PHP

PHP HA RECORRIDO UN LARGO CAMINO Y SE POSICIONA DE MANERA INMEJORABLE EN EL ÁMBITO DE LAS APLICACIONES WEB, PERO SUS POSIBILIDADES CRECEN DÍA A DÍA, TAMBIÉN, EN OTROS ESPACIOS DE TRABAJO O EN FORMA RELACIONADA CON OTRAS TECNOLOGÍAS. PARA MEJORAR NUESTRO NIVEL Y CONOCER SUS PROS Y SUS CONTRAS, NADA MEJOR QUE IMPLEMENTAR TÉCNICAS DE PROGRAMACIÓN EN VARIAS CAPAS, MEJORAR NUESTRA PRODUCTIVIDAD Y ACELERAR EL RENDIMIENTO SIN DEJAR DE ANALIZAR LA SEGURIDAD GENERAL DE NUESTROS PROYECTOS. A CONTINUACIÓN, UN REPASO PROFUNDO POR EL PASADO, EL PRESENTE Y EL FUTURO DE ESTA TECNOLOGÍA.

por Cristian H. Gaitano Ornia
Analista de sistemas
ornia@arnet.com.ar

Como sabrán, PHP es un lenguaje de programación interpretado, que fue desarrollado inicialmente en 1994 por Rasmus Lerdorf. Si bien él es el creador y quien dio el impulso inicial, en su evolución, la plataforma fue acumulando un conjunto muy amplio de mejorías y rescrituras, producto del esfuerzo de una gran comunidad de desarrolladores. Por eso mismo, PHP está tan arraigado y crece en forma tan vertiginosa, actualizándose en función de las necesidades de la sociedad; es decir, es un lenguaje creado por y para los desarrolladores. Y ésta realmente no es una frase hecha, sino, por el contrario, una realidad incontrastable. Si navegan por **www.php.net/usage.php**, podrán apreciar que PHP sobrepasa los 19.720.000 dominios en uso. En esta nota analizaremos en profundidad todos sus pros y sus contras, el papel del desarrollador frente a esta plataforma, así como su proyección presente y futura.

En relación con su primacía, se ha dicho siempre que una de las grandes ventajas de PHP es la increíble cantidad de funciones, librerías, clases y aplicaciones desarrolladas y expuestas a la comunidad. Como contracara de la moneda, así como esta libertad de elección y de configuración atrae a muchos profesionales, para algunos otros resulta a veces incómoda, si buscan una plataforma que se instale con componentes preestablecidos. Si bien esto no ocurre en prácticamente ninguna tecnología que se precie de innovadora, también es cierto que PHP depende, en gran medida, de librerías y de módulos adicionales, que amplían su potencial en forma determinante. De todos modos, hay dos puntos que cabe destacar: dichas instalaciones, por lo general, son sumamente sencillas, estables y adaptables; además, la tendencia hace que, en las últimas versiones, ya muchas extensiones sean absorbidas en forma nativa.

Existen muchos aspectos que hacen que PHP sea la elección más amplia en el ámbito Web. En resumidas cuentas, podemos mencionar los siguientes:

→ **Multiplataforma**: se pueden desarrollar aplicaciones en Linux, Windows, MAC, Solaris, etc. Su portabilidad y su disponibilidad son amplias, y éstas son dos de sus armas más excluyentes. Tiene soporte para diversos servidores Web y puede correr tanto en IIS como en Apache.

→ **Sintaxis**: se utiliza una sintaxis clara y bien definida, muy similar a la de C.

→ **Flexibilidad y facilidad en el proceso de aprendizaje**: en este sentido, es clara su supremacía con respecto a otros lenguajes y plataformas competidores. Además, los resultados son rápidos y efectivos; incluso, es posible reutilizar mucho código expuesto libremente, lo cual lo vuelve más práctico a la hora de buscar soluciones.

→ Excelente soporte nativo para todo tipo de bases de datos: habitualmente se lo combina con MySQL, pero cuenta con soporte para muchas otras bases. Es así que vemos funciones para MySQL, MSSQL, Oracle, Postgre, SYBASE, ADABAS, SQLite, Informix, DBase y también ODBC, lo que le permite conectarse prácticamente con cualquier base de datos.

→ Es muy seguro, y cuando se descubren bugs, inmediatamente la comunidad encuentra soluciones y mejoras.

→ Es abierto y gratuito: el hecho de dejar este punto en último lugar no desmerece en nada su importancia y apuntala el éxito del lenguaje en todo el mundo. Hoy por hoy, cualquier desarrollador o empresa, siguiendo y respetando el régimen de licencias Open Source, puede modificar y utilizar componentes sin arriesgar ni perder en costos.

Como sabrán, y a diferencia de un lenguaje compilado, PHP es interpretado por el servidor que envía el resultado al navegador. Este es un argumento que suele esgrimirse como desventaja en relación a otras plataformas. Y si bien no podemos negar cierta razón en la definición, a medida que avancemos, veremos que PHP permite relativizar en gran medida ese concepto teórico.

Asimismo, y hace algún tiempo atrás, cuando se le buscaban defectos visibles al lenguaje, se resaltaba su incipiente pero no definitiva orientación a objetos, la cual nació en PHP 3 y continuó, sin grandes cambios, en la versión siguiente.

En ese momento, los objetos eran vistos como grandes colecciones de datos y métodos, que se agrupaban en clases que soportaban herencia simple. Recordemos que los objetos se pasaban como parámetros por valor y no por referencia, y los métodos y atributos sólo contaban con visibilidad pública y carecían de un manejo privado o protegido.

Hoy, estos puntos en contra ya no son tan sustentables. Actualmente, PHP se basa en la POO y el nuevo soporte de objetos, el cual ha sido completamente rescrito, con lo cual permite obtener un mejor rendimiento y nuevas características. Es cierto que PHP no nació como un lenguaje puramente POO, pero se han establecido nuevas pautas que lo hacen mucho más sólido en este sentido.

En este tema, podemos destacar las siguientes características:

* Los parámetros son pasados por referencia en los métodos y funciones. Existen restricciones en el tipo de los parámetros recibidos en los métodos.
* Visibilidad en los métodos y atributos (pública, protegida y privada).
* Uso de interfaces que soportan herencia múltiple.
* Clases y métodos abstractos.
* Creación de métodos y variables estáticas.
* Soporte para el manejo de excepciones.
* La escritura se hace más sencilla y legible; es del tipo: $objeto->metodo()->metodo2(). Se incorporan nombres por defecto para el constructor y el destructor, nuevas funciones y métodos (como __autoload-,__get, __set, __call).

Este último punto se enlaza con otro supuesto factor de desventaja, que residía en la esfera del tratamiento y depuración de errores. En ese sentido, el lenguaje también incorpora nuevas funciones que permiten un soporte muy bien definido en el área del manejo de excepciones. Esto se realiza con las sentencias try, catch y throw, por lo cual ahora una excepción puede ser lanzada, intentada o capturada, y, además, se pueden crear excepciones como subclases de **Exception**.

Lo que viene

Podemos ir avistando nuevas características para la futura versión de PHP (la versión 5.1). Dos de los aspectos más interesantes que podemos destacar son la instalación y la definitiva integración oficial de la incipiente pero poderosa plataforma PDO (*PHP Data Objects Interface*), y una nueva Zend Virtual Machine, aunque este último punto una gran incógnita. Por su parte, PDO (www.php.net/pdo) es una nueva API de base de datos que brinda una interfaz uniforme y soporta muchas características avanzadas. Representa un esquema totalmente orientado a objetos de acceso a sistemas de bases de datos, y promete llegar más rápido a un núcleo de PHP mucho más mejorado.

Si bien aún no existen demasiados datos con respecto al nuevo motor Zend, es posible que éste juegue todas sus fichas a mejorar considerablemente su performance, ya que hace por lo menos dos años, Rasmus Lerdorf, el creador de PHP, trabaja en la idea de colocar a PHP sobre la égida de Pharrot, el increíblemente veloz motor de Perl 6. Este proyecto se juega también con la idea de poder integrar, bajo una sola máquina, virtual PHP, Python y Perl.

Además, Lerdorf está abocado a desarrollar una extensión que realice eficientemente un minucioso filtro de entradas y reemplace a la actual validación de entrada de datos, mediante HTML y JavaScript. De todos modos, esta definición, aún sin nombre, no estaría incorporada por defecto y podría regularse desde el archivo de configuración principal en el servidor.

Posiblemente, PHP también vaya en dirección de poder integrarse con AJAX (*Asynchronous JavaScript and XML*). Esta tecnología, también llamada "JavaScript remoto", está llamada a cambiar gran parte del típico paradigma Web mediante la integración de XML, HTML dinámico y JavaScript como Web services, y ya es usada por Gmail. En esta área, se destaca un trabajo similar, desarrollado por el proyecto JPSPAN (jpspan.sourceforge.net), gracias al cual podemos evaluar algo más cercano a esta prometedora tecnología.

Asimismo, es destacable ver cómo maduran algunas ideas relacionadas con la posibilidad de crear aplicaciones que permiten compilar el código PHP a ejecutables.

En ese sentido, hoy en día podemos encontrar ejemplos como PriadoBlender (www.priadoblender.com), una herramienta que va por la versión 0.3 y que permite distribuir PHP-GTK, así como también integrar en un solo archivo los fuentes y el ejecutable. Por su parte, es recomendable seguir el desarrollo de RoadSend Compiler (www.roadsend.com), que si bien es un producto comercial, incluye un IDE que realiza tareas de compilación a código nativo y cuenta con muy buena performance.

De todos modos, sobre algunos aspectos no se puede conjeturar demasiado, hasta tanto no se propaguen las conclusiones a las que se lleguen en las disertaciones de Québec, donde se presentarán muchas nuevas características que definirán el futuro inmediato de PHP.

Utilizando la función **set_exception_handler**, podemos crear una función personalizada de respuesta, para situaciones en las que una excepción llegue al nivel principal de ejecución sin haber sido capturada. También se incluye el excelente modulo **Tidy**, altamente provechoso para el diagnóstico, la limpieza y la corrección de defectos en documentos HTML, XHTML y XML.

En PHP, el crecimiento es muy sostenido y no parece tener un techo definido, considerando que nació, más que nada, como un lenguaje de script potente pero sencillo, y hoy está escapando a esa definición relativa. Es habitual ver cómo surgen nuevos proyectos y, en ese sentido, queda aún en el tintero ver de qué manera aumentan las posibilidades en el terreno de las aplicaciones de escritorio de la mano de PHP-GTK (ver la nota de Achitecture, de **.code #08**). Esto vendrá a suplir la desventaja que representa no ser una plataforma totalmente orientada a la creación de aplicaciones de este tipo, como sí ocurre en Java o en .NET.

Sólo nos resta esperar, entonces, las nuevas sorpresas que nos traerá el último Congreso de PHP en Quebec, Canadá. Pero antes, podemos ir espiando algunas posibles mejoras.

DESTRUYENDO MITOS: PRODUCTIVIDAD Y ALCANCES

En algunos sectores se dice con cierta frecuencia que PHP no alcanza para aplicaciones de gran tamaño o bien que es un lenguaje menor. De ahí se pasa al argumento de que no es un framework sino un lenguaje, que al ser interpretado, pierde en calidad y rendimiento. Sean estos comentarios bien o malintencionados, lo cierto es que no representan la verdad ni mucho menos. Por eso, no es aconsejable dejarse llevar por opiniones que varían según se modifica una plataforma de negocios. Según sea el caso, todas las tecnologías tienen su lado fuerte y su lado débil. Cada opinión es respetable, y es recomendable verificar cada una de ellas personalmente y elegir la más conveniente en ese momento y para cada problema.

Además, PHP tiene un área de aplicación y uso en la cual es líder, y que lo abstrae en gran parte de la actual batalla de plataformas compiladas, como J2SE-.NET.

De todos modos, y en aquellos aspectos en los cuales las plataformas sí son comparables, es posible analizar si muchas de estas preguntas son ciertas o no. ¿Se codifica más en PHP que en el ambiente Web de una plataforma como ASP.NET o JSP? ¿Es realmente tan así? ¿Se carece de un buen framework? Veamos algunas posibles respuestas.

En el caso del aprendizaje y del proceso de codificación de PHP, es posible que la productividad de un programador novato aumente si viene de C, JSP/J-2SE o Perl, más que en el caso de un programador de Delphi. Pero esto sólo se ve de forma poco ostensible en un principio, ya que este lenguaje es bastante reconocido por la reducida curva de aprendizaje que tiene. En ese sentido, sólo hace falta mirar la gran cantidad de desarrolladores que migran desde ASP clásico a PHP, y la adaptación no suele ser nada traumática.

En relación con una mayor codificación, el mito es muy poco sostenible. PHP cuenta con una excelente propuesta en lo que se refiere a sintaxis y, además, utiliza muchos de los estándares empleados en otros lenguajes, como la separación de diseño y código (ver los motores de plantillas), la compilación condicional, variables estáticas, notificación estricta de errores y una orientación a objetos más rigurosa.
En relación a los frameworks, es muy natural que se piense de esa manera si se desconocen las bondades de Pear o de algún otro entorno tan prometedor como WACT. Es más, es muy posible que, si una persona está acostumbrada a un solo ambiente de trabajo, ignore con total razón que pueden coexistir diversos ámbitos de trabajo eficaces y paralelos para un mismo lenguaje.
En ese sentido, hoy, la mayoría de las miradas se dirigen, por cantidad, calidad, solidez e historia, a **Pear** (pear.php.net), que es el framework oficial de PHP. Este representa una extensa colección de más de 270 librerías que funciona como un repositorio indispensable destinado a unificar extensiones y aplicaciones escritas tanto en PHP como en C/C++. En virtud de esto, este entorno se divide en Pear propiamente dicho y PECL. En el último tiempo, el repositorio creció de tal modo que, a partir de la versión 4.3 de PHP, el paquete principal de Pear fue integrado de forma nativa.
Por eso mismo, sus posibilidades crecen día a día y son verdaderamente atractivas, incluyendo procesos de abstracción de conexión a datos, manejo avanzado de autentificaciones, generación de archivos y documentación, manipulación de complejas ecuaciones matemáticas, etc. (todo con pocas líneas de código). También se dice que PHP no es apto para proyectos de mediana y gran envergadura.
Para terminar, dejaremos un par de inquietudes en virtud de lo que consideramos puede ser eje de otro mito bastante polémico: el soporte. Si bien mucho de esto está relacionado con la discusión en torno al software libre, vale la pena preguntarse cuál es el valor real que tiene un supuesto soporte, merced a una licencia paga. Quienes conocen sobre este tema pueden ratificar la relatividad de ese punto. Vale decir, ¿cuán eficaz puede ser la ayuda técnica dada en ciertas circunstancias límites? ¿Es comparable ese soporte con el de una plataforma como PHP, que cuenta con infinidad de documentación y foros en línea donde expertos de todo el mundo solucionan consultas al momento? En ese punto subjetivo, no hay una única definición o verdad absoluta, pero podemos afirmar que una empresa, al elegir PHP o un desarrollo abierto, no está precisamente sola a la hora de encontrar soluciones.

INTEROPERABILIDAD

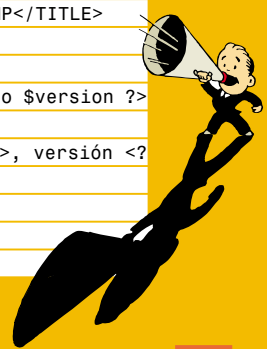
Establecer comparativas puede transformar un análisis en algo quizá tedioso, o bien poco objetivo y neutral. Por eso nos concentraremos en una opción mucho más interesante y útil. Esto es, básicamente, poder verificar cómo y en qué medida PHP se conecta y relaciona cada vez más con otros ámbitos. En este sentido, PHP seguramente cuenta con una de las ventajas competitivas más amplias. Además de todo el soporte para XML (ver el Special Report de **.code #12**) y Web Services, PHP se hace completamente interoperable con otros lenguajes y protocolos, incluyendo C/C++, Java, Perl, COM/.NET, LDAP, etc.

Con Java

La integración de Java y PHP puede otorgar grandes ventajas y satisfacciones a este último. De hecho, algunos desarrolladores vaticinan que la razón por la cual muchos de los cambios sintácticos parecen basarse en Java es porque, en un futuro, hay serias posibilidades de integrase a esa plataforma. Más aún, si vemos la reducida aceptación general que tuvo JSP, a pesar de ser un lenguaje muy aceptable. Como desarrolladores Java o PHP, existen dos formas claramente inversas de integrar ambas tecnologías.
La primera, y tal vez la más usada y recomendada, es integrar PHP dentro de un entorno **Java Servlet**, usando el **Java Service Access Point Identifier** (SAPI), el cual se emplea como interfaz del servidor de servlets. Esta solución es reconocida como la más eficiente y estable, y permite que el procesador PHP corra como un servlet.
La segunda opción es combinar Java dentro de PHP, lo cual se realiza mediante la extensión nativa de Java. Una vez habilitada dicha extensión, podemos crear y llamar objetos y métodos de Java desde PHP. Un ejemplo muy típico de esta librería **dll** es el que muestra la plataforma y versión donde está corriendo el entorno Java, basándose en una instancia de un objeto inicial de la extensión Java en PHP. Como veremos a continuación, a partir de la instancia $sistema, y a través de sus métodos y propiedades, podremos ir guardando en variables datos tales como la versión de Java, el desarrollador, y el nombre y versión del sistema operativo, para luego mostrarlos por pantalla.

```
<?
//Instanciación objeto principal
$sistema = new Java('java.lang.System');
//datos de entorno y SO
$version = $sistema->getProperty('java.version');
$devJVM = $sistema->getProperty('java.vendor');
$osnombre = $system->getProperty('os.name');
$osversion = $system->getProperty('os.version');

?>
<HTML>
<HEAD>
<TITLE>Prueba de integración entre Java y PHP</TITLE>
</HEAD>
<BODY>
<H1> Se esta corriendo la versión Java <? echo $version ?>
 del desarrollador <? echo $devJVM ?><br>
 en un sistema operativo <? echo $osnombre ?>, versión <?
  echo $osversion ?>
 en forma satisfactoria. </H1>
</BODY>
</HTML>
```



> Podemos afirmar que una empresa, al elegir PHP, no está precisamente sola a la hora de encontrar soluciones

La instanciación del objeto tiene las siguientes características:

```
new Java(clase, parametros)
```

El primer parámetro es la clase que fue invocada, y el segundo, los argumentos pasados al constructor del objeto. Si no se le pasa ningún parámetro, y existe un adecuado constructor por defecto, se puede acceder a la mayoría de sus métodos, ya que éstos son estáticos.
A continuación, veremos otro caso que demostrará cuán sencillo puede ser interactuar debidamente entre plataformas. Por ejemplo, si desde Java creamos una pequeña clase emitiendo un mensaje:

```
public class MensajeJava {
public MensajeJava() {}
public String EmitirMensaje (String mensaje) {
return "El mensaje es"+ mensaje;
}
}
```

Una vez compilada esta clase, podríamos verla desde PHP de esta forma:

```
<?
$objeto_java = new Java ("MensajeJava");
$mensaje_java = $objeto_java->EmitirMensaje
("La conexión JAVA-PHP se realizo con exito");
echo $mensaje_java;
?>
```

Es importante destacar, de todos modos, que la extensión está aún en fase experimental, por lo que es muy recomendable evaluar esta alternativa o bien la primera, hoy por hoy, más sólida y más estable. Además, la configuración de dicha extensión puede ser muy sencilla en Windows, pero se complica un poco más su puesta a punto en Linux. Por eso, un grupo de personas desarrolló un proyecto en paralelo que es conocido como "Php/Java Bridge" (sourceforge.net/projects/php-java-bridge). Este se comunica con la máquina virtual a través de sockets locales, usando un protocolo de comunicación determinado. La idea es muy amplia e interesante. Esperamos que esta extensión pueda ir creciendo rápidamente, ya que apunta también a cubrir y establecer nexos con nuestra siguiente tecnología relacionada: .NET.

Con .NET

En el último tiempo, y gracias a los denominados objetos COM, se torna sencillo utilizar componentes surgidos de la plataforma .NET, escritos en C# o VB.NET desde PHP. Imaginemos, por ejemplo, que creamos en .NET una aplicación estándar que maneja datos de empleados de una empresa. Podríamos invocar eso desde un archivo PHP de esta manera:

```
'Instanciamos la clase creada en Net
$Empleado = new COM ("clasephp.aplicacion.
empleado");
Cargamos sus propiedades
$Empleado->nombre = "Juan";
$Empleado->apellido = "Perez";
Mostramos por pantalla
echo "Nombre : $Empleado->nombre";
echo "<BR>Apellido : $Empleado->apellido";
```

Actualmente, podemos encontrar también un incipiente pero interesante proyecto de un grupo de estudiantes checo, que ofrece un novedoso compilador PHP para .NET. Esta compleja solución es conocida como **Phalanger** (www.php-compiler.net) y viene a suplir (si bien no se basa en éste) el vacío generado por el ya abandonado proyecto PHP Sharp. Esta librería permite utilizar PHP como un lenguaje nativo en .NET, es decir que con ella podremos compilar código PHP a MSIL (el cual será ejecutado por la .NET Common Language Runtime). Pero PHPlanger no sólo es un compilador, sino que también ofrece un funcional esquema de integración con Visual Studio, para así poder editar y depurar distintos programas dentro de este completo entorno de desarrollo. Con Phalanger podremos combinar y usar clases PHP desde un programa desarrollado con .NET, además de utilizar clases .NET desde PHP.
De esta forma, podemos conciliar en gran medida lo mejor de ambos mundos. Esta aplicación, además, es compatible sólo con la última versión de PHP y posee una buena documentación de ayuda.
Si bien Php1: mono está mucho más inmaduro y en fase experimental, ofrece similares prestaciones y, a diferencia, es totalmente Open Source (**www.php.net/~sterling/mono/**).

SEGURIDAD

Si algo se destaca de la filosofía Open Source es que miles de desarrolladores actúan como un elemento sostenedor y depurador de un patrón de calidad constante. Muchas veces se ha tratado de atacar a PHP por el lado de la inseguridad, sin recordar que muchos agujeros son descubiertos testeando las aplicaciones atacadas, y se los soluciona en tiempo récord. Por eso, dichos aspectos son ampliamente custodiados por toda la comunidad, en base a plataformas sólidas, con verificaciones diarias y procesos de depuración actualizados.
La plataforma cuenta con un grupo de personas especialmente dedicadas al tema, agrupadas en el denominado **PHP Security Consortium** (phpsec.org). Este organismo dedica muchos esfuerzos a acelerar, publicar y mejorar todas las medidas de seguridad existentes. Sobre la base de sus postulados, es posible determinar con precisión muchos aspectos que se deben tener en cuenta en este tema.
PHP puede operar en modo seguro (*safe-mode*), lo cual restringe el acceso de archivos al sistema y, así, nos permite deshabilitar funciones que puedan presentar riesgos. En cuanto al uso de memoria en PHP, esto se puede determinar y configurar con antelación.
PHP siempre intenta corregir y reemplazar defectos o vicios de versiones antiguas. Es así que, por ejemplo, en la última versión aparecen deshabilitadas por defecto las variables globales. Si bien este cambio generó algún resquemor por algún tipo de incompatibilidad entre programas escritos en PHP4 y PHP5, el cambio proporcionó un mejor uso y control. Recordemos que una gran cantidad de scripts hechos en PHP se basaban en la conversión automática de nombres de campos HTML a variables globales de PHP, y provocaban el volumen más elevado de fallas dentro de la plataforma.
Pasando al área de base de datos, es importante prestar debida y especial atención a contrarrestar técnicas potencialmente nefastas, como la de **Inyección Directa de Comandos SQL**. Como sabrán, ésta es una técnica en la cual un posible atacante crea o altera comandos SQL existentes, exponiendo o sobrescribiendo datos escondidos. Si

bien esto nace del conocimiento de la estructura de datos por parte del atacante, es también cierto que esto es perfectamente común y virtualmente posible. En este sentido, ningún detalle es paranoico, y deben tenerse en cuenta desde aspectos de creación restringida de privilegios, hasta control de campos que solemos no controlar. Como veremos más adelante, también es recomendable prestar atención a los formateos numéricos y de caracteres.
Si bien PHP no puede proteger una base de datos por sí solo, es posible fijar pautas y consejos que nos permitan asegurarnos de que, al acceder y manipular bases de datos desde nuestros scripts, el proceso se desarrolla en forma conveniente.
Es fundamental poder proteger no sólo los datos de tipo cadena, sino también los de tipo numérico. Por ejemplo, es recomendable utilizar apóstrofos en ambos lados de un valor numérico:

```
SELECT * FROM tabla WHERE indice=01
```

usaríamos la forma:

```
SELECT * FROM tabla WHERE indice='01'
```

→ MySQL convierte automáticamente esta cadena en un número, al eliminar todos los elementos no numéricos.
Es muy importante poder probar la introducción de caracteres, espacios y otros símbolos especiales en los campos numéricos de nuestros formularios, y actuar en consecuencia ante alguna reacción negativa de la aplicación.
→ Utilizar todas las herramientas de control de texto que PHP nos proporciona.
Esta es una gran ventaja que PHP pone a nuestro alcance y que lo distingue de cualquier otra plataforma. En este sentido, al tratar con cadenas y bases de datos, se pierde gran cantidad de tiempo en desarrollo y se provocan dolores de cabeza por desconocer muchas funciones de gran poder y potencial. Por ejemplo, podemos utilizar AddSlashes() para filtrar cualquier cadena de texto antes de que sea almacenada en una base de datos. Esta función se encarga de incluir el carácter correspondiente en nuestra cadena de texto, delante de cada símbolo problemático.
En forma inversa, **StripSlashes()** devuelve la cadena de texto a su estado original y, en este caso, la usaríamos para recuperar dichos nombres de la base antes de mostrarlos en pantalla.
→ Proporcionar una salida adecuada, en caso de que sean modificados los tipos de datos en los enlaces dinámicos (tipo numérico a tipo carácter o viceversa).
→ Comprobar fehacientemente el tamaño de los datos antes de que éstos lleguen a la base MySQL.
En este sentido, y relacionado al primer consejo, una saludable modalidad de testeo es introducir comillas, (') y ("), en los formularios pertenecientes a la aplicación por testear.
Pensemos, por último, que quizá no sea del todo agradable ver que nos devuelve un error, pero es mucho más interesante razonar que estamos realizando un trabajo verdaderamente profesional, tapando eventuales fallas graves de instrucción, si las detectamos y arreglamos en tiempo y forma.

> PHP es reconocido por su gran rendimiento, que lo eleva y lo hace competitivo frente a otras tecnologías

RENDIMIENTO

Si por algo es reconocido PHP, es por su gran rendimiento. Este rasgo lo eleva y lo hace realmente competitivo frente a otras tecnologías. De todos modos, es muy importante prestar debida atención a este punto en aplicaciones o sitios web de alto tráfico. Un buen rendimiento va de la mano de variados factores que lo nutren o alteran, según sea el caso. Por eso es fundamental estar atentos a dichos aspectos en forma adecuada. Enumerando los principales, podemos mencionar aspectos tales como la extensibilidad en el uso de estándares que permitan una correcta y fluida comunicación con otros sistemas, la posibilidad de agregar nuevos servidores ante la necesidad de mayores requerimientos (lo cual se traduce en una mayor y mejor escalabilidad), la facilidad de mantenimiento y desarrollo, la calidad y capacidad brindada en dicho proceso de creación, la capacidad de soportar fallas en servidores de una forma totalmente transparente y permitiendo una adecuada disponibilidad, como así también la confiabilidad y seguridad general.
Por otro lado, yendo a un simple proceso dinámico, el solo hecho de procesar una página de este tipo genera cierto trabajo extra al servidor, en relación a lo que sería una página estática común. Por eso, la idea general es prestar atención a la máxima optimización posible en el proceso, que consiste en la devolución del resultado procesado por el motor PHP.
Mas allá de los usuales métodos de optimización, por medio de herramientas que actúan a nivel de servidor, es también posible mejorar la performance a nivel de código. Una alternativa es utilizar, como caché o almacén temporario, archivos estáticos para mostrar el resultado de la ejecución de datos, que si bien son dinámicos, no se están modificando constantemente. **No obstante, hay que reconocer que este método representa un peligro en sí mismo. Si esa condición no se cumple, no configuramos debidamente el proceso, asegurándonos la fijación de tiempos de caducidad del archivo estático que se usa como caché.**
También es posible utilizar las funciones que guardan la salida generada en un buffer temporario. De esta manera, tomamos absoluto control de los contenidos enviados al navegador, y ésta es una opción muy interesante para manipular cookies y cabeceras.
Un truco práctico y sencillo, que aconsejamos adaptar a gusto de cada desarrollador, consiste en activar el modo buffering. Esto permite generar primero la página, enviándola de una sola vez, sin tener que hacerlo a medida que se genera. Para eso, usamos **ob_start** ("ob_gzhandler"). En ese caso, el parámetro permite que, antes de ser enviado, se comprima todo lo que tenemos en el buffer. Luego de incluir toda la lógica de la página, cerramos el buffer y enviamos todo con **ob_end_flush()**.
Pasando al tema servidores, podemos ratificar que existe un gran mito relacionado con el rendimiento, cuando se piensa que PHP funciona bien sólo con Apache.
También debe tenerse en cuenta que el legendario servidor es insuperable en cuanto a estabilidad, pero reduce algo su rendimiento en situaciones de alto tráfico.
Por este motivo, y aunque parezca extraño, es aconsejable utilizar la mayor cantidad de contenido estático posible, ya que el servidor lo entrega con más rapidez, y se optimiza el uso de memoria y procesador. También es recomendable separar el acceso a contenido estático en diversos equipos.
De todos modos, en toda esta tarea optimizadora, no estamos so-

los. Como decíamos antes, PHP cuenta con herramientas de optimización que, actuando sobre el servidor, otorgan mejores posibilidades en ese sentido. Una de ellas es Eaccelerator (eaccelerator.sourceforge.net), que sigue los pasos de Turck MMCache (**turck-mmcache.sourceforge.net**), una herramienta gratuita y Open Source que actúa como un excelente acelerador, optimizador, encriptador y manejador de caché. Esta aplicación rusa promete reducir la carga del servidor e incrementar la velocidad de scripts entre 1 y 10 veces más; por otra parte, ofrece un cuadro en el que le saca ventajas en varios productos similares, en aspectos tales como respuestas por segundo.
Otro proyecto interesante es el británico PHP Accelerator (**www.php-accelerator.co.uk**), de la empresa Ion cube. Tan importante es, que la misma Yahoo lo ha utilizado para mejorar su performance. En otro sector, también debemos citar a APC (Alternative PHP Cache), un optimizador de caching que está inmerso en el proyecto PECL, y en cuyo equipo de mantenimiento encontramos al mismísmo Rasmus Lerdorf.
Podremos probar la funcionalidad de APC desde pecl.php.net/package/APC.
Citaremos, por último, al sencillo pero poderoso JpCache (**www.jpcache.com**) y a AfterBunner (bwcache.bware.it/cache.htm), un proyecto sin demasiadas actualizaciones pero que vale la pena probar.

Entornos, librerías y clases de relevancia

Resulta francamente complejo elegir una selección objetiva de entornos, librerías y clases, sin dejar ninguno fuera, habiendo tantos y tan buenos. De todos modos, la propuesta nace de ir en orden, empezando en forma ascendente por los frameworks que conjugan muchas de estas geniales extensiones.
Comenzaremos por WACT (**wact.sourceforge.net**), que significa Web Application Component Toolkit. Se trata de un modelo de componentes que representa una nueva arquitectura de construcción de aplicaciones Web, sustentado en un diseño modular basado en patterns. En general, muchos vieron en WACT una respuesta clara a ASP.NET, pero consideramos que, sin pretender menoscabar a la plataforma de Gates, esto sería reducir la verdadera naturaleza del proyecto. Además, detrás de WACT hay un reconocido grupo de desarrolladores de extensa trayectoria en PHP, por lo cual es importante ver este entorno como uno de esos proyectos que marcan el rumbo de futuros cambios y mejoras. Con WACT se busca facilitar el uso de técnicas, como la refactorización o bien el diseño dirigido por pruebas en esta plataforma, así como simplificar el proceso de abstracción
En nuestra selección incluimos a Prado (www.xisc.com), un framework muy premiado y reconocido en el último tiempo, que implementa una API funcional e inteligente que crea un RAD con orientación a objetos. Este proyecto está basado en el **Apache Tapestry** de Java y, como vemos en su definición original, este ambiente de trabajo reconceptualiza todo el desarrollo de aplicaciones Web, dándole una nueva orientación en términos de componentes, eventos y propiedades. Para alguien que venga de .NET o Java, puede situárselo en un punto de comparación con ASP.NET y Java Server Fases, respectivamente.
Un componente Prado está formado por una eficaz combinación de tres archivos esenciales: una especificación xml, una plantilla html y una clase PHP. Prado promete cinco beneficios clave en el desarrollo de aplicaciones: reusabilidad, facilidad de uso, robustez, rendimiento basado en técnicas de caché, y separación de contenidos en dos capas (de presentación y lógica). Este framework trabaja con dos extensiones (SimpleXML y zlib) y facilita en gran medida módulos muy usuales, como los de autentificación.
En su galería de ejemplos, podemos verificar una buena cantidad de ejemplos que documentan su facilidad de uso. En ese sentido, podríamos ver, por ejemplo, cómo utiliza el componente **TDataGrid** en dos archivos tpl y PHP, para el rápido armado de una grilla de datos. En el archivo **tpl** especificaríamos las características de la capa de presentación de dicha grilla; por ejemplo, sus tamaños y colores. Por su parte, el archivo PHP (capa lógica) se encarga de la instalación y la conexión de datos.
Tampoco debemos olvidarnos de diversos frameworks que están basados en el dise-

Zend y otros casos

Otra empresa esencial que se ha centrado en ese aspecto es Zend (creadores del núcleo de PHP).
Para empezar a describirla, diremos que Zend Optimizer es una de las muestras más fehacientes del interés de Zend por mejorar la productividad. En efecto, esta aplicación gratuita permite ejecutar archivos codificados (mediante Zend Encoder) que mejoran notablemente la velocidad de ejecución de nuestras aplicaciones.
Para lograrlo, usa las llamadas optimizaciones multi-pase. Este incremento reduce las cargas excesivas en el procesador, así como el tiempo de ejecución, en una estimación de entre un 20% y un 50%. Por otro lado, tenemos a Zend Accelerator, un aplicativo con costo que optimiza el código más crítico y aplica un conjunto de técnicas y optimizaciones. Gracias a su uso, se acorta la latencia a niveles muy bajos, ya que incrementa hasta tres veces el número de peticiones que pueden manejarse. Con él podremos monitorear el rendimiento general y el incremento de velocidad del sitio.
Por su parte, Zend Encoder es otra aplicación paga que permite proteger la propiedad intelectual de un script antes de distribuirlo, con lo cual evita plagios o copias ilegales de nuestro código, que vayan en desmedro de nuestro trabajo y esfuerzo. En este área, es muy recomendable seguir el desarrollo del proyecto BinaryPHP, que se presenta como una herramienta que puede convertir un script PHP en código fuente C++, que podría ser compilado con un compilador C++ estándar.
Siguiendo con Zend, y hacia un horizonte más orientado a empresas y administradoras de hosting, dicha firma también presenta una serie de elementos de gran valor. En ese sentido, PHP Intelligence se integra a Zend Studio y suministra diversos informes, con detalle de eventuales errores de funcionamiento relacionados con el rendimiento, bases de datos y código. Se monitorean en forma activa todas las aplicaciones PHP en funcionamiento, y se crean marcas de control de eventos, con información necesaria para la toma de decisiones concretas.
Tambien podemos citar a PHP/Java Interoperability, aplicativo que se encarga de aliviar y complementar el espacio de acción entre PHP y Java. Completan la suite PHP Performance Management y PHP Configuration Control, la cual optimiza la sincronización y la garantía de consistencia de la configuración de los scripts PHP en múltiples servidores.

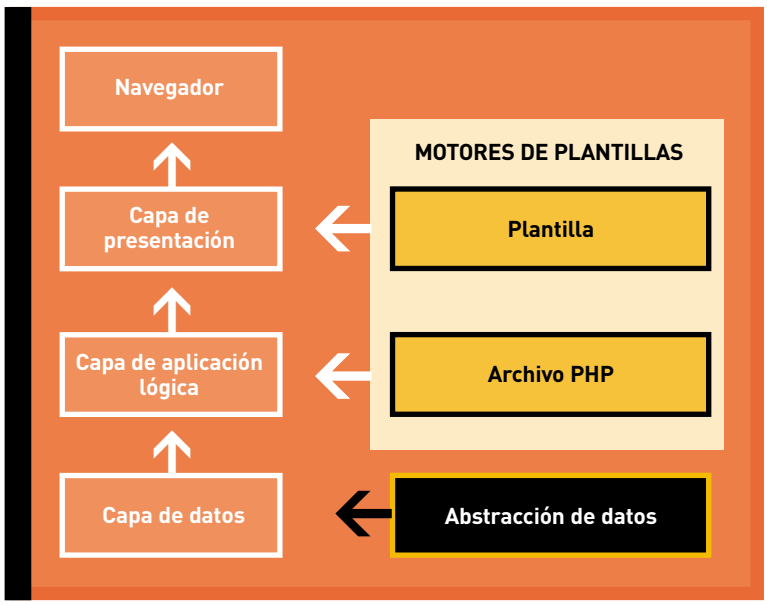
ño de patrón **MVC** (*Model-View-Controller*), como **Mojavi** (www.mojavi.org), **Binarycloud** (www.binarycloud.com), **phpMVC** (www.phpmvc-.net), **Horde** (www.horde.org/horde), **Phrame** (phrame.sourceforge-.net) o **Studs** (mojavelinux.com/projects/studs), este último emparentado en gran medida con el ámbito Java.

Como recordarán, MVC se sustenta en el diseño de aplicaciones basadas en el paradigma Modelo 2. Este modelo de diseño permite que la capa de presentación o vista Web (View) aparezca separada del código o núcleo de la aplicación (Controller/Model), lo cual es esencial en la construcción de grandes desarrollos con características POO.

En cuanto a preeminencia, si bien phpMVC y Mojavi parecen estar algo más extendidos que sus competidores, todos tienen características propias que vuelcan la balanza de preferencias hacia uno u otro lado en determinados aspectos puntuales.

Por su parte, **phpCodeGenie** o, simplemente, **PCG** (phpcodegenie-.sourceforge.net) es un generador de código para aplicaciones que manejan bases de datos, y se posiciona como uno de los productos más exitosos del ambiente PHP. La razón de esto, posiblemente, sea la facilidad con la que se generan **ABMs** y todo tipo de soluciones de manejo de datos, desde distintos ámbitos y con un mínimo esfuerzo, suponiendo un ahorro de tiempo muy considerable que puede dedicarse a realizar tareas de mayor complejidad. Asimismo, incorpora un original sistema de plugins que permiten, incluso, generar código para otras plataformas y lenguajes.

Siguiendo esta tendencia, y si deseamos uniformar la creación y el procesamiento de formularios desde el servidor, el paquete PEAR: **HTML_QuickForm** (pear.php.net/package/HTML_QuickForm) provee de productivos métodos de creación, validación y procesamiento rápido de más de 20 elementos para formularios html. También incluye soporte para upload, creación de reglas, manejo automático de validación y filtro del lado del servidor, soporte XHTML e integración con diversos motores de plantillas. Haciendo un breve repaso por las librerías más acotadas, en relación con su propósito, encontramos una variada oferta de clases y repositorios que ofrecen, en pocas líneas, la posibilidad de generar y manipular rápidamente documentos de suma utilidad para efectuar trabajos empresariales o de oficina. En ese sentido, contamos, por ejemplo, con un excelente soporte para archivos pdf o planillas de cálculo Excel.



Con la aplicación de plantillas y abstracción de datos, se mejora la productividad.

En el primer grupo, vemos clases con diversos grados de complejidad. Es así que, entre las más requeridas, encontramos tanto una librería como PHP Pdf Creation, que permite la generación simple de pdfs sin necesitar módulos externos; hasta una clase como TCPDF, que con soporte en la última versión de PHP, posee avanzadas características de creación de documentos "on the fly" y soporta UTF-8, Unicode, HTML y XHTML.

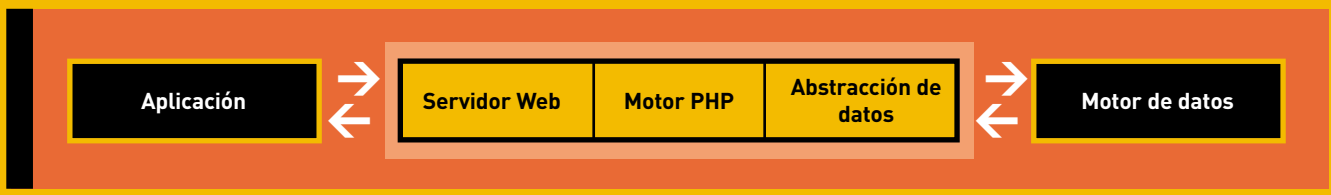
En el segundo sector, podemos mencionar librerías como Spreadsheet_Excel_Writer (de Pear), que permite validar y generar hojas de cálculo en forma rápida y segura. Con este completo paquete, podemos insertar fórmulas o incorporar imágenes, manejando distintas hojas de cálculo en un mismo archivo.

Si sólo requerimos una buena y exhaustiva lectura, recomendamos la librería PHP-ExcelReader, que permite cargar y manipular un documento de Excel con suma facilidad, posibilitando, incluso, su carga y procesamiento dentro de arrays.

De todos modos, y fijándonos un límite en nuestro criterio de elección, es muy importante resaltar que nuestra idea no ha sido repasar aplicaciones Web instalables, sino más bien, utilidades que faciliten el trabajo del desarrollador. Cabe destacar, por supuesto, que la información en el sector LAMP es tan grande y variada, que sería necesario realizar varias reseñas para poder hacer justicia a tanto material disponible.

ABSTRACCIÓN DE DATOS

Es conocido e indudable el beneficio que implica usar librerías de abstracción a datos, no sólo desde el punto de vista de una interfaz común de funciones para toda base de datos que usemos, sino también de otros tantos aspectos que a veces pasan inadvertidos. Esto abarca detalles muy importantes que suelen afectar de distinta forma el proyecto de calidad, tales como poder mantener un control centralizado de errores o unificar distintos criterios de trabajo entre grupos de desarrollo no tan homogéneos. En cuanto a opciones de este tipo, en PHP gratamente la situación se "complica", ya que tenemos varias para elegir. Por eso, iremos analizando una por una, con la mayor objetividad posible.



→ **PEAR:DB():** Esta popular API, escrita en PHP, proporciona una capa de consultas orientada totalmente a objetos y, sin dudas, se destaca sobre las otras en cuanto es incluida con todo el paquete Pear. Eso no menoscaba su eficiencia, ya que está bajo constante desarrollo y muchos programadores del núcleo de PHP forman parten de ese crecimiento. Como verán, en el ejemplo se utiliza un DSN (o nombre de fuente de datos que actúa como una URL hacia la base de datos). Así es que cada conexión se optimiza usando una cadena única con un formato especial, en la que algunos de los parámetros son opcionales. Esta extensión también soporta precomprobación y limitación de consultas SQL, así como conexiones persistentes y transacciones.

Ejemplo:

```
Include('DB.php');
$dsn = "$tipodebasededatos://$usuario:
$password@$host/$BaseDeDatos";
$db = DB::connect($dsn, true);
$db = DB::connect($dsn);
$consulta = "SELECT * FROM tabla";
resultado = $db->query($consulta);
while ($row = $resultado->fetchRow()) {
 $indice = $row[0];
}
```

→ **ADODB (adodb.sourceforge.net/):** Esta es la otra herramienta que, actualmente, gana en preferencias debido a su gran rendimiento y maduración. Permite consultar, actualizar e insertar registros por medio de una API portable, con gran facilidad. Además, maneja más bases de datos que **PearDB**, siendo totalmente orientada a objetos. Brinda un soporte adecuado para PHP 5 y, entre sus interesantes características, da la posibilidad de generar archivos cvs. Es realmente sencilla de aprender y usa muchas convenciones similares a ADO de Microsoft, por lo que tiene excelente acogida entre ex programadores de ASP.
Dispone de clases especiales muy prácticas en casos puntuales. Una de ellas es una clase de diccionario de datos llamada **datadict**, que facilita en gran medida una creación de tablas e índices portable. Utiliza un monitor de rendimiento de base de datos, y esto incluye una muy útil clase orientada al ámbito de la performance llamada **monitoring**, que puede usarse para ajustar por medio de enunciados SQL. Asimismo, permite un manejo profundo de sesiones con soporte para base de datos con la clase **session management**.

Ejemplo:

```
include('adodb.inc.php');
$DB = NewADOConnection('mysql');
$DB->Connect($host, $usuario, $password, $basedatos);
$consulta = "SELECT * FROM tabla";
$recordset = $DB->Execute($consulta);
while (!$recordset->EOF) {
 echo($recordset->fields);
 $recordset->MoveNext();
}
```

→ **PHPLib (phplib.sourceforge.net):** A diferencia de los anteriores, en este caso, la capa de abstracción de PHPLib forma parte de un framework que permite un manejo avanzado de objetos, mediante clases y subclases de todo tipo. En ese sentido, la abstracción de datos se restringe a su clase **DB_Sql**, pero además posee clases para el armado de formularios (**HTML Widgets**), manejo de sesiones (clase **Session** y **User**), autentificación (clase **Auth**) y permisos (clase **Perm**). Por si esto fuera poco, se extiende a dos clases más, que permiten manejar plantillas y carros de compras. PHPlib tuvo un gran apogeo durante PHP 3, ya que soportaba sesiones y mostraba el camino en el ámbito del concepto de templates.
Si bien esta capa no cuenta con tantas opciones de acceso a datos como vemos en **Pear:DB** o **AdoDb**, incluye las más requeridas, y es una opción para tener en cuenta, especialmente si consideramos el rico entorno de clases que la rodea.

Ejemplo:

```
include ('db_mysql.inc');
 $DB= new DB_Sql;
$DB->Host = "Host";
$DB->BaseDeDatos = "BaseDeDatos";
 $DB->Usuario = "Usuario";
$DB->Password = "Password";
$DB->query($query);
while($DB->next_record())
{
 echo $DB->f("Indice") . " " . $DB->f("Campo2") . "<br>";
}
```

Otras abstracciones

Existen diversas abstracciones, además de esas tres más conocidas. El ya nombrado WACT posee una de ellas, y también podemos citar a Metabase (**www.phpclasses.org/metabase**). Esta abstracción nació de la mano del portugués Manuel Lemos, a partir de ver a PEAR::DB algo lento, y considerando que esa extensión no proveía de una verdadera abstracción a la interacción con servidores de datos. Sea esto real o no, lo cierto es que, más bien, es materia de opinión y polémica. Lo seguro es que la librería ganó sus adeptos con el paso del tiempo y se posicionó como una opción valida y eficaz. Otra posibilidad también interesante es ezSQL (php-.justinvincent.com), una clase que se destaca por la rapidez y la facilidad de uso en la implementación. Complementando su funcionalidad, es importante destacar que puede trabajar en conjunto con Smarty y que en su página oficial podemos encontrar una clase adicional de paginación que se ensambla a ezSQL y complementa el volcado de datos por pantalla.

Smarty y los motores de plantillas

En este sentido, cabe resaltar la importancia que se le está dando a este tipo de proyectos, que, en definitiva (tal como lo hacen, por ejemplo, ASP.NET u otras tecnologías), separan la presentación o diseño web de la parte lógica. En ese sentido, se destaca el crecimiento acelerado que está teniendo **Smarty** (smarty.php.net) en cuanto a uso y demanda de trabajo. Este motor de templates fue

desarrollado hace algún tiempo por Andrei Zmetvski, autor de la reconocida extensión GTK. Esto incide enormemente como aval de desarrollo, y es un factor determinante en cuanto a su reconocimiento y divulgación masivos. Esa divulgación mayoritaria es rescatable porque representa un paso adelante, no sólo en el avance del desarrollo genérico de futuros proyectos con templates, sino también en una búsqueda de un mayor ordenamiento y calidad en la capas de presentación y código.
Si bien cuenta con una serie de características que lo acercan al usuario intermedio, sus detractores lamentan cierta complejidad en el uso de un lenguaje de código propio de Smarty que se aparta de PHP. De todos modos, su instalación es sencilla y no requiere de ninguna extensión en particular para funcionar. Smarty incluye también un eficiente sistema de caché para optimizar el catching parcial o total de páginas dinámicas, y posee entre sus ventajas más claras unas técnicas avanzadas de compilación y recopilación que reconocen cambios y aceleran el funcionamiento.
Pero más allá de todo aspecto técnico, con Smarty podemos separar la capa de presentación de la lógica, en forma sencilla, ordenada y óptima a nivel de desarrollo. ¿De qué manera? Por medio de plantillas o templates. Cada plantilla es un conjunto de elementos html entremezclados con llamadas a código Smarty, definidas entre llaves, el cual es guardado con la extensión tpl. Su formato es similar al de este simple ejemplo:

```
Testeando plantilla Smarty <br>
Revista: {$nombre} <br> Numero: {$numero}
```

Imaginen esta mejora si debieran adaptar un sitio web a su formato wap. El paso de html a wml sería sencillo, y no habría que tocar la definición lógica de directivas PHP, que incluye manejo de bases de datos, etc.
A nivel sintáctico, el motor funciona con un cuidado set de instrucciones propias que son invocadas desde un template para generar iteraciones, formatear textos, recorrer vectores, etc.
En la clase deberemos configurar y dejar asentado si usaremos el sistema de caché disponible y si queremos un sistema de depuración de errores para nuestros templates. Un objeto derivado de esa clase será el que usaremos para mostrar las plantillas.
Como mencionamos antes, luego es sólo cuestión de empezar a usar el objeto instanciado para asignar variables, procesar formularios, acceder a la base de datos, utilizar el sistema de sesiones y, por supuesto, mostrar las plantillas.
Así, si quisiéramos crear el archivo PHP que determinara el área lógica de nuestra plantilla de ejemplo, sólo deberíamos asignar dichas variables y mostrar la plantilla:

```
$numero = 14;
assign('nombre','Code');
assign('numero',$numero);
$obj_smarty->display('revista.tpl');
```

Smarty necesitará algunos directorios para almacenar archivos de configuración y de caché, así como todo el grupo de plantillas, tanto las matrices como las compiladas; de esta forma, podrá trabajar correctamente.

OTROS MOTORES DE PLANTILLAS

Es igualmente importante destacar que hubo sistemas de plantillas anteriores muy valorables, y lo que es aún más importante, existen hoy otros proyectos que quizá puedan llegar a superar en algunos aspectos a éste.

Esto se debe a que, actualmente, coexisten dos corrientes bien definidas en cuanto a motores de plantillas. Por un lado, aquellos que están basados en la compilación de PHP, en el cual se genera código en forma dinámica a partir de la plantilla básica. Esta opción tiene la fama de ser más rápida, ya que utiliza todos los recursos de PHP como sistema de plantillas. Pero es susceptible a recibir ataques externos, si se logra manipular de algún modo el código generado.
La otra vertiente es más segura pero más lenta, y está basada en expresiones regulares, lo que supone un mayor procesamiento interno de cadenas.
En ese sentido, vemos un buen ejemplo con dos proyectos que están incluidos en el framework Pear, los cuales están basados en el uso de expresiones regulares. Estamos hablando de **HTML_Template_IT** (pear.php.net/package/HTML_Template_IT) y **HTML_Template_Sigma** (pear.php.net/package/HTML_Template_Sigma); este último agrega otra modalidad en la técnica de compilación de templates. Otra buena posibilidad para evaluar dentro de Pear es el reconocido motor **Xipe** (pear.php.net/package/HTML_Template_Xipe), que utiliza buenos procedimientos de mejora de caché.
Por otro lado, muchos desarrolladores reniegan de Smarty, y resaltan, en cambio, motores como **Savant** (phpsavant.com), que funciona casi como su "contracara" más seria y eficaz ya desde su propia página web, en donde se autodefine como "la más simple, elegante y poderosa alternativa a Smarty". Esto es, básicamente, porque este proyecto, del experto desarrollador Paul M Jones, consigue iguales resultados o superiores, mejorando algunos postulados de Smarty o difiriendo de ellos. Se ejecuta sin depender de un lenguaje propio intermedio y no sigue la modalidad de compilar código (al menos por defecto). En ese sentido, incluso Savant permite desarrollar nuestra propias reglas de compilación, si por determinado motivo deseamos establecer esto en forma más personalizada o sólo queremos restringir de algún modo el área de acción de los diseñadores. También hay que resaltar que Savant tiene un excelente sistema de manejo de errores.
Fuera de estas importantes opciones, encontramos otros motores menos divulgados pero muy completos. Este es el caso de **Template Tamer** (www.templatetamer.org), que incluye un framework integrado y un IDE muy útil destinado a facilitar en gran medida la implementación de su motor de plantillas; o de **Zope Page Templates** (www.zope.org/Members/4am/ZPT), un motor que conviene seguir muy de cerca porque sobresale del resto, al estar basado en basado en espacios de nombres xml.
Y si bien ya tienen sus años, es justo nombrar tanto a **FastTemplates**, que, basándose en un modelo de Perl, marcó en algún momento el camino y sigue teniendo adeptos; como a la ya mencionada biblioteca de librerías **PHPLib**, que cuenta con un poderoso motor de plantillas y una comunidad propia.
Si seguimos analizando, la lista se tornaría indefinida: **Pear-Feliz**, **PowerTemplate**, **Virtual Template**, **Pear-Itx**, **SimpleTemplate**, **HTML_Template_Sigma** y **Xtemplate** cumplen en su medida con

> Con Smarty podemos separar la capa de presentación de la lógica, en forma sencilla, ordenada y óptima.

> Con Smarty notaremos los beneficios que brinda un estilo de codificación más práctico, cómodo y profesional.

muchos de los postulados de un buen motor de plantillas; la cantidad de opciones resulta por demás extensa.
De todos modos, acá no termina la recorrida. Por el contrario, es menester estar atentos y mantenernos actualizados, evaluando nuevos proyectos que siguen apareciendo y crecen a medida que los usamos, y que se ajustan a nuestras necesidades.
Y a no dudar de que, cuando encontremos o desarrollemos un motor, abstracción o framework que realmente nos convenza, notaremos rápidamente los beneficios que brinda un estilo de codificación más practico, cómodo y profesional.

SQLite

Según dicen, "lo bueno, si breve, dos veces bueno". Y sin dudas, Sqlite se presenta en el nuevo PHP confirmando ese antiguo refrán. Esta librería escrita en **C** funciona como un excelente motor para bases de datos SQL empotrable (no inicia servicios en equipos independientes de la aplicación referida), por lo que puede ser incluida e incorporada en nuestro código fuente (sí, aunque parezca increíble, como un simple include). De esta forma, cualquier aplicación que sea enlazada a SQLite accede a una base de datos con todo el poder del lenguaje SQL, sin tener que ejecutar un programa de RDBMS por separado.
De instalación sencilla y reducido tamaño, se la define como totalmente portable y rápida (incluso más veloz que MySQL o Postgress), además de que cuenta con un efectivo soporte de transacciones.
De todos modos, y al poseer mecanismos de bloqueo más bien básicos, SQLite aún no permite que múltiples usuarios accedan en forma paralela y en modo escritura a la base de datos. Por eso, es más recomendable para backends o entornos de administración acotados, que requieran de una gran rapidez en las consultas, trabajando sólo con un usuario operador encargado de realizar modificaciones.
Por otro lado, SQLite posee, por consola, una herramienta que actúa como interfaz de comunicación, pero actualmente ya poseemos herramientas de administración por Web como **PhpSQLiteAdmin** (phpsqliteadmin.sourceforge.net).

A continuación, veremos un ejemplo de código PHP embebido con una consulta a una base de datos SQLite:

```
<?php
// Configuramos la ruta donde tenemos el
archivo
$db = $_SERVER['DOCUMENT_ROOT']."/Base
_Datos_Ejemplo.db";
// Abrimos el archivo SQLite que contiene
la base de datos
$handle = sqlite_open($db) or die("Could not open database");
// Creamos la consulta
$query = "SELECT * FROM tabla";
// Ejecutamos la consulta y formateamos la salida de un
eventual error
$resultado = sqlite_query($handle, $query) or die("Error en
consulta:".sqlite_error_string(sqlite_last_error($handle)));
// Si existen registros
if (sqlite_num_rows($result) > 0) {
//Mostramos información por pantalla
 while($row = sqlite_fetch_array($resultado)) {
 echo."Primer Campo: ".$row[0]."<br>";
 echo."Segundo Campo: ".$row[1]."<br>";
 }
 }
//Cerramos la conexión
sqlite_close($handle);
?>
```

IDEs y programas complementarios

La empresa más importante en torno a PHP, **Zend**, provee de varias herramientas, gratuitas y pagas, que hemos visto en detalle en párrafos anteriores. Estas herramientas son sumamente útiles para la creación y optimización de código PHP.
Sin embargo, el corazón de esta suite es **Zend Studio**, el IDE por excelencia de PHP (ver la sección review en **.code #09**). Posee grandes facilidades en cuanto a gestión de proyectos y depuración de errores. De más está destacar su absoluta integración con PHP, así como las otras herramientas de mejoramiento que proporciona. Pero además de Zend tenemos otras opciones, tanto comerciales como Open Source, de gran valor. Una de ellas es **Maguma**, creada por Tobias Ratschiller, el mismo desarrollador que trabajó en el nacimiento del reconocido proyecto **PHPMyAdmin**. Fue así que el incipiente phpCoder pasó a ser Maguma Studio y se posicionó como un IDE muy difundido. El siguiente paso fue seguir en esa dirección, pero con prestaciones más avanzadas, que llevaron finalmente al actual **Maguma Workbench**, que además de su afinada gestión de depuración (DBG Debugger), es extensible para el desarrollador, puesto que su estructura es modular. Su nivel de integración permite la automatización de muchas tareas comunes de desarrollo, lo cual acelera el proceso de la integración de aplicaciones tales como el navegador **CVS** y **SFTP**.
Dev-PHP IDE es una herramienta Open Source muy interesante, que permite compilar PHP-GTK e incorpora un visor de compilación.
Otra grata sorpresa es el reciente **PHP Designer 2005** (www.mpsoftware.dk/phpdesigner.php), de interfaz impecable (con diversas skins), posibilidades de debuggeo y codificación. Otra puede ser PHPEdit (www.waterproof.fr/products/PHPEdit), que cuenta con la mayoría de las opciones requeridas en un desarrollo moderno y agrega dos herramientas acoplables, destinadas a la optimización de código y al formateo xml (phpCodeBeautifier y XMI2PHP).
De todos modos, la lista de opciones es larga y realmente muy buena. Incluye tanto editores más genéricos, como **Ultraedit**, como otros entornos indicados, como Nusphere o PHP Expert.

Por otro lado, y para aquellos que vienen de otras tecnologías, el traspaso será menos conflictivo recurriendo a extensiones que posibilitan utilizar los IDES más extendidos en esas plataformas. Así es que los desarrolladores de .NET podrán utilizar PHP con el novedoso editor VS.Php de la empresa JCX software (**www.jcxsoftware-.com**), el cual se integra a VisualStudio .NET 2002/3.
Para los desarrolladores orientados a Java, existe el editor Tru Studio (**www.xored.com/trustudio**) o bien el set de plugins PHP-Eclipse para ese framework, disponible **phpeclipse.de**.

EXPERTO

Si bien es imposible abarcar todo el universo PHP en un solo artículo, en el transcurso de estas páginas hemos visto muchos aspectos que hacen a la formación de un buen desarrollador de este lenguaje. Y aunque es bastante subjetivo considerar que un ajustado conjunto de parámetros determina el nivel de un desarrollador PHP, podemos sí destacar algunas pautas que nos ayudarán a alcanzar un alto nivel de calidad y de experiencia, y nos permitirán verificar, de modo más o menos claro, en qué escalón nos encontramos.
Por eso, es esencial considerar, con la mayor precisión y humildad posibles, qué puntos fuertes tenemos, para entonces profundizarlos; a la vez que debemos poner debida atención a falencias o distracciones que nos perjudican, y establecee un plan estricto y sistemático para subsanarlas. Y aunque el ejemplo sirva para el caso de una aplicación LAMP o una extensión PHP, vale la pena analizar una pequeña parábola que, adaptada a estos hechos, ilustra gráficamente el tema en función de poder reconocer un nivel de capacidad. Esta idea sostiene que un desarrollador novato PHP debería tanto poder reconocer correctamente qué aplicación debe instalar, como llevar dicho proceso de instalación a buen puerto. Por su parte, un buen programador intermedio debe poder superar esa instancia y ser perfectamente capaz de analizar una aplicación, con el fin de mejorarla y/o adaptarla en mayor o menor medida a sus necesidades específicas.
Para terminar el concepto, podemos pensar que un programador PHP experto evaluará metódicamente las mejores herramientas que tiene a su alcance, y escribirá una propia aplicación que busque combinar y canalizar sus mejores ideas.
Pero más allá de cualquier abstracción teórica, es posible determinar, a grandes rasgos, aspectos importantes que hacen al desarrollo de un programador experto en PHP. Esto parte, esencialmente, no sólo de conocer las herramientas disponibles que ofrece esta plataforma, sino también de poder tener un buen manejo de la programación orientada a objetos. Esto implica, además, estar lo suficientemente preparado para encarar correctamente el análisis, el armado, y la modificación de funciones y extensiones propias y ajenas.
En referencia al mercado, conviene conocer los frameworks y los motores de plantillas más difundidos, sean o no de nuestro agrado. También es aconsejable tener a mano herramientas como las que ya vimos, que mejoren nuestra productividad en el desarrollo de ABMs y formularios que puedan facilitarnos un trabajo más compacto y profesional. Vale decir, es muy importante haber conocido cómo se hace un ABM, pero si podemos aligerar esa carga de tiempo, centrándonos en otros aspectos, será mucho mejor aún. En este punto, es igualmente importante destacar que, si nos piden o elegimos una aplicación de terceros, es requisito indispensable conocer en detalle esa aplicación. Y si debemos elegir nosotros el repositorio LAMP, necesitaremos sopesar con precisión y anticipación dicha elección (ni mayor ni menor que los requerimientos del cliente), al igual que la mejor forma de adaptarla a nuestra necesidad. De otro modo, es altamente preferible crear nuestro propio desarrollo y considerar que la idea original de acelerar los tiempos con esa herramienta puede volverse en nuestra contra.
De todos modos, y como podrán notar, en el espíritu de este informe especial, tratamos de dejar bien asentado que el universo PHP tiene oportunidades, profesionales y posibilidades laborales para todos, pero guarda un lugar muy especial para aquellos inquietos desarrolladores que buscan trascender con originalidad ese espacio pleno de posibilidades e ideas.

	NOVATO / JUNIOR	INTERMEDIO	SENIOR / EXPERTO
DATOS	Conexiones básicas nativas a bases de datos.	Manejo de al menos un esquema de abstracción de datos.	Creación de esquemas de abstracción de datos propio.
LOGICO	Manejo y creación básica de funciones nativas y propias. Instalación de librerías externas.	Se verifica buen manejo de POO. Se crean librerías personalizadas. Se utilizan los framweworks y librerías más avanzados.	Creación o dirección de proyectos de frameworks, extensiones y librerías propias. Utilización avanzada de patrones.
OTROS	No se separa la vista del esquema lógico. Instalación de aplicaciones LAMP.	Utilización de motores de plantillas. Modificación de utilerías. LAMP. Servicios web e e-comerce. 3 años	Creación de aplicaciones LAMP y motor de plantillas personalizado. Manejo de equipos de desarrollo.
EXPERIENCIA	1 año		5 años

Los parámetros se desprenden del estudio de las certificaciones y los requerimientos laborales de pedidos de desarrolladores profesionales PHP. En general la escala más conflictiva surge de diferenciar un nivel intermedio, de uno experto. En este análisis se ha buscado consensuar de la forma más justa de este paso aunque es algo difuso, ya que no existen escalas de referencia en ese sentido y no siempre suele ser tenido en cuenta de la misma forma.