uso: php hola.php [nombre]
> php hola.php mundo
Hola, mundo
```

- Aprenda acerca de la ejecución de PHP desde la línea de comando
- Aprenda como configurar Windows para ejecutar PHP desde la línea de comando

XDebug

Es una de las herramientas más útiles en el desarrollo de software, es un depurador o *debugger*. Permite el trazado de ejecución de tu código y monitorear el contenida de la pila de ejecución. XDebug, depurador para PHP, puede ser utilizado por varios IDEs para proveer *Breakpoints* e inspeccionar la pila de ejecución. También permite que herramientas como PHPUnit y KCacheGrind realicen análisis de cobertura del código.

Si te encuentras en un aprieto, y recurres a *var_dump/print_r*, y sigues sin encontrar la solución - tal ves necesitas usar un depurador.

Instalar XDebug puede ser complicado, pero una de las características más importantes es la “Depuración Remota” - Si ésta desarrollando su código localmente y después lo prueba dentro de una VM (Máquina Virtual) u en otro servidor, la Depuración Remota es la característica que necesitaras habilitar desde un comienzo.

Tradicionalmente, tendrá que modificar su VHost Apache o el archivo .htaccess con los siguientes valores:

```
php_value xdebug.remote_host=192.168.?.?
php_value xdebug.remote_port=9000
```

El “remote host” y el “remote port” corresponderán a su computadora local y el puerto de escucha que usted configuró para su IDE. Ahora solo es cuestión de que ponga su IDE dentro del modo “escuchar conexiones”, y cargue la URL:

```
http://your-website.example.com/index.php?XDEBUG_SESSION_START=1
```

Su IDE ahora interceptará el estado actual de la ejecución de su script, permitiéndole establecer *breakpoints* y verificar los valores en memoria.

Los depuradores gráficos hacen muy fácil el proceso de recorrer el código, inspeccionar variables, y evaluar el código en tiempo de ejecución. Varios IDE’s tienen integrado o pueden ser integrados vía plugin la depuración gráfica con XDebug. MacGDBp es un GUI de XDebug para Mac y también es gratuito, open-source y stand-alone.

- Leer más acerca XDebug
- Leer más acerca de MacGDBp

Volver al Inicio

Gestión de Dependencias

Existen un sinnúmero de librerías, entorno de trabajo (*frameworks*) y componentes de PHP de donde escoger y lo más probable es que tu proyecto utilice varios de ellos. A esto se les llama dependencias de proyecto. Hasta muy recientemente, PHP no tenía una manera fácil de gestionar tales dependencias. Aun si te hacías cargo de estas manualmente, todavía tendrías que mantenerte al tanto de los cargadores automáticos (autoloaders). Ahora ya no es así.

Actualmente hay dos sistemas principales de gestión de paquetes para PHP: Composer y PEAR. ¿Cuál es el adecuado para ti? La respuesta es, ambos.

- Utiliza **Composer** cuando manejes las dependencias de un solo proyecto.

- Utiliza **PEAR** cuando manejes las dependencias de todo el sistema de PHP en su plataforma.

Generalmente, los paquetes de Composer solo estarán disponibles en los proyectos que especifiques explícitamente, mientras que un paquete de PEAR estaría disponible a todos los proyectos bajo el sistema PHP. Puede que PEAR parezca la propuesta más accesible a primera vista, sin embargo, también existen ventajas al manejar las dependencias con Composer en base al proyecto en que se está trabajando.

Composer y Packagist

Composer es un **excelente** gestor de dependencias para PHP. Solo tienes que añadir las dependencias de tu proyecto a un archivo llamado `composer.json`, ejecutar algunos comandos y Composer descargará automáticamente las dependencias para el proyecto y configurará el cargador automático en cuestión de segundos.

Ya existen muchas librerías PHP compatibles con Composer y están listas para que las uses en tus proyectos. Hay una lista de estos "paquetes" en el sitio Packagist, que es el repositorio oficial de librerías compatibles con Composer.

Como configurar Composer

Se puede instalar Composer localmente (en el directorio de trabajo actual, aunque esto ya no es recomendable) o globalmente (por ejemplo, en /user/local/bin). Supongamos que quieres configurar Composer localmente. Desde el directorio raíz de tu proyecto:

```
curl -s http://getcomposer.org/installer | php
```

Esto descargará el archivo binario `composer.phar`. Se puede ejecutar este archivo con `php` para manejar las dependencias de tu proyecto. *Por favor, ten en cuenta que:* Si ejecutas el código directamente en el intérprete de PHP al mismo tiempo que lo descargas, deberías leer cuidadosamente el código descargado del programa con anterioridad y confirmar que es seguro.

Como configurar Composer (manualmente)

Configurar Composer manualmente requiere técnicas avanzadas. No obstante, hay varias razones por las cuales un desarrollador como tú prefiera configurarlo de esta manera en vez de utilizar la rutina de configuración interactiva. La configuración interactiva verifica tu sistema de PHP para asegurar que:

- Está utilizando una versión compatible de PHP
- Los archivos `.phar` pueden ser ejecutados correctamente
- Tiene suficientes permisos en los directorios
- Algunas extensiones que pueden causar problemas no han sido cargadas
- El archivo `php.ini` tiene los ajustes necesarios

Ya que la configuración manual no realiza ninguna de estas verificaciones, necesitas tomar en consideración si tal funcionamiento es lo que deseas. De todas maneras, puedes obtener y configurar Composer manualmente de la siguiente manera:

```
curl -s http://getcomposer.org/composer.phar -o $HOME/local/bin/composer
chmod +x $HOME/local/bin/composer
```

La ruta `$HOME/local/bin` (o un directorio que escojas) necesita estar en el variable de entorno `$PATH`. De esta manera, el comando `composer` estará habilitado en cualquier directorio del sistema.

Cuando encuentres instrucciones que sugieran ejecutar Composer con `php composer.phar install`, esto se puede sustituir con:

```
composer install
```

Como Definir y Configurar Dependencias

Primero, se crea un archivo `composer.json` en el mismo directorio donde se encuentra `composer.phar`. Enseguida, encontramos un ejemplo que define el paquete Twig como una dependencia del proyecto:

```
{
    "require": {
        "twig/twig": "1.8.*"
    }
}
```

El siguiente paso es ejecutar Composer desde el directorio raíz de tu proyecto:

```
php composer.phar install
```

Esto descargará y configurará la dependencia dentro de un directorio llamado `vendors/`. Una vez configurado, añade esta línea de código al archivo principal de tu aplicación:

```
<?php
require 'vendor/autoload.php';
```

Esta instrucción le dice al programa que use el cargador automático de Composer para cargar cualquier dependencia que hayas configurado. Ahora tus dependencias serán cargadas dinámicamente según tu programa las requiera.

- Aprenda más acerca de Composer

PEAR

Otra alternativa para la gestión de paquetes que muchos desarrolladores de PHP prefieren es PEAR. Su funcionamiento es muy similar al de Composer y vale la pena investigar más sus aspectos sobresalientes.

Aprenda acerca de PEAR.

Volver al Inicio

Buenas Prácticas

Fundamentos

PHP es un gran lenguaje que permite a cualquier desarrollador producir código de forma rápida y de manera eficiente. Sin embargo, mientras que avanzamos con el lenguaje, a menudo olvidamos (o pasamos por alto) las prácticas básicas que aprendimos al inicio, tomando así atajos o malas prácticas. Para ayudar a combatir este problema común, esta sección tiene como objetivo recordar a los desarrolladores las buenas prácticas dentro de PHP.

- Continua leyendo acerca de Los Fundamentos

Fecha y Hora

PHP dispone de una clase llamada **DateTIme** para asistir en la lectura, escritura, comparación y calculación de la fecha y hora. Existen muchas funciones en PHP relacionadas con la manipulación de la fecha y la hora, sin embargo, la clase **DateTime** provee una mejor interface orientada a objetos para los situaciones más comunes. También tiene la capacidad de utilizar los husos horarios; esta información no se considerará en esta corta introducción.

Para comenzar a trabajar con DateTime primero se convierte la cadena de texto que contiene la fecha y la hora a un objeto con el método de fabricación `createFromFormat()` o también se puede crear un objeto con `new \DateTime` que contendrá la fecha y hora actual. Después, puede utilizar el método `format()` para convertir el objeto DateTime de regreso a una cadena de texto e imprimirla:

```
<?php
$fecha = '22. 11. 1968';
$inicio = \DateTime::createFromFormat('d. m. Y', $fecha);

echo "Fecha de Inicio: " . $inicio->format('m/d/Y') . "\n";
```

Se puede utilizar la clase DateInterval para realizar calculaciones con DateTime. La clase DateTime tiene métodos como `add()` y `sub()` que requieren que pase un DateInterval como argumento. Nunca escriba código que cuente con el mismo número de segundos en todos los días ya que los ajustes de los husos horarios y el horario de verano (daylight savings time) invalidaría esa suposición. En vez de eso, utilice los intervalos de fechas para hacer sus calculaciones. Para calcular la diferencia entre fechas utilice el método `diff()`. Este le devolverá un `return new DateInterval`, lo cual puede ser fácilmente impreso en la pantalla.

```
<?php
// Crea una copia de $inicio y añade un mes y 6 días
$final = clone $inicio;
$final->add(new \DateInterval('P1M6D'));

$diff = $final->diff($inicio);
echo "Diferencia: " . $diff->format('%m mes, %d días (total: %a días)') . "\n";
// Diferencia: 1 mes, 6 días (total: 37 días)
```

Es posible comparar fácilmente los objetos DateTime:

```
<?php
if($inicio < $final) {
    echo "¡El inicio sucede antes que el final!\n";
}
```

Este último ejemplo demuestra cómo se utiliza la clase DatePeriod para iterar sobre eventos periódicos. El objeto toma dos objetos DateTime, uno para el inicio y el otro para el final, y el intervalo que define el número de eventos periódicos que se devuelven.

```
<?php
// Imprimir todos los jueves entre $inicio y $final
$intervaloDePeriodo = \DateInterval::createFromDateString('first thursday');
$iteradorDePeriodo = new \DatePeriod($inicio, $intervaloDePeriodo, $final, \DatePeriod::EXCLUDE_START_DATE);
foreach($iteradorDePeriodo as $fecha)
{
    // Imprimir cada fecha en el periodo
    echo $fecha->format('m/d/Y') . " ";
}
```

- Leer acerca de la clase DateTime
- Como formatear la fecha (Opciones de los formatos aceptados para una fecha)

Patrones de Diseño

Cuando se desarrolla una aplicación es muy bueno utilizar patrones de uso común en su código y patrones para la estructura general de su proyecto. El usar estos patrones ayuda y facilita el manejo de su código y permite a otros desarrolladores a entender fácilmente como es que encajan todas las partes de su aplicación.

Si utiliza un armazón (*framework*) para el desarrollo de su proyecto, entonces la mayor parte del nivel superior de su código estará basado en este armazón, así que muchas de las decisiones ya han sido hechas por usted. Pero todavía depende de usted el escoger los mejores patrones a seguir en el código que desarrolla sobre este armazón. Si, por otro lado, no está utilizando un armazón para desarrollar su aplicación, entonces necesita encontrar los patrones que mejor se adapten al tipo y tamaño de su proyecto.

- Continúe leyendo acerca de los Patrones de Diseño

Trabajando con UTF-8

Esta sección fue escrita originalmente por Alex Cabal sobre PHP Best Practices y se ha utilizado como base para nuestros propias recomendaciones acerca de UTF-8 advice.

No hay una sola manera. Sea cuidadoso, detallista y consistente.

Ahora mismo, PHP no soporta Unicode a bajo nivel. Hay formas de asegurarse de que las cadenas UTF-8 sean procesadas correctamente, pero no es sencillo, además requiere que excarvemos en todos los niveles de nuestra aplicación web, desde HTML, pasando por SQL hasta PHP. Nos esforzaremos por hacer un breve resumen práctico.

UTF-8 al nivel de PHP

Las operaciones básicas con cadenas, como la concatenación de dos cadenas o la asignación de cadenas a variables, sno necesitan nada especial de UTF-8. Sin embargo, la mayoría de las funciones para el manejo de cadenas como `strpos()` o `strlen()`, necesitan consideración especial. A menudo, estas funciones tienen su contrapartida con funciones `mb_*` como por ejemplo: `mb_strpos()` o `mb_strlen()`. Estas cadenas `mb_*` están disponibles para Usted vía la extensión para el manejo de Cadenas de Caracteres Cadenas Multibyte, y están diseñadas específicamente para operar con cadenas Unicode.

Siempre debe usar las funciones `mb_*` para trabajar con cadenas Unicode. Por ejemplo, si usa `substr()` sobre una cadena UTF-8, hay una gran posibilidad de que el resultado incluya algunos caracteres ilegibles. La función adecuada a usar es su contrapartida multibyte, `mb_substr()`.

La parte difícil es acordarse de usar las funciones `mb_*` todo el tiempo. Si lo olvida sólo una vez, su cadena Unicode podría quedar ilegible luego de su procesamiento posterior.

No todas las funciones de cadena tienen una contrapartida `mb_*`. Si no existe alguna para hacer lo que desea, entonces podría haber perdido su suerte.

Debe usarse la función `mb_internal_encoding()` al principio de cualquier script PHP que escriba (o en la parte superior de su script global), y la función `mb_http_output()` justo después, si su script genera salidas hacia un navegador. Definiendo explícitamente la codificación de sus cadenas en cada script le ahorrará muchos dolores de cabeza en el camino.

Adicionalmente, muchas de las funciones PHP que operan sobre cadenas tienen un parámetro opcional que le permite especificar la codificación de caracteres. Siempre se debe indicar explícitamente UTF-8 cuando se les da la opción. Por ejemplo, `htmlentities()` tiene una opción para la codificación de caracteres, y siempre se debe especificar UTF-8 si se trata de este tipo de cadenas. Tenga en cuenta que a partir de PHP 5.4.0, UTF-8 es la codificación por defecto para `htmlentities()` y `htmlspecialchars()`.

Por último, si usted está construyendo una aplicación distribuida y no está seguro de que la extensión `mbstring` se habilitará, entonces considere el uso del paquete de Composer patchwork/utf8. Esto le permitirá usar `mbstring` si está disponible, o usará funciones no UTF-8 en caso contrario.

UTF-8 a nivel de la Base de Datos

Si su script accede a MySQL, hay grandes posibilidades de que sus cadenas sean almacenadas como cadenas No-UTF-8 aún habiendo tomado todas las precauciones anteriores.

Para asegurarse de que sus cadenas van de PHP a MySQL como UTF-8, establezca el juego de caracteres y el cotejamiento (collation) a `utf8mb4` para su base de datos y todas las tablas, use también `utf8mb4` en su cadena de conexión PDO. Vea el código de ejemplo a continuación. Esto es de *importancia crítica*.

Tenga en cuenta de que debe usar el juego de caracteres `utf8mb4` y no `utf8` para un soporte completo UTF-8. Lea más adelante para saber porque qué.

UTF-8 a nivel del Navegador

Use la función `mb_http_output()` para asegurarse de que su script PHP genera una salida con cadenas UTF-8 hacia su navegador.

La respuesta HTTP le indicará al navegador que esa página debería considerarse como UTF-8. Anteriormente, para hacer esto, se incluía la etiqueta `<meta>` charset en el `<head>` de la página. Esta solución es perfectamente válida, pero establecer el juego de caracteres en la cabecera `Content-Type` es mucho más rápido.

```php
<?php
// Le decimos a PHP que usaremos cadenas UTF-8 hasta el final del script
mb_internal_encoding('UTF-8');
```

```php
// Le indicamos a PHP que necesitamos una salida UTF-8 hacia el navegador
mb_http_output('UTF-8');

// Nuestra cadena UTF-8 de prueba
$string = 'Êl síla erin lû e-govaned vîn.';

// Transformamos la cadena usando una función multibyte
// Observe que cortamos la cadena en un caracter NO-ASCII con propósitos de demostración
$string = mb_substr($string, 0, 15);

// Conectarse a la base de datos para almacenar la cadena transformada
// Observe el ejemplo PDO en este documento para más información
// ¡Observe el comando `set names utf8mb4`!
$link = new \PDO(
    'mysql:host=your-hostname;dbname=your-db;charset=utf8mb4',
    'your-username',
    'your-password',
    array(
        \PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION,
        \PDO::ATTR_PERSISTENT => false
    )
);

// Almacenamos nuestra cadena UTF-8 en nuestra base de datos
// Su BD y tablas están configuradas para usar el juego de caracteres y el cotejamiento (collation) utf8mb4, ¿correcto?
$handle = $link->prepare('insert into ElvishSentences (Id, Body) values (?, ?)');
$handle->bindValue(1, 1, PDO::PARAM_INT);
$handle->bindValue(2, $string);
$handle->execute();

// Recuperamos la cadena guardada para verificar que está correctamente almacenada
$handle = $link->prepare('select * from ElvishSentences where Id = ?');
$handle->bindValue(1, 1, PDO::PARAM_INT);
$handle->execute();

// Almacenamos el resultado en un objeto que nos permitirá generar HTML después
$result = $handle->fetchAll(\PDO::FETCH_OBJ);

header('Content-Type: text/html; charset=UTF-8');
?><!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Página de prueba UTF-8</title>
```

```
    </head>
    <body>
        <?php
        foreach($result as $row){
            print($row->Body);  // Esto debería mostrar correctamente en el navegador nuestra cadena UTF-8
        }
        ?>
    </body>
</html>
```

Otras lecturas

- PHP Manual: Operadores de Cadenas
- PHP Manual: Funciones de Cadenas
    - strpos()
    - strlen()
    - substr()
- PHP Manual: Funciones de Cadenas de Caracteres Multibyte
    - mb_strpos()
    - mb_strlen()
    - mb_substr()
    - mb_internal_encoding()
    - mb_http_output()
    - htmlentities()
    - htmlspecialchars()
- PHP UTF-8 Cheatsheet
- Stack Overflow: What factors make PHP Unicode-incompatible?
- Stack Overflow: Best practices in PHP and MySQL with international strings
- How to support full Unicode in MySQL databases
- Brining Unicode to PHP with Portable UTF-8

Volver al Inicio

Dependency Injection

From Wikipedia:

> *Dependency injection is a software design pattern that allows the removal of hard-coded dependencies and makes it possible to change them, whether at run-time or compile-time.*

This quote makes the concept sound much more complicated than it actually is. Dependency Injection is providing a component with it's dependencies either through constructor injection, method calls or the setting of properties. It is that

simple.

Basic Concept

We can demonstrate the concept with a simple, yet naive example.

Here we have a `Database` class that requires an adapter to speak to the database. We instantiate the adapter in the constructor and create a hard dependency. This makes testing difficult and means the `Database` class is very tightly coupled to the adapter.

```
<?php
namespace Database;

class Database
{
    protected $adapter;

    public function __construct()
    {
        $this->adapter = new MySqlAdapter;
    }
}

class MysqlAdapter {}
```

This code can be refactored to use Dependency Injection and therefore loosen the dependency.

```
<?php
namespace Database;

class Database
{
    protected $adapter;

    public function __construct(MySqlAdapter $adapter)
    {
        $this->adapter = $adapter;
    }
}

class MysqlAdapter {}
```

Now we are giving the `Database` class its dependency rather than it creating it itself. We could even create a method that would accept an argument of the dependency and set it that way, or if the `$adapter` property was `public` we could set it directly.

Complex Problem

If you have ever read about Dependency Injection then you have probably seen the terms *"Inversion of Control"* or *"Dependency Inversion Principle"*. These are the complex problems that Dependency Injection solves.

Inversion of Control

Inversion of Control is as it says, "inverting the control" of a system by keeping organisational control entirely separate from our objects. In terms of Dependency Injection, this means loosening our dependencies by controlling and instantiating them elsewhere in the system.

For years, PHP frameworks have been achieving Inversion of Control, however, the question became, which part of control are you inverting, and where to? For example, MVC frameworks would generally provide a super object or base controller that other controllers must extend to gain access to its dependencies. This **is** Inversion of Control, however, instead of loosening dependencies, this method simply moved them.

Dependency Injection allows us to more elegantly solve this problem by only injecting the dependencies we need, when we need them, without the need for any hard coded dependencies at all.

Dependency Inversion Principle

Dependency Inversion Principle is the "D" in the S.O.L.I.D set of object oriented design principles that states one should *"Depend on Abstractions. Do not depend on concretions."*. Put simply, this means our dependencies should be interfaces/contracts or abstract classes rather than concrete implementations. We can easily refactor the above example to follow this principle.

```php
<?php
namespace Database;

class Database
{
    protected $adapter;

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

```
interface AdapterInterface {}

class MysqlAdapter implements AdapterInterface {}
```

There are several benefits to the `Database` class now depending on an interface rather than a concretion.

Consider that you are working in a team and the adapter is being worked on by a colleague. In our first example, we would have to wait for said colleague to finish the adapter before we could properly mock it for our unit tests. Now that the dependency is an interface/contract we can happily mock that interface knowing that our colleague will build the adapter based on that contract.

An even bigger benefit to this method is that our code is now much more scalable. If a year down the line we decide that we want to migrate to a different type of database, we can write an adapter that implements the original interface and inject that instead, no more refactoring would be required as we can ensure that the adapter follows the contract set by the interface.

Containers

The first thing you should understand about Dependency Injection Containers is that they are not the same thing as Dependency Injection. A container is a convenience utility that helps us implement Dependency Injection, however, they can be and often are misused to implement an anti-pattern, Service Location. Injecting a DI container as a Service Locator in to your classes arguably creates a harder dependency on the container than the dependency you are replacing. It also makes your code much less transparent and ultimately harder to test.

Most modern frameworks have their own Dependency Injection Container that allows you to wire your dependencies together through configuration. What this means in practice is that you can write application code that is as clean and de-coupled as the framework it is built on.

Further Reading

- Learning about Dependency Injection and PHP
- What is Dependency Injection?
- Dependency Injection: An analogy
- Dependency Injection: Huh?
- Dependency Injection as a tool for testing

Volver al Inicio

Databases

Many times your PHP code will use a database to persist information. You have a few options to connect and interact with your database. The recommended option **until PHP 5.1.0** was to use native drivers such as mysqli, pgsql, mssql, etc.

Native drivers are great if you are only using *one* database in your application, but if, for example, you are using MySQL and a little bit of MSSQL, or you need to connect to an Oracle database, then you will not be able to use the same drivers. You'll need to learn a brand new API for each database — and that can get silly.

MySQL Extension

The mysql extension for PHP is no longer in active development, and is officially deprecated as of PHP 5.5.0, meaning that it will be removed within the next few releases. If you are using any functions that start with `mysql_*` such as `mysql_connect()` and `mysql_query()` in your applications then these will simply not be available in later versions of PHP. This means you will be faced with a rewrite at some point down the line, so the best option is to replace mysql usage with mysqli or PDO in your applications within your own development schedules so you won't be rushed later on.

If you are starting from scratch then absolutely do not use the mysql extension: use the MySQLi extension, or use PDO.

- PHP: Choosing an API for MySQL
- PDO Tutorial for MySQL Developers

PDO Extension

PDO is a database connection abstraction library — built into PHP since 5.1.0 — that provides a common interface to talk with many different databases. For example, you can use basically identical code to interface with MySQL or SQLite:

```
// PDO + MySQL
$pdo = new PDO('mysql:host=example.com;dbname=database', 'user', 'password');
$statement = $pdo->query("SELECT some\_field FROM some\_table");
$row = $statement->fetch(PDO::FETCH_ASSOC);
echo htmlentities($row['some_field']);

// PDO + SQLite
$pdo = new PDO('sqlite:/path/db/foo.sqlite');
$statement = $pdo->query("SELECT some\_field FROM some\_table");
$row = $statement->fetch(PDO::FETCH_ASSOC);
echo htmlentities($row['some_field']);
```

PDO will not translate your SQL queries or emulate missing features; it is purely for connecting to multiple types of database with the same API.

More importantly, `PDO` allows you to safely inject foreign input (e.g. IDs) into your SQL queries without worrying about database SQL injection attacks. This is possible using PDO statements and bound parameters.

Let's assume a PHP script receives a numeric ID as a query parameter. This ID should be used to fetch a user record from a database. This is the `wrong` way to do this:

```
<?php
$pdo = new PDO('sqlite:/path/db/users.db');
$pdo->query("SELECT name FROM users WHERE id = " . $_GET['id']); // <-- NO!
```

This is terrible code. You are inserting a raw query parameter into a SQL query. This will get you hacked in a heartbeat, using a practice called SQL Injection. Just imagine if a hacker passes in an inventive `id` parameter by calling a URL like `http://domain.com/?id=1%3BDELETE+FROM+users`. This will set the `$_GET['id']` variable to `1;DELETE FROM users` which will delete all of your users! Instead, you should sanitize the ID input using PDO bound parameters.

```
<?php
$pdo = new PDO('sqlite:/path/db/users.db');
$stmt = $pdo->prepare('SELECT name FROM users WHERE id = :id');
$stmt->bindParam(':id', $_GET['id'], PDO::PARAM_INT); // <-- Automatically sanitized by PDO
$stmt->execute();
```

This is correct code. It uses a bound parameter on a PDO statement. This escapes the foreign input ID before it is introduced to the database preventing potential SQL injection attacks.

- Learn about PDO

You should also be aware that database connections use up resources and it was not unheard-of to have resources exhausted if connections were not implicitly closed, however this was more common in other languages. Using PDO you can implicitly close the connection by destroying the object by ensuring all remaining references to it are deleted, i.e. set to NULL. If you don't do this explicitly, PHP will automatically close the connection when your script ends - unless of course you are using persistent connections.

- Learn about PDO connections

Interacting with Databases

When developers first start to learn PHP, they often end up mixing their database interaction up with their presentation logic, using code that might look like this:

```
<ul>
<?php
foreach ($db->query('SELECT * FROM table') as $row) {
    echo "<li>".$row['field1']." - ".$row['field1']."</li>";
}
?>
</ul>
```

This is bad practice for all sorts of reasons, mainly that its hard to debug, hard to test, hard to read and it is going to output a lot of fields if you don't put a limit on there.

While there are many other solutions to doing this - depending on if you prefer OOP or functional programming - there must be some element of separation.

Consider the most basic step:

```
<?php
function getAllFoos($db) {
    return $db->query('SELECT * FROM table');
}

foreach (getAllFoos($db) as $row) {
    echo "<li>".$row['field1']." - ".$row['field1']."</li>"; // BAD!!
}
```

That is a good start. Put those two items in two different files and you've got some clean separation.

Create a class to place that method in and you have a "Model". Create a simple `.php` file to put the presentation logic in and you have a "View", which is very nearly MVC - a common OOP architecture for most frameworks.

foo.php

```
<?php

$db = new PDO('mysql:host=localhost;dbname=testdb;charset=utf8mb4', 'username', 'password');

// Make your model available
include 'models/FooModel.php';

// Create an instance
$fooList = new FooModel($db);
```

```
// Show the view
include 'views/foo-list.php';
```

models/FooModel.php

```
<?php
class Foo()
{
    protected $db;

    public function __construct(PDO $db)
    {
        $this->db = $db;
    }

    public function getAllFoos() {
        return $this->db->query('SELECT * FROM table');
    }
}
```

views/foo-list.php

```
<? foreach ($fooList as $row): ?>
    <?= $row['field1'] ?> - <?= $row['field1'] ?>
<? endforeach ?>
```

This is essentially the same as what most modern frameworks are doing, all be it a little more manual. You might not need to do all of that every time, but mixing together too much presentation logic and database interaction can be a real problem if you ever want to unit-test your application.

PHPBridge have a great resource called Creating a Data Class which covers a very similar topic, and is great for developers just getting used to the concept of interacting with databases.

Abstraction Layers

Many frameworks provide their own abstraction layer which may or may not sit on top of PDO. These will often emulate features for one database system that is missing from another by wrapping your queries in PHP methods, giving you actual database abstraction instead of just the connection abstraction that PDO provides. This will of course add a little

overhead, but if you are building a portable application that needs to work with MySQL, PostgreSQL and SQLite then a little overhead will be worth it the sake of code cleanliness.

Some abstraction layers have been built using the PSR-0 or PSR-4 namespace standards so can be installed in any application you like:

- Aura SQL
- Doctrine2 DBAL
- Propel
- ZF2 Db

Volver al Inicio

Plantillas

Las plantillas proporcionan una manera conveniente de separar el controlador y la lógica de dominio de su lógica de presentación. Las plantillas normalmente contienen el HTML de tu aplicación, pero pueden ser usadas también para otros formatos, como XML. A las plantillas también se les llaman 'vistas', que constituyen parte del segundo componente del patrón de arquitectura de software modelo-vista-controlador (MVC).

Benefits

The main benefit to using templates is the clear separation they create between the presentation logic and the rest of your application. Templates have the sole responsibility of displaying formatted content. They are not responsible for data lookup, persistence or other more complex tasks. This leads to cleaner, more readable code which is especially helpful in a team environment where developers work on the server-side code (controllers, models) and designers work on the client-side code (markup).

Templates also improve the organization of presentation code. Templates are typically placed in a "views" folder, each defined within a single file. This approach encourages code reuse where larger blocks of code are broken into smaller, reusable pieces, often called partials. For example, your site header and footer can each be defined as templates, which are then included before and after each page template.

Finally, depending on the library you use, templates can offer more security by automatically escaping user-generated content. Some libraries even offer sand-boxing, where template designers are only given access to white-listed variables and functions.

Plain PHP Templates

Plain PHP templates are simply templates that use native PHP code. They are a natural choice since PHP is actually a template language itself. That simply means that you can combine PHP code within other code, like HTML. This is

beneficial to PHP developers as there is no new syntax to learn, they know the functions available to them, and their code editors already have PHP syntax highlighting and auto-completion built-in. Further, plain PHP templates tend to be very fast as no compiling stage is required.

Every modern PHP framework employs some kind of template system, most of which use plain PHP by default. Outside of frameworks, libraries like Plates or Aura.View make working with plain PHP templates easier by offering modern template functionality such as inheritance, layouts and extensions.

Simple example of a plain PHP template

Using the Plates library.

```
<?php // user_profile.php ?>

<?php $this->insert('header', ['title' => 'User Profile']) ?>

<h1>User Profile</h1>
<p>Hello, <?=$this->escape($name)?></p>

<?php $this->insert('footer') ?>
```

Example of plain PHP templates using inheritance

Using the Plates library.

```
<?php // template.php ?>

<html>
<head>
    <title><?=$title?></title>
</head>
<body>

<main>
    <?=$this->section('content')?>
</main>

</body>
</html>
```

```php
<?php // user_profile.php ?>

<?php $this->layout('template', ['title' => 'User Profile']) ?>

<h1>User Profile</h1>
<p>Hello, <?=$this->escape($name)?></p>
```

Compiled Templates

While PHP has evolved into a mature, object oriented language, it hasn't improved much as a templating language. Compiled templates, like Twig or Smarty*, fill this void by offering a new syntax that has been geared specifically to templating. From automatic escaping, to inheritance and simplified control structures, compiled templates are designed to be easier to write, cleaner to read and safer to use. Compiled templates can even be shared across different languages, Mustache being a good example of this. Since these templates must be compiled there is a slight performance hit, however this is very minimal when proper caching is used.

**While Smarty offers automatic escaping, this feature is NOT enabled by default.*

Simple example of a compiled template

Using the Twig library.

```
{% include 'header.html' with {'title': 'User Profile'} %}

<h1>User Profile</h1>
<p>Hello, {{ name }}</p>

{% include 'footer.html' %}
```

Example of compiled templates using inheritance

Using the Twig library.

```
// template.html

<html>
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
```

```
<main>
    {% block content %}{% endblock %}
</main>

</body>
</html>
```

```
// user_profile.html

{% extends "template.html" %}

{% block title %}User Profile{% endblock %}
{% block content %}
    <h1>User Profile</h1>
    <p>Hello, {{ name }}</p>
{% endblock %}
```

Further Reading

Articles & Tutorials

- Templating Engines in PHP
- An Introduction to Views & Templating in CodeIgniter
- Getting Started With PHP Templating
- Roll Your Own Templating System in PHP
- Master Pages
- Working With Templates in Symfony 2

Libraries

- Aura.View *(native)*
- Blade *(compiled, framework specific)*
- Dwoo *(compiled)*
- Latte *(compiled)*
- Mustache *(compiled)*
- PHPTAL *(compiled)*
- Plates *(native)*
- Smarty *(compiled)*
- Twig *(compiled)*
- Zend\View *(native, framework specific)*

Errors and Exceptions

Errors

In many “exception-heavy” programming languages, whenever anything goes wrong an exception will be thrown. This is certainly a viable way to do things, but PHP is an “exception-light” programming language. While it does have exceptions and more of the core is starting to use them when working with objects, most of PHP itself will try to keep processing regardless of what happens, unless a fatal error occurs.

For example:

```
$ php -a
php > echo $foo;
Notice: Undefined variable: foo in php shell code on line 1
```

This is only a notice error, and PHP will happily carry on. This can be confusing for those coming from “exception-heavy” languages, because referencing a missing variable in Python for example will throw an exception:

```
$ python
>>> print foo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'foo' is not defined
```

The only real difference is that Python will freak out over any small thing, so that developers can be super sure any potential issue or edge-case is caught, whereas PHP will keep on processing unless something extreme happens, at which point it will throw an error and report it.

Error Severity

PHP has several levels of error severity. The three most common types of messages are errors, notices and warnings. These have different levels of severity; `E_ERROR`, `E_NOTICE`, and `E_WARNING`. Errors are fatal run-time errors and are usually caused by faults in your code and need to be fixed as they’ll cause PHP to stop executing. Warnings are non-fatal errors, execution of the script will not be halted. Notices are advisory messages caused by code that may or may not cause problems during the execution of the script, execution is not halted.

Another type of error message reported at compile time are `E_STRICT` messages. These messages are used to suggest changes to your code to help ensure best interoperability and forward compatibility with upcoming versions of PHP.

Changing PHP's Error Reporting Behaviour

Error Reporting can be changed by using PHP settings and/or PHP function calls. Using the built in PHP function `error_reporting()` you can set the level of errors for the duration of the script execution by passing one of the predefined error level constants, meaning if you only want to see Warnings and Errors - but not Notices - then you can configure that:

```
error_reporting(E_ERROR | E_WARNING);
```

You can also control whether or not errors are displayed to the screen (good for development) or hidden, and logged (good for production). For more information on this check out the Error Reporting section.

Inline Error Suppression

You can also suppress specific errors from being displayed using the Error Control Operator `@`. You simply put this operator at the beginning an expression, and any error that would be caused as a direct result of the specific expression will be silenced.

```
echo @$foo['bar'];
```

This will output `$foo['bar']` if it exists, but will simply return a null and print nothing if the variable `$foo` or `'bar'` key does not exist.

This might seem like a good idea, but due to the way PHP actually handles `@` it is incredibly unperformant, and has the unexpected effect of still actually logging the error anyway. If you have this code in a loop that runs 100 times in an instance, and you run that 1 million times, then you've got 100 million lines in your logs.

Instead, use the following:

```
echo isset($foo['bar']) ? $foo['bar'] : '';
```

This will be much quicker, and save filling up your logs with junk. SitePoint go a step further and say you should never suppress notices with `@`.

One instance where error suppression might make sense is where `fopen()` fails to find a file to load. You could check for existence of the file before you try to load it, but if the file is deleted after the check and before the `fopen()` (which might sound impossible, but it can happen) then `fopen()` will return false *and* throw an error. This is potentially something PHP should resolve, but is one case where error suppression might seem like the only valid solution.

In a stock PHP system, the behavior of the error control operator is irreversible. There are no configuration settings which allow a suppressed error to be temporarily un-suppressed.

However, xDebug has an `xdebug.scream` ini setting which will disable the error control operator. You can set this via your `php.ini` file with the following.

```
xdebug.scream = On
```

You can also set this value at runtime with the `ini_set` function

```
ini_set('xdebug.scream', '1')
```

The "Scream" PHP extension offers similar functionality to xDebug's, although Scream's ini setting is named `scream.enabled`.

This is most useful when you're debugging code and suspect an informative error is suppressed. Use scream with care, and as a temporary debugging tool. There's lots of PHP library code that may not work with the error control operator disabled.

- Error Control Operators
- SitePoint
- never suppress notices
- xDebug
- Scream

ErrorException

PHP is perfectly capable of being an "exception-heavy" programming language, and only requires a few lines of code to make the switch. Basically you can throw your "errors" as "exceptions" using the `ErrorException` class, which extends the `Exception` class.

This is a common practice implemented by a large number of modern frameworks such as Symfony and Laravel. By default Laravel will display all errors as exceptions using the Whoops! package if the `app.debug` switch is turned on, then hide them if the switch is turned off.

By throwing errors as exceptions in development you can handle them better than the usual result, and if you see an exception during development you can wrap it in a catch statement with specific instructions on how to handle the situation. Each exception you catch instantly makes your application that little bit more robust.

More information on this and details on how to use `ErrorException` with error handling can be found at ErrorException Class.

- Error Control Operators
- Predefined Constants for Error Handling
- error_reporting
- Reporting

Exceptions

Exceptions are a standard part of most popular programming languages, but they are often overlooked by PHP programmers. Languages like Ruby are extremely Exception heavy, so whenever something goes wrong such as a HTTP request failing, or a DB query goes wrong, or even if an image asset could not be found, Ruby (or the gems being used) will throw an exception to the screen meaning you instantly know there is a mistake.

PHP itself is fairly lax with this, and a call to `file_get_contents()` will usually just get you a `FALSE` and a warning. Many older PHP frameworks like CodeIgniter will just return a false, log a message to their proprietary logs and maybe let you use a method like `$this->upload->get_error()` to see what went wrong. The problem here is that you have to go looking for a mistake and check the docs to see what the error method is for this class, instead of having it made extremely obvious.

Another problem is when classes automatically throw an error to the screen and exit the process. When you do this you stop another developer from being able to dynamically handle that error. Exceptions should be thrown to make a developer aware of an error; they then can choose how to handle this. E.g.:

```
<?php
$email = new Fuel\Email;
$email->subject('My Subject');
$email->body('How the heck are you?');
$email->to('guy@example.com', 'Some Guy');

try
{
    $email->send();
}
catch(Fuel\Email\ValidationFailedException $e)
{
    // The validation failed
}
catch(Fuel\Email\SendingFailedException $e)
{
    // The driver could not send the email
}
finally
{
```

```
    // Executed regardless of whether an exception has been thrown, and before normal execution resumes
}
```

SPL Exceptions

The generic `Exception` class provides very little debugging context for the developer; however, to remedy this, it is possible to create a specialized `Exception` type by sub-classing the generic `Exception` class:

```
<?php
class ValidationException extends Exception {}
```

This means you can add multiple catch blocks and handle different Exceptions differently. This can lead to the creation of a *lot* of custom Exceptions, some of which could have been avoided using the SPL Exceptions provided in the SPL extension.

If for example you use the `__call()` Magic Method and an invalid method is requested then instead of throwing a standard Exception which is vague, or creating a custom Exception just for that, you could just `throw new BadFunctionCallException;`.

- Read about Exceptions
- Read about SPL Exceptions
- Nesting Exceptions In PHP
- Exception Best Practices in PHP 5.3

Volver al Inicio

Seguridad

Seguridad en Aplicaciones Web

Es importante reconocer que existen individuos sin escrúpulos que buscan explotar e inhabilitar su aplicación web. Por esta razón es imperativo que tome las precauciones necesarias y trabaje en mejorar la seguridad de su proyecto. Afortunadamente, la comunidad del The Open Web Application Security Project (OWASP) han compilado una lista completa de problemas de seguridad conocidos y los métodos que puede seguir para protegerse contra ellos. Esta guía es lectura obligada para cualquier desarrollador consciente de la seguridad en su aplicación.

- Leer la Guía de seguridad de OWASP (Disponible solo en inglés)

Hash de contraseñas

Con el tiempo todo el mundo desarrolla una aplicación PHP que se apoya en la conexión del usuario. Los nombres de usuario y las contraseñas se almacenan en una base de datos y posteriormente son utilizados para autenticar a los usuarios en el login.

Es importante que usted realice hash de las contraseñas correctamente antes de almacenarlos. Una vez realizado el hash a la contraseña con una función aplicada contra la contraseña del usuario, el paso es irreversible. Esto produce una cadena de longitud fija que no puede ser revertida factiblemente. Esto significa que puede comparar un hash contra otro para determinar si ambos proceden de la misma cadena de origen, pero no se puede determinar la cadena original. Si las contraseñas no están hasheadas y se accede a la base de datos por un tercero no autorizado, todas las cuentas de usuario están comprometidas. Algunos usuarios pueden (por desgracia) utilizar la misma contraseña para otros servicios. Por lo tanto, es importante tomar en serio la seguridad.

Hash de contraseñas con `password_hash`

En PHP 5.5 fue introducido `password_hash`. En este momento está usando BCrypt, el algoritmo más fuerte actualmente soportado por PHP. Se actualizará en el futuro para apoyar a más algoritmos, según sea necesario. La librería `password_compat` fue creada para proveer compatibilidad a las versiones PHP >= 5.3.7.

Más abajo hacemos hash a una cadena de texto, y seguidamente comprobamos el hash con una nueva cadena. Debido a que las fuentes de nuestras dos cadenas son distintas (‘contraseña-secreta’ vs ‘mala-contraseña’) el login fallará.

```php
<?php

require 'password.php';

$passwordHash = password_hash('contraseña-secreta', PASSWORD_DEFAULT);

if (password_verify('mala-contraseña', $passwordHash)) {
    // Contraseña correcta
} else {
    // Contraseña incorrecta
}
```

- Aprende más acerca de `password_hash`
- `password_compat` para PHP >= 5.3.7 && < 5.5
- Obtener información acerca de hashing en cuanto a la criptografía
- PHP `password_hash` RFC

Filtrado de Datos

Nunca, jamás, se confíe de los datos que provienen del exterior de su aplicación PHP. Siempre sanee y verifique los datos de entrada antes de usarlos en su código. Las funciones `filter_var()` y `filter_input()` proporcionan saneamiento de los datos y verifican la validez del formato del texto (por ejemplo, las direcciones de correo electrónico).

Los datos de entrada exteriores pueden contener cualquier cosa: los datos provenientes de formularios en `$_GET` y `$_POST`, algunos valores provenientes del súper global `$_SERVER`, el cuerpo de la solicitud HTTP vía la función `fopen('php://input', 'r')`. Recuerde que la entrada exterior de datos no está limitada a los datos que provienen de un usuario a través de un formulario, también provienen de la subida y descarga de archivos, valores de sesión, datos de cookies, y los servicios web exteriores.

Aunque los datos que provienen del exterior pueden ser guardados, combinados y se puede acceder a ellos posteriormente, todavía siguen siendo datos exteriores. Cada vez que procesa, imprime, concatena, o los incluye con otros datos en su código, pregúntese si los datos han sido filtrados apropiadamente y si es confiable.

Los datos pueden ser *filtrados* de diferente manera dependiendo de su proposito. Por ejemplo, cuando se pasan datos sin filtrar a la salida de HTML de su página, corre el riesgo de ejecutar código sin verificar de HTML o JavaScript en su sitio. Esto es conocido (en inglés) como Cross-Site Scripting (XSS), y puede causar mucho daño a su aplicación. Una manera de prevenir estos ataques es sanear todas las etiquetas de HTML en sus datos de entrada al remover etiquetas o convirtiéndolas en entidades de HTML.

Otro ejemplo es cuando se pasan opciones que van a ser ejecutadas en la línea de comando. Esto puede ser extremadamente peligroso y en general no es una buena idea. Sin embargo, puede usar la función `escapeshellarg` que viene incluida en PHP para sanear los argumentos de ejecución de un comando.

Un último ejemplo es el de aceptar la entrada de datos para determinar que archivo se necesita cargar del sistema de archivos. Esto puede ser explotado al cambiar el nombre del archivo a una ruta de archivo diferente. En este caso es necesario remover los caracteres "/", "../", bytes nulos y otros de la ruta de archivo para que no permita cargar archivos escondidos, no públicos, o de otra manera sensitivos.

- Aprenda acerca del filtrado de datos
- Aprenda acerca de la función `filter_var`
- Aprenda acerca de la función `filter_input`
- Aprenda como manipular los bytes nulos

Saneamiento

El saneamiento remueve (o evita) los caracteres ilegales o inseguros que provienen de la entrada de datos externa.

Por ejemplo, debería sanear la entrada de datos antes de incluirla en el código HTML o insertarla a una consulta de SQL. Cuando utiliza parámetros consolidados con PDO, este saneara la entrada de datos automáticamente.

En veces es requerido que permita que algunas etiquetas de HTML que se consideren inofensivas puedan ser incluidas en la entrada de datos para una página de su sitio. Esto es algo muy difícil de realizar de una manera segura y muchos lo

evitan al usar otros estándares de formato de texto más restrictivos como Markdown o BBCode, aunque librerías que proveen una lista blanca de etiquetas como el HTML Purifier se concibieron para este propósito.

Vea los filtros de saneamiento

Validación

La validación asegura que lo que contiene la entrada de datos es lo que usted esperaba. Por ejemplo, quizás desee validar una dirección de correo electrónico, un número telefónico, o la edad de un usuario al procesar una solicitud de registro.

Vea los filtros de validación

Archivos de Configuración

Cuando esté trabajando con archivos de configuración para su aplicación, las mejores prácticas dictan que utilice uno de los métodos siguientes:

- Es recomendable que almacene sus archivos de configuración donde no se pueda acceder a ellos directamente por medio del sistema de archivos.
- Si no tiene otra alternativa más que almacenar sus archivos de configuración en la raíz de documentos de su aplicación, debe adjuntar la extensión `.php` al nombre de sus archivos. De esta manera, aun si alguien accede a ellos directamente, la información no será impresa en forma de texto a la pantalla.
- La información en los archivos de configuración debe ser protegida ya sea por medio de codificación o con los permisos de grupo/usuario pertinentes en el sistema de archivos.

Register Globals

NOTA: Desde el lanzamiento de PHP 5.4, la opción `register_globals` ha sido eliminada y ya no puede ser utilizada.

Cuando se habilita la opción `register_globals` en la configuración de PHP, varios tipos de variables (incluidos los de `$_POST`, `$_GET` y `$_REQUEST`) se hacen disponibles en el ámbito global de su aplicación. Esto puede muy fácilmente conducirlo a problemas de seguridad ya que su aplicación no tiene una manera efectiva de verificar de dónde provienen esos datos.

Si está usando una versión de PHP anterior a 4.2.0, tenga en cuenta que corre el riesgo de que esta opción le cause problemas. Desde la versión 4.2.0 en adelante, la opción `register_globals` viene desactivada. Para garantizar la seguridad de su aplicación, asegúrese de que esta opción **siempre** este desactivada, con el valor “off”, si es que está disponible.

- Register_globals en el manual de PHP

Reporte de Errores

El registro de errores puede ser muy útil para encontrar los lugares donde existen problemas en su aplicación, pero también puede exponer información acerca de la estructura de su aplicación al exterior. Para proteger efectivamente su aplicación de problemas que pudieran ser causados por la salida de mensajes de error, necesita configurar su servidor de desarrollo de diferente manera que su servidor de producción.

Desarrollo

Para mostrar los errores durante el **desarrollo** de su aplicación, configure las siguientes opciones en su archivo `php.ini`:

- display_errors: On
- error_reporting: E_ALL
- log_errors: On

Producción

Para esconder los errores en su entorno de **producción**, configure su archivo `php.ini` de la siguiente manera:

- display_errors: Off
- error_reporting: E_ALL
- log_errors: On

Con estas opciones en su entorno de producción, los errores seguirán siendo registrados en los registros de errores de su servidor web, pero no serán mostrados al usuario. Para más información en cuanto a estas opciones, vea el manual de PHP:

- Error_reporting
- Display_errors
- Log_errors

Volver al Inicio

Pruebas

La creación de pruebas automatizadas para su código PHP es considerada una buena práctica y conduce a aplicaciones sólidamente construidas. Las pruebas automatizadas son una gran herramienta que asegura que su aplicación no sea afectada negativamente al hacer cambios o al añadir nuevas características a su código; estas son ventajas que no se puede ignorar.

Existen diferentes tipos de herramientas (o armazones) para pruebas disponibles en PHP, las cuales tiene enfoques diferentes, pero todas intentan eliminar las pruebas manuales y la necesidad de disponer grandes equipos de Control de Calidad que se encarguen de asegurar que la aplicación sigue trabajando bien cuando introduce nuevos cambios al código.

Test Driven Development

From Wikipedia:

> *Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes a failing automated test case that defines a desired improvement or new function, then produces code to pass that test and finally refactors the new code to acceptable standards. Kent Beck, who is credited with having developed or 'rediscovered' the technique, stated in 2003 that TDD encourages simple designs and inspires confidence*

There are several different types of testing that you can do for your application

Unit Testing

Unit Testing is a programming approach to ensure functions, classes and methods are working as expected, from the point you build them all the way through the development cycle. By checking values going in and out of various functions and methods, you can make sure the internal logic is working correctly. By using Dependency Injection and building "mock" classes and stubs you can verify that dependencies are correctly used for even better test coverage.

When you create a class or function you should create a unit test for each behavior it must have. At a very basic level you should make sure it errors if you send it bad arguments and make sure it works if you send it valid arguments. This will help ensure that when you make changes to this class or function later on in the development cycle that the old functionality continues to work as expected. The only alternative to this would be var_dump() in a test.php, which is no way to build an application - large or small.

The other use for unit tests is contributing to open source. If you can write a test that shows broken functionality (i.e. fails), then fix it, and show the test passing, patches are much more likely to be accepted. If you run a project which accepts pull requests then you should suggest this as a requirement.

PHPUnit is the de-facto testing framework for writing unit tests for PHP applications, but there are several alternatives

- atoum
- Enhance PHP
- PUnit
- SimpleTest

Integration Testing

From Wikipedia:

> *Integration testing (sometimes called Integration and Testing, abbreviated "I&T") is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.*

Many of the same tools that can be used for unit testing can be used for integration testing as many of the same principles are used.

Functional Testing

Sometimes also known as acceptance testing, functional testing consists of using tools to create automated tests that actually use your application instead of just verifying that individual units of code are behaving correctly and that individual units can speak to each other correctly. These tools typically work using real data and simulating actual users of the application.

Functional Testing Tools

- Selenium
- Mink
- Codeception is a full-stack testing framework that includes acceptance testing tools
- Storyplayer is a full-stack testing framework that includes support for creating and destroying test environments on demand

Desarrollo Guiado por Comportamiento

Existen dos tipos diferentes de Desarrollo Guiado por Comportamiento (BDD, por sus siglas en inglés): SpecBDD y StoryBDD. SpecBDD se enfoca en el comportamiento o funcionamiento técnico o de parte del código, mientras que StoryBDD se enfoca en el funcionamiento de negocios, el comportamiento de las características o las interacciones. PHP proporciona armazones para estos dos tipos de BDD.

Con el StoryBDD, se escriben historias legibles que describen el comportamiento de su aplicación. Estas historias pueden ser ejecutadas como pruebas en contra de su aplicación. El armazón que se utiliza en PHP para StoryBDD se llama Behat, el cual fue inspirado por el proyecto Cucumber de Ruby e implementa el Gherking DSL para describir el comportamiento de características.

Con el SpecBDD, se escriben las especificaciones que describen como su código debe de funcionar. En vez de probar una función o método, solo se describe como la función o método se tiene que comportar. PHP ofrece el armazón PHPSpec

para este propósito. Este armazon fue inspirado por el proyecto RSpec de Ruby.

Enlaces de BDD

- Behat el armazón de StoryBDD para PHP
- PHPSpec el armazón deSpecBDD paraPHP
- Codeception es un armazón completo para pruebas que utiliza los principios de BDD

Herramientas Complementarias de Pruebas

Además de armazones para pruebas individuales y guiados por comportamiento, también existen varios armazones genéricos y librerías auxiliares que son útiles para cualquier enfoque que se tome.

Enlaces de herramientas

- Selenium es una herramienta de automatización del navegador que puede ser integrada a PHPUnit
- Mockery es un armazón para objetos de maqueta que puede ser integrado con PHPUnit o PHPSpec

Volver al Inicio

Servidores y Despliegue

Las aplicaciones PHP se pueden desplegar y ejecutar en servidores web de producción de varias maneras.

Plataforma como Servicio (PaaS)

Las aplicaciones PHP se pueden desplegar y ejecutar en servidores web de producción de varias maneras.

PaaS provee la arquitectura adecuada del sistema y la red que se necesitan para ejecutar aplicaciones PHP en la web. Esto significa que no es necesaria una configuración extensa del servidor para lanzar aplicaciones o armazones de PHP.

Recientemente, PaaS se ha convertido en un método muy popular de desplegar, alojar, y ampliar aplicaciones PHP de todos los tamaños. Encontrará una lista de **Proveedores de PHP PaaS** en la sección de recursos.

Servidores Virtuales o Dedicados

Si se siente cómodo trabajando en la administración de sistemas, o está interesado en aprender a hacerlo, entonces los servidores virtuales o dedicados le proporcionaran el control completo del entorno de producción de su aplicación.

nginx y PHP-FPM

PHP, via el Gestor de Procesos FastCGI de PHP (FPM, con sus siglas en inglés), hace buen par con el servidor nginx, que es un servidor ligero de alto rendimiento. Utiliza menos memoria que Apache y puede manejar más solicitudes simultáneas. Esto es de especial importancia en servidores que no disponen de mucha memoria.

- Leer más sobre nginx
- Leer más sobre PHP-FPM
- Leer más sobre configurar, y asegurar, nginx y PHP-FPM

Apache y PHP

PHP y Apache gozan de una larga historia juntos. Apache es muy configurable y tiene muchos módulos disponibles que extienden las características del servidor. Es una elección popular para servidores compartidos y es fácil de configurar armazones de desarrollo y aplicaciones como WordPress para que trabajen en el servidor. Desafortunadamente, Apache utiliza más recursos que nginx y por defecto no está configurado para manejar un alto número de visitantes al mismo tiempo.

Es posible configurar Apache para ejecutar PHP de diferentes maneras. La más común y más fácil de configurar es usando el módulo `mod_php5` con el prefork MPM. Aunque no es el modulo más eficiente en cuanto al uso de la memoria, es el mas fácil de utilizar. Esta opción es la más viable si no está interesado en adentrarse en los aspectos de administrar un servidor. Cabe notar que si usa el módulo `mod_php5`, tiene que usar el *prefork MPM*.

Alternativamente, si desea sacar más rendimiento y estabilidad de Apache, entones puede tomar ventaja del mismo sistema de FPM que usa nginx y utilizar el worker MPM o el event MPM con mod_fastcgi o mod_fcgid. Esto le proporcionara una configuración más eficiente en memoria y mucho más rápida, aunque es más difícil de configurar.

- Leer más sobre
- Leer más sobre los módulos para Multi-Processing
- Leer más sobre mod_fastcgi
- Leer más sobre mod_fcgid

Servidores Compartidos

PHP es un lenguaje popular gracias a su proliferación en servidores compartidos. Es difícil encontrarse con un servicio de alojamiento que no incluya PHP en sus opciones, sin embargo es importante que sea la versión más nueva. Los servidores compartidos le permiten a varios desarrolladores desplegar sitios web en un solo sistema. La ventaja de esto es que se ha convertido en una mercancía barata. La desventaja es que nunca sabe qué tipo de alboroto crearan sus inquilinos vecinos; La sobrecarga del servidor y los huecos abiertos de seguridad serían sus preocupaciones principales. Por esta razón, es recomendable evitar este tipo de servicios si el presupuesto de su proyecto lo permite.

Building and Deploying your Application

If you find yourself doing manual database schema changes or running your tests manually before updating your files (manually), think twice! With every additional manual task needed to deploy a new version of your app, the chances for potentially fatal mistakes increase. Whether you're dealing with a simple update, a comprehensive build process or even a continuous integration strategy, build automation is your friend.

Among the tasks you might want to automate are:

- Dependency management
- Compilation, minification of your assets
- Running tests
- Creation of documentation
- Packaging
- Deployment

Build Automation Tools

Build tools can be described as a collection of scripts that handle common tasks of software deployment. The build tool is not a part of your software, it acts on your software from 'outside'.

There are many open source tools available to help you with build automation, some are written in PHP others aren't. This shouldn't hold you back from using them, if they're better suited for the specific job. Here are a few examples:

Phing is the easiest way to get started with automated deployment in the PHP world. With Phing you can control your packaging, deployment or testing process from within a simple XML build file. Phing (which is based on Apache Ant) provides a rich set of tasks usually needed to install or update a web app and can be extended with additional custom tasks, written in PHP.

Capistrano is a system for *intermediate-to-advanced programmers* to execute commands in a structured, repeatable way on one or more remote machines. It is pre-configured for deploying Ruby on Rails applications, however people are **successfully deploying PHP systems** with it. Successful use of Capistrano depends on a working knowledge of Ruby and Rake.

Dave Gardner's blog post PHP Deployment with Capistrano is a good starting point for PHP developers interested in Capistrano.

Chef is more than a deployment framework, it is a very powerful Ruby based system integration framework that doesn't just deploy your app but can build your whole server environment or virtual boxes.

Chef resources for PHP developers:

- Three part blog series about deploying a LAMP application with Chef, Vagrant, and EC2
- Chef Cookbook which installs and configures PHP 5.3 and the PEAR package management system

Further reading:

- Automate your project with Apache Ant
- Maven, a build framework based on Ant and how to use it with PHP

Continuous Integration

> *Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily — leading to multiple integrations per day. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.*

– Martin Fowler

There are different ways to implement continuous integration for PHP. Recently Travis CI has done a great job of making continuous integration a reality even for small projects. Travis CI is a hosted continuous integration service for the open source community. It is integrated with GitHub and offers first class support for many languages including PHP.

Further reading:

- Continuous Integration with Jenkins
- Continuous Integration with PHPCI
- Continuous Integration with Teamcity

Volver al Inicio

Virtualización

Ejecutar tu aplicación en diferentes entornos en desarrollo y producción puede llevar a extraños errores que aparecen en vivo. También es complicado mantener diferentes ambientes de desarrollo actualizados con las mismas versiones de todas las bibliotecas utilizadas cuando se trabaja con un equipo de desarrolladores.

Si estás desarrollando en Windows y desplegando en Linux (o cualquier cosa que no sea Windows) o si estás desarrollando en un equipo, deberías considerar el uso de una máquina virtual. Esto suena complicado, pero además de los entornos de virtualización ampliamente conocidos como VMware o VirtualBox, hay herramientas adicionales que pueden ayudarte a configurar un entorno virtual en unos pocos pasos sencillos.

Vagrant

Vagrant te ayuda a construir cajas virtuales (virtual boxes) sobre entornos virtuales conocidos y configurará estos entornos usando un simple archivo de configuración. Estas cajas pueden configurarse manualmente o puedes usar software “aprovisionado” como Puppet o Chef para que haga todo el trabajo por ti. Suministrar la base para una caja es una excelente manera de asegurarte de que múltiples cajas serán configuradas de forma idéntica y elimina la necesidad

de mantener complicadas listas de comandos de “configuración”. También puedes “destruir” una caja base y volver a crearla en unos pocos pasos, lo que facilita la creación de una instalación “nueva”.

Vagrant crea carpetas para compartir tu código entre tu host y tu máquina virtual, lo que significa que puedes crear y editar tus archivos en tu máquina host y luego ejecutar el código dentro de tu máquina virtual.

Una pequeña ayuda

Si necesitas ayuda para empezar a usar Vagrant, estos son algunos servicios que podrían serte muy útiles:

- Rove: servicio que te permitirá “pre-generar” builds Vagrant típicos, PHP está entre sus opciones. El “aprovisionamiento” es hecho con Chef.
- Puphpet: Sencilla interfase gráfica de usuario (GUI por sus siglas en inglés) para configurar máquinas virtuales para desarrollo con PHP. **Fuertemente enfocadas en PHP**. Además de las MVs (máquinas virtuales) locales, Puphept también puede ser usado para hacer despliegues hacia servicios en la nube. El “aprovisionamiento” es hecho con Puppet.
- Protobox: Es una capa en el tope de Vagrant y una interfase web para configurar máquinas virtuales para desarrollo web. Un simple documento YAML controla todo lo que será instalado en la MV.
- Phansible: proporciona una sencilla interfase que te facilita la generación de Ansible Playbooks para proyectos basados en PHP.

Docker

Beside using Vagrant, another easy way to get a virtual development or production environment up and running is Docker. Docker helps you to provide Linux containers for all kind of applications. There are many helpful docker images which could provide you with other great services without the need to install these services on your local machine, e.g. MySQL or PostgreSQL and a lot more. Have a look at the Docker Hub Registry to search a list of available pre-built containers, which you can then run and use in very few steps.

Example: Runnning your PHP Applications in Docker

After you installed docker on your machine, you can start an Apache with PHP support in one step. The following command will download a fully functional Apache installation with the latest PHP version and provide the directory `/path/to/your/php/files` at `http://localhost:8080`:

```
docker run -d --name my-php-webserver -p 8080:80 -v /path/to/your/php/files:/var/www/html/ php:apache
```

After running `docker run` your container is initialized and running. If you would like to stop or start your container again, you can use the provided name attribute and simply run `docker stop my-php-webserver` and `docker start my-php-webserver` without providing the above mentioned parameters again.

Learn more about Docker

The commands mentioned above only show a quick way to run an Apache web server with PHP support but there are a lot more things that you can do with Docker. One of the most important things for PHP developers will be linking your web server to a database instance, for example. How this could be done is well described within the Docker User Guide.

- Docker Website
- Docker Installation
- Docker Images at the Docker Hub Registry
- Docker User Guide

Volver al Inicio

Caché

En sí, PHP es muy rápido, pero pueden surgir embotellamientos cuando realiza conexiones remotas, carga archivos y así por el estilo. Afortunadamente, existen varias herramientas disponibles para acelerar ciertas partes de su aplicación o reducir el número de veces que se ejecutan estas tareas que consumen tanto tiempo.

Opcode Cache

When a PHP file is executed, under the hood it is first compiled to opcodes and, only then, the opcodes are executed. If a PHP file is not modified, the opcodes will always be the same. This means that the compilation step is a waste of CPU resources.

This is where opcode caches come in. They prevent redundant compilation by storing opcodes in memory and reusing it on successive calls. Setting up an opcode cache takes a matter of minutes, and your application will speed up significantly. There's really no reason not to use it.

As of PHP 5.5, there is a built-in opcode cache called OPcache. It is also available for earlier versions.

Read more about opcode caches:

- OPcache (built-in since PHP 5.5)
- APC (PHP 5.4 and earlier)
- XCache
- Zend Optimizer+ (part of Zend Server package)
- WinCache (extension for MS Windows Server)
- list of PHP accelerators on Wikipedia

Object Caching

There are times when it can be beneficial to cache individual objects in your code, such as with data that is expensive to get or database calls where the result is unlikely to change. You can use object caching software to hold these pieces of data in memory for extremely fast access later on. If you save these items to a data store after you retrieve them, then pull them directly from the cache for following requests, you can gain a significant improvement in performance as well as reduce the load on your database servers.

Many of the popular bytecode caching solutions let you cache custom data as well, so there's even more reason to take advantage of them. APCu, XCache, and WinCache all provide APIs to save data from your PHP code to their memory cache.

The most commonly used memory object caching systems are APCu and memcached. APCu is an excellent choice for object caching, it includes a simple API for adding your own data to its memory cache and is very easy to setup and use. The one real limitation of APCu is that it is tied to the server it's installed on. Memcached on the other hand is installed as a separate service and can be accessed across the network, meaning that you can store objects in a hyper-fast data store in a central location and many different systems can pull from it.

Note that when running PHP as a (Fast-)CGI application inside your webserver, every PHP process will have its own cache, i.e. APCu data is not shared between your worker processes. In these cases, you might want to consider using memcached instead, as it's not tied to the PHP processes.

In a networked configuration APCu will usually outperform memcached in terms of access speed, but memcached will be able to scale up faster and further. If you do not expect to have multiple servers running your application, or do not need the extra features that memcached offers then APCu is probably your best choice for object caching.

Example logic using APCu:

```php
<?php
// check if there is data saved as 'expensive_data' in cache
$data = apc_fetch('expensive_data');
if ($data === false) {
    // data is not in cache; save result of expensive call for later use
    apc_add('expensive_data', $data = get_expensive_data());
}

print_r($data);
```

Note that prior to PHP 5.5, APC provides both an object cache and a bytecode cache. APCu is a project to bring APC's object cache to PHP 5.5+, since PHP now has a built-in bytecode cache (OPcache).

Learn more about popular object caching systems:

- APCu
- APC Functions

- Memcached
- Redis
- XCache APIs
- WinCache Functions

Volver al Inicio

Recursos

Directamente de la fuente

- Sitio web de PHP
- Documentación de PHP

Gente a seguir

- Rasmus Lerdorf
- Fabien Potencier
- Derick Rethans
- Chris Shiflett
- Sebastian Bergmann
- Matthew Weier O'Phinney
- Nikita Popov

Mentores

- phpmentoring.org - Asesoramiento formal y directo para la comunidad de PHP.

Proveedores de PHP PaaS

- PagodaBox
- PHP Fog
- Engine Yard Orchestra PHP Platform
- Red Hat OpenShift Platform
- dotCloud
- AWS Elastic Beanstalk
- cloudControl
- Windows Azure

Frameworks

En lugar de reinventar la rueda, muchos desarrolladores de PHP usan **Frameworks** para desarrollar nuevas aplicaciones web. Los Frameworks abstraen mucho del funcionamiento fundamental de la aplicación y proveen interfaces útiles y fáciles de usar para desempeñar tareas comunes.

No es necesario usar un framework en todos sus proyectos. A veces la mejor alternativa es utilizar PHP puro, pero si cree que necesitas un framework hay tres tipos disponibles:

- Micro Frameworks
- Frameworks Full-Stack (Completos)
- Frameworks de Componentes

Los **micro frameworks** son, en esencia, una envoltura para dirigir las solicitudes de HTTP a un controlador o método lo más rápido posible, y en veces incluyen otras librerías para asistir en el desarrollo, tal como envolturas básicas para bases de datos y así por el estilo. Estos frameworks son utilizados en su mayoría para desarrollar servicios remotos de HTTP.

Muchos frameworks añaden un número considerable de funciones y características sobre la base disponible en un micro framework. A estos se les conoce como **frameworks full-stack (completos)**. A menudo, estos frameworks incluyen paquetes ORM para bases de datos, paquetes de autenticación, y otros.

Los **frameworks de componentes** son colecciones de librerías especializadas para un propósito en particular. Los componentes de estos frameworks pueden utilizarse con otras librerías para construir micro frameworks o frameworks full-stack (completos).

- Frameworks populares de PHP

Components

As mentioned above “Components” are another approach to the common goal of creating, distributing and implementing shared code. Various component repositories exist, the main two of which are:

- Packagist
- PEAR

Both of these repositories have command line tools associated with them to help the installation and upgrade processes, and have been explained in more detail in the Dependency Management section.

There are also component-based frameworks, which allow you to use their components with minimal (or no) requirements. For example, you can use the FuelPHP Validation package, without needing to use the FuelPHP framework itself. These projects are essentially just another repository for reusable components:

- Aura
- FuelPHP
- Symfony Components
- The League of Extraordinary Packages
- Laravel's Illuminate Components
    - Eloquent ORM
    - Queue

Laravel's Illuminate components will become better decoupled from the Laravel framework. For now, only the components best decoupled from the Laravel framework are listed above.

Volver al Inicio

Comunidad

La comunidad de PHP es una comunidad grande y con mucha diversidad. Sus miembros están listos y dispuestos para apoyar a los nuevos programadores en PHP. Usted debería considerar la posibilidad de unirse a un grupo local de usuarios de PHP (PUG, con sus siglas en inglés) o atender una de las conferencias enfocadas en PHP para aprender más acerca de las mejores prácticas expuestas en esta página. También puede asociarse en el canal de IRC, #phpc, disponible en irc.freenode.com y seguir la cuenta de twitter @phpc. Le exhortamos a que conozca a otros desarrolladores, aprenda más acerca de otros tópicos y, sobre todo, entable nuevas amistades.

Lea el calendario oficial de eventos de PHP

Grupos de Usuarios de PHP

Si usted vive en una ciudad grande, es muy probable que exista un grupo de usuarios de PHP cerca. Aunque no existe todavía una lista oficial de estos grupos, si puede encontrar fácilmente un grupo local usando Google o Meetup.com. Si usted vive en una ciudad más pequeña, quizás no haya grupos de usuarios locales. Si esa es su situación, ¡lo invitamos a que inicie su propio grupo!

Lea más acerca de los grupos de usuarios en el wiki de PHP

Conferencias de PHP

La comunidad de PHP también organiza conferencias de ámbito regional y nacional en muchos países alrededor del mundo. Miembros bien conocidos de la comunidad de PHP usualmente tienen intervenciones en estos eventos así que es una buena oportunidad para aprender directamente de estos líderes de la industria.

Encuentre una conferencia de PHP

PHPDoc

PHPDoc is an informal standard for commenting PHP code. There are a *lot* of different tags available. The full list of tags and examples can be found at the PHPDoc manual.

Below is an example of how you might document a class with a few methods;

```php
<?php
/**
 * @author A Name <a.name@example.com>
 * @link http://www.phpdoc.org/docs/latest/index.html
 * @package helper
 */
class DateTimeHelper
{
    /**
     * @param mixed $anything Anything that we can convert to a \DateTime object
     *
     * @return \DateTime
     * @throws \InvalidArgumentException
     */
    public function dateTimeFromAnything($anything)
    {
        $type = gettype($anything);

        switch ($type) {
            // Some code that tries to return a \DateTime object
        }

        throw new \InvalidArgumentException(
            "Failed Converting param of type '{$type}' to DateTime object"
        );
    }

    /**
     * @param mixed $date Anything that we can convert to a \DateTime object
     *
     * @return void
     */
    public function printISO8601Date($date)
    {
```

```
        echo $this->dateTimeFromAnything($date)->format('c');
    }

    /**
     * @param mixed $date Anything that we can convert to a \DateTime object
     */
    public function printRFC2822Date($date)
    {
        echo $this->dateTimeFromAnything($date)->format('r');
    }
}
```

The documentation for the class as a whole firstly has the @author tag, this tag is used to document the author of the code and can be repeated for documenting several authors. Secondly is the @link tag, used to link to a website indicating a relationship between the website and the code. Thirdly it has the @package tag, used to categorize the code.

Inside the class, the first method has an @param tag documenting the type, name and description of the parameter being passed to the method. Additionally it has the @return and @throws tags for documenting the return type, and any exceptions that could be throw respectively.

The second and third methods are very similar and have a single @param tag as did the first method. The import difference between the second and third method is doc block is the inclusion/exclusion of the @return tag. `@return void` explicitly informs us that there is no return, historically omitting the `@return void` statement also results in the same (no return) action.

Volver al Inicio

Créditos

Creado y mantenido por
Josh Lockhart

Colaboradores del proyecto
Kris Jordan
Phil Sturgeon

Contribuyentes al proyecto
codeguy
joseayram
KrisJordan
grakic

stevenbenner
mantrax314
3rn3st0
chartjes
isra00
xosofox
oyepez003
zeroecco
ricval
auroraeosrose
eoconnell
getjump
inoryy
Blackshawk
padraic
tvlooy
wilmoore
ziadoz
enygma
jcarouth
AlexBroadcast
agarzon
briannesbitt
SyntaxC4
iJanki
dmouse
Xeoncross
eoinoc
coudenysj
xiongchiamiov
jeremeamia
JoelSutherland
MattHeard
rdohms
robertboloc
edorian
huglester
primitive-type
tot-ra

Patrocinadores del proyecto

New Media Campaigns