Cubre PHP 5.3

FUNDAMENTOS DE PHP

Prólogo de Carl Evans,
editor en jefe de Zend Developer Zone

Cubre:
Sintaxis • Estructura de datos • XML • Páginas externas • MySQL • Seguridad • Extensiones de terceros y más

Vikram Vaswani

McGraw Hill

BIBLIOTECA DEL PROGRAMADOR

Fundamentos de PHP

Fundamentos de PHP

Vikram Vaswani

Traducción

Luis Antonio Magaña Pineda

Traductor profesional

McGraw Hill

MÉXICO • BOGOTÁ • BUENOS AIRES • CARACAS • GUATEMALA • MADRID • NUEVA YORK
SAN JUAN • SANTIAGO • SÃO PAULO • AUCKLAND • LONDRES • MILÁN • MONTREAL
NUEVA DELHI • SAN FRANCISCO • SINGAPUR • ST. LOUIS • SIDNEY • TORONTO

Director editorial: Fernando Castellanos Rodríguez
Editor: Miguel Ángel Luna Ponce
Supervisor de producción: Zeferino García García

FUNDAMENTOS DE PHP

Prohibida la reproducción total o parcial de esta obra,
por cualquier medio, sin la autorización escrita del editor.

Educación

DERECHOS RESERVADOS © 2010, respecto a la primera edición en español por
McGRAW-HILL/INTERAMERICANA EDITORES, S.A. DE C.V.
A Subsidiary of ***The McGraw-Hill*** *Companies, Inc.*
Corporativo Punta Santa Fe,
Prolongación Paseo de la Reforma 1015, Torre A
Piso 17, Colonia Desarrollo Santa Fe,
Delegación Álvaro Obregón
C.P. 01376, México, D.F.
Miembro de la Cámara Nacional de la Industria Editorial Mexicana, Reg. Núm. 736

ISBN: 978-970-10-7132-8

Translated from the 1st English edition of
PHP: A beginner's guide
By: Vikram Vaswani
Copyright © 2009 by The McGraw-Hill Companies. All rights reserved.
ISBN: 978-0-07-154901-1

1234567890 1234567890

Impreso en México *Printed in Mexico*

Para Gurgle y Tonka, mis dos bebés

Contenido breve

Parte I Entender las bases de PHP

1 Introducción a PHP 3

2 Utilizar variables y operadores 21

3 Controlar el flujo del programa 49

4 Trabajar con matrices 85

5 Usar funciones y clases 121

Parte II Trabajar con datos de otras fuentes

6 Trabajar con archivos y directorios 159

7 Trabajar con bases de datos y SQL 185

8 Trabajar con XML 249

9 Trabajar con cookies, sesiones y encabezados 293

Parte III Seguridad y solución de problemas

10 Manejo de errores 317

11 Seguridad con PHP 349

12 Extender PHP 377

Parte IV Apéndices

A Instalar y configurar los programas requeridos 391

B Respuestas a los autoexámenes 419

Índice 445

Contenido

SOBRE EL AUTOR xvii
PREFACIO xix
AGRADECIMIENTOS xxi
INTRODUCCIÓN xxiii

Parte I Entender las bases de PHP

1 Introducción a PHP 3
Historia 4
Características únicas 5
Conceptos básicos de desarrollo 7
Crear tu primer script PHP 10
Escribir y ejecutar el script 10
Entender el script 11
Manejar los errores del script 12
Prueba esto 1-1: Mezclar PHP con HTML 13
Caracteres de escape especiales 15
Aplicaciones de ejemplo 16
phpMyAdmin 17
phpBB 17
Gallery 17
PoMMo 17

Smarty 18
Squirrelmail 18
eZPublish 18
Mantis 18
Wordpress 18
Resumen 19

2 Utilizar variables y operadores 21
Almacenar datos en variables 22
Asignar valores a variables 23
Destruir variables 24
Inspeccionar el contenido de la variable 25
Comprender los tipos de datos de PHP 26
Establecer y verificar el tipo de datos de la variable 27
Usar constantes 29
Manipular variables con operadores 30
Realizar operaciones aritméticas 30
Unir cadenas de texto 31
Comparar variables 32
Realizar pruebas lógicas 33
Otros operadores útiles 34
Comprender la precedencia de los operadores 36
Prueba esto 2-1: Construir un convertidor dólar-euro 37
Manejar datos de entrada para formularios 39
Prueba esto 2-2: Construir un muestrario HTML interactivo de colores 42
Resumen 45

3 Controlar el flujo del programa 49
Escribir declaraciones condicionales sencillas 50
La declaración if 50
La declaración if-else 51
Prueba esto 3-1: Probar números pares y nones 53
Escribir declaraciones condicionales más complejas 54
La declaración if-elseif-else 55
La declaración switch-case 55
Prueba esto 3-2: Asignar niños exploradores a su tienda de campaña 57
Combinar declaraciones condicionales 58
Repetir acciones con bucles 59
El bucle while 60
El bucle do-while 60
El bucle for 61

Combinar bucles 62
Interrumpir y omitir bucles 63
Prueba esto 3-3: Construir una calculadora factorial 64
Trabajar con funciones de cadenas de texto y numéricas 66
Utilizar funciones de cadena de texto 66
Utilizar funciones numéricas 73
Prueba esto 3-4: Procesar un formulario para registro de miembros 77
Resumen 82

4 Trabajar con matrices 85
Almacenar datos en matrices 86
Asignar valores a matrices 87
Modificar valores de matrices 89
Recuperar el tamaño de la matriz 90
Matrices anidadas 91
Procesar matrices con bucles e iteradores 92
El bucle foreach 93
El iterador de matrices 94
Prueba esto 4-1: Promediar las calificaciones de un grupo 95
Utilizar matrices con formularios 97
Prueba esto 4-2: Seleccionar sabores de pizzas 97
Trabajar con funciones de matrices 100
Prueba esto 4-3: Verificar números primos 107
Trabajar con fechas y horas 110
Generar fechas y horas 111
Formar fechas y horas 112
Funciones de fecha y hora útiles 113
Validar una fecha 114
Convertir cadenas de caracteres en sellos cronológicos 114
Prueba esto 4-4: Construir una calculadora de edad 116
Resumen 118

5 Usar funciones y clases 121
Crear funciones definidas por el usuario 122
Crear e invocar funciones 123
Utilizar argumentos y valores de retorno 124
Establecer valores de argumentos por defecto 126
Utilizar listas dinámicas de argumentos 127
Comprender el ámbito de las variables 128
Utilizar funciones recursivas 129
Prueba esto 5-1: Calcular MCD y mcm 132

Crear clases 135
Introducción a clases y objetos 135
Definir y utilizar clases 136
Prueba esto 5-2: Cifrar y descifrar texto 139
Utilizar conceptos avanzados de programación orientada a objetos 143
Utilizar constructores y destructores 143
Extender clases 144
Ajustar la configuración de visibilidad 147
Prueba esto 5-3: Generar formularios para listas de selección 148
Resumen 154

Parte II Trabajar con datos de otras fuentes

6 Trabajar con archivos y directorios 159
Leer archivos 160
Leer archivos locales 160
Leer archivos remotos 161
Leer segmentos específicos de un archivo 162
Escribir archivos 163
Prueba esto 6-1: Leer y escribir archivos de configuración 165
Procesar directorios 169
Realizar otras operaciones de archivos y directorios 172
Prueba esto 6-2: Crear una galería de fotografías 180
Resumen 183

7 Trabajar con bases de datos y SQL 185
Introducción a bases de datos y SQL 186
Comprender las bases de datos, registros y llaves primarias 187
Comprender relaciones y llaves externas 188
Comprender las declaraciones SQL 189
Prueba esto 7-1: Crear y alimentar una base de datos 191
Crear la base de datos 192
Añadir tablas 192
Añadir registros 194
Utilizar la extensión MySQLi de PHP 201
Recuperar datos 201
Añadir y modificar datos 205
Manejo de errores 209
Prueba esto 7-2: Añadir empleados a una base de datos 209
Utilizar la extensión SQLite de PHP 216
Introducción a SQLite 216
Recuperar datos 220

Añadir y modificar datos 224
Manejo de errores 225
Prueba esto 7-3: Crear una lista personal de pendientes 225
Utilizar las extensiones PDO de PHP 234
Recuperar datos. 234
Añadir y modificar datos 237
Manejar errores 240
Prueba esto 7-4: Construir un formulario de inicio de sesión 241
Utilizar una base de datos MySQL. 241
Conmutar a una base de datos diferente 246
Resumen. 247

8 Trabajar con XML 249

Introducción a XML 250
Aspectos básicos de XML 250
Anatomía de un documento XML 251
XML bien formado y válido 253
Métodos de segmentación de XML 254
Tecnologías XML 254
Prueba esto 8-1: Crear un documento XML 256
Utilizar las extensiones SimpleXML de PHP 257
Trabajar con elementos 258
Trabajar con atributos 259
Prueba esto 8-2: Convertir XML a SQL 260
Alterar elementos y valores de atributos. 262
Añadir nuevos elementos y atributos 263
Crear nuevos documentos XML 264
Prueba esto 8-3: Leer informes RSS 266
Utilizar la extensión DOM de PHP 269
Trabajar con elementos 270
Trabajar con atributos 275
Prueba esto 8-4: Procesar recursivamente un documento árbol de XML 276
Alterar elementos y valores de atributos. 279
Crear nuevos documentos XML 281
Conversión entre DOM y SimpleXML. 283
Prueba esto 8-5: Leer y escribir archivos de configuración XML. 284
Resumen. 289

9 Trabajar con cookies, sesiones y encabezados 293

Trabajar con cookies 294
Aspectos básicos de las cookies 294
Atributos de las cookies 295

Encabezados de cookies 296
Establecer cookies. 297
Leer cookies 297
Eliminar cookies 298
Prueba esto 9-1: Guardar y restablecer preferencias del usuario. 298
Trabajar con sesiones 302
Aspectos básicos de las sesiones 302
Crear sesiones y variables de sesión 302
Eliminar sesiones y variables de sesión 304
Prueba esto 9-2: Rastrear visitas previas a la página 305
Utilizar encabezados HTTP 306
Prueba esto 9-3: Construir un formulario de ingreso mejorado. 308
Resumen. 313

Parte III Seguridad y solución de problemas

10 Manejo de errores. 317
Manejo de errores de script 318
Controlar el reporte de errores 321
Utilizar un controlador de errores personalizado 322
Prueba esto 10-1: Generar una página de errores legible. 325
Utilizar excepciones. 330
Utilizar excepciones personalizadas 334
Prueba esto 10-2: Validar datos de entrada en un formulario 335
Registro de errores 341
Depurar errores 342
Resumen. 347

11 Seguridad con PHP. 349
Higiene en los datos de entrada y salida 350
Asegurar los datos 353
Asegurar los archivos de configuración 353
Asegurar el acceso a la base de datos 354
Asegurar las sesiones 355
Validar los datos de entrada del usuario. 356
Trabajar con campos obligatorios. 356
Trabajar con números 358
Trabajar con cadenas de texto. 361
Trabajar con fechas. 367
Prueba esto 11-1: Validar datos de entrada de un formulario 368
Configurar la seguridad con PHP. 373
Resumen. 375

12 Extender PHP 377
Utilizar PEAR 378
Instalar paquetes PEAR 379
Prueba esto 12-1: Acceder a buzones electrónicos POP3 con PEAR 380
Utilizar PECL. 384
Instalar extensiones PECL 384
Prueba esto 12-2: Crear archivos Zip con PECL 386
Resumen. 388

Parte IV Apéndices

A Instalar y configurar los programas requeridos 391
Obtener el software 392
Instalar y configurar los programas 394
Instalar en UNIX. 394
Instalar en Windows 401
Probar el software 412
Probar MySQL 412
Probar PHP 413
Realizar pasos posteriores a la instalación. 415
Establecer la contraseña de superusuario en MySQL. 416
Configurar MySQL y Apache para comenzar automáticamente 416
Resumen. 417

B Respuestas a los autoexámenes 419

Índice 445

Sobre el autor

Vikram Vaswani es fundador y presidente del consejo de administración de Melonfire (www.melonfire.com/), firma de consultores expertos en herramientas y tecnología de código libre. Es apasionado en su apoyo al movimiento de código libre y contribuye frecuentemente con artículos y tutoriales sobre estas tecnologías (incluidas Perl, Python, PHP, MySQL y Linux) para la comunidad a su alcance. Algunos de sus libros anteriores son *PHP Soluciones de programación, MySQL: The Complete Reference* (www.mysql-tcr.com/) y *How to Do Everything with PHP and MySql* (www.everythingphpmysql.com/).

Vikram tiene más de diez años de experiencia como desarrollador de aplicaciones en PHP y MySQL. Es autor de las series de PHP para principiantes *PHP 101*, de Zend Technologies, y tiene amplia experiencia aplicando PHP en diversos ambientes (incluidos intranets corporativas, sitios Web en Internet con tráfico pesado y aplicaciones cliente de gestión crítica).

Estudiante en la Universidad de Oxford, Inglaterra, Vikram combina su interés por el desarrollo de aplicaciones Web con varias actividades adicionales. Cuando no está planeando cómo dominar el mundo, se entretiene leyendo novelas policiacas, viendo películas antiguas, jugando squash y participando en blogs. Lea más acerca de él y *Fundamentos de PHP* (en inglés) en www.php-beginners-guide.com.

Acerca del editor técnico

Chris Cornutt ha participado en la comunidad de PHP durante más de ocho años. Poco después de descubrir el lenguaje, inició su propia estación de noticias: PHPDeveloper.org, para compartir los acontecimientos más recientes y las opiniones de otros seguidores de PHP en todo el mundo. Chris ha escrito en publicaciones sobre PHP como *php\architect* e *International PHP Magazine*, sobre temas que van de la geocodificación al trackback. También es coautor del libro *PHP String Handling* (Wrox Press, 2003). Chris vive en Dallas, Texas, con su esposa e hijo, y trabaja para una gran empresa distribuidora de gas natural manteniendo su sitio Web y desarrollando aplicaciones basadas en PHP.

Prefacio

He programado computadores durante mucho tiempo. En ese lapso, me he movido entre más lenguajes de programación de los que puedo contar. Con cada nuevo lenguaje he dicho que una vez que aprendes a programar correctamente, el resto es sólo sintaxis. Todavía lo considero cierto para muchos lenguajes, pero con PHP eso sería una simplificación exagerada.

En PHP, por lo general existen varias maneras de realizar una tarea específica. Algunas son mejores que otras, pero otras (en particular cualquier instrucción que requiera el comando globals) están completamente equivocadas. Eso siempre confunde a los programadores novatos de PHP, porque si existen diversas maneras correctas de realizar una misma tarea, ¿cómo saber cuál es la mejor? Las "mejores prácticas" han sido un tema recurrente en la comunidad de PHP durante varios años y se ha intentado responder a esa pregunta.

Cada vez que un nuevo miembro de la comunidad de PHP me pregunta dónde puede aprender las mejores prácticas para programar en PHP, invariablemente le recomiendo las series "PHP 101" de Vikram, publicadas en muchos sitios Web. Su trabajo en esa serie de 14 partes le ha dado renombre en la comunidad como autoridad no sólo para enseñar a los nuevos usuarios, sino para enseñarles a programar correctamente.

He tenido el placer de trabajar con Vikram desde hace dos años en DevZone. Sus artículos son, sin lugar a dudas, de los más populares que hemos publicado. Al leer este libro comprenderás por qué.

Cal Evans
Zend's DevZone, Editor en jefe

Agradecimientos

Escribir un libro sobre un lenguaje que aún está desarrollándose es siempre un reto. Por fortuna, durante el proceso recibí ayuda inconmensurable por parte de diversos grupos de personas; todos ellos tuvieron un papel importante en hacer llegar este libro a tus manos.

Primero y antes que nada, quisiera darle las gracias a mi esposa, quien me ayudó a mantener los pies en la tierra durante el proceso. Belleza e inteligencia: yo carezco de ambos, pero afortunadamente ella tiene lo suficiente para compensar mi carencia. ¡Gracias, bebé!

Ha sido un placer trabajar con el equipo editorial y de mercadotecnia de McGraw-Hill Professional (como siempre). Éste es mi cuarto libro con ellos y parece que mejoran y mejoran con cada uno. La coordinadora de compras Jennifer Housh, el editor técnico Chris Cornutt y la editora ejecutiva Jane Brownlow guiaron este libro durante el proceso de desarrollo y desempeñaron un papel nada pequeño en transformarlo de un concepto a una realidad. Quisiera agradecerles por su experiencia, dedicación y esfuerzo.

Finalmente, por hacer el proceso entero de escribir un libro más divertido de lo que suele ser, gracias a: Patrick Quinlan, Ian Fleming, Bryan Adams, los Stones, Peter O'Donnell, la *Revista MAD*, Scott Adams, FHM, Gary Larson, VH1, George Michael, Kylie Minogue, *Buffy*, Farah Malegam, Adam y Anna, Stephen King, Shakira, Anahita Marker, Park End, John le Carre, Barry White, Gwen Stefani, Robert Crais, Robert B. Parker, Baz Luhrmann, Stefy, Anna Kournikova, John Connolly, Wasabi, Omega, Pidgin, Cal Evans, Ling's Pavilion, Tonka y su gemelo malévolo Bonka, Din Tai Fung, HBO, Mark Twain, Tim Burton, Harish Kamath, Madonna, John Sandford, Iron Man, the Tube, Dido, Google.com, *The Matrix*, Lee Child, Michael Connelly, Quentin Tarantino, Alfred Hitchcock, Woody Allen, Percy Jackson, St.

Hugh's College, Booty Luv, Mambo's and Tito's, Easyjet, Humphrey Bogart, Thai Pavilion, Brix, Wikipedia, *24*, Amazon.com, U2, The Three Stooges, Pacha, Oscar Wilde, Hugh Grant, Punch, Kelly Clarkson, Scott Turow, Slackware Linux, Calvin and Hobbes, Blizzard Entertainment, Alfred Kroop, Otto, Pablo Picasso, Popeye and Oliva, Denis Lehane, Trattoria, Dire Straits, Bruce Springsteen, David Mitchell, *The West Wing*, Santana, Rod Stewart, y todos mis amigos, en casa y en cualquier lugar.

Introducción

No importa desde qué perspectiva lo veas, PHP es asombroso: un lenguaje estructurado por programadores voluntarios que hoy en día tiene la envidiable distinción de ser utilizado por más de *un tercio* de los servidores Web del planeta. Flexible, escalable, extendible, estable, abierto. PHP es todo esto y más, por lo que resulta uno de los paquetes más populares de herramientas para la programación en el mundo.

Aún así, pregúntenme por qué me gusta PHP y mis razones no tienen nada que ver con lo ya expuesto y sí tienen todo que ver con lo amistoso y agradable del lenguaje. No hay sintaxis tortuosa ni código confuso en un script promedio de PHP: en cambio, es claro, comprensible, fácil de leer y hace que la programación y corrección de código sean un placer. Éste no es un logro minúsculo: una curva de aprendizaje corta permite que los programadores novatos comiencen rápidamente a "hacer algo útil" con el lenguaje, e incrementa tanto su interés como su nivel de adaptación. No se trata sólo de buen diseño, ¡también es buena publicidad!

Como proyecto de código abierto, PHP es completamente gratuito y tiene soporte de una comunidad mundial de voluntarios. Este método de código abierto, orientado a la comunidad, ha producido una plataforma significativamente más robusta y libre de errores en comparación con las opciones comerciales. Por lo tanto, utilizar PHP es bueno también para la economía de las empresas: les permite ahorrar en licencia y servidores costosos, mientras que, a la par, producen productos de alta calidad en tiempos más cortos.

Si éstas te parecen buenas razones para comenzar a explorar PHP, bueno, estás en el lugar correcto. Este libro te guiará por el mundo de PHP, te enseñará a escribir programas básicos

PHP, para después enriquecerlos con características más avanzadas, como consultas, datos de entrada en XML y extensiones de terceros. En pocas palabras, tiene todo lo necesario para convertirte en un programador experto en PHP... ¡y tal vez te haga reír en ocasiones!

Así que adelante y comencemos.

Quién debe leer este libro

Como seguramente lo adivinaste por el título, *Fundamentos de PHP* se concentra en los usuarios que inician en el lenguaje de programación PHP. Al contrario de otros libros, *Fundamentos de PHP* no da por hecho que el lector cuenta con conocimientos previos sobre programación en Web o fundamentos sobre bases de datos. En cambio, enseña a partir de ejemplos, utiliza proyectos por capítulo y ejemplos para explicar conceptos básicos; de esa manera, busca incrementar gradualmente la familiaridad del lector con los conceptos y las herramientas de programación de PHP. Por ello, se acopla mejor a los programadores novatos que están familiarizados con HTML y CSS, y quienes están interesados en aumentar sus habilidades para construir sitios dinámicos sobre bases de datos con PHP.

Qué abarca este libro

Fundamentos de PHP contiene información sobre el conjunto de herramientas de PHP 5.2 y 5.3-alfa y las características de mayor uso: integración con las bases de datos MySQL y SQLite, capacidades de procesamiento XML y extensiones de terceros. Se detiene a cubrir aspectos relacionados con la instalación de programas, sintaxis del lenguaje y estructura de datos, rutinas de control de flujo, funciones integradas y las mejores prácticas de programación. También contiene en cada capítulo numerosos proyectos de práctica que el lector puede "seguir al pie de la letra" para comprender en la práctica lo que se explica en el texto.

El siguiente esquema describe el contenido del libro y muestra su división en capítulos enfocados a cada tarea.

Parte I: Entender las bases de PHP

Capítulo 1: Introducción a PHP, presenta una introducción al lenguaje de programación PHP, explica por qué es tan popular en el desarrollo de aplicaciones Web y describe la manera en que interactúan los componentes de un sistema PHP típico.

Capítulo 2: Utilizar variables y operadores, explica los tipos de datos, las variables y los operadores propios de PHP, y aborda una de las aplicaciones más populares de PHP: procesamiento de los datos de entrada de un formulario.

Capítulo 3: Controlar el flujo del programa, presenta la manera de añadir inteligencia a los scripts PHP con declaraciones condicionales, automatizar tareas repetitivas con bucles y utilizar las funciones integradas para trabajar con cadenas de texto y números.

Capítulo 4: Trabajar con matrices, presenta una introducción del tipo de datos de matriz de PHP, explica la manera en que puede utilizarse con bucles y formularios Web y muestra algunas funciones integradas de PHP para ordenar, combinar, añadir, modificar y dividir arreglos.

Capítulo 5: Usar funciones y clases, proporciona un curso relámpago sobre dos de las funciones más complejas de PHP: funciones y clases. Recursión, listas de argumentos de longitud de variables, capacidad de extensión y reflejo, son sólo algunos de los temas abordados en este capítulo, que se concentra en el esquema de PHP para convertir bloques de código de uso frecuente en componentes reciclables.

Parte II: Trabajar con datos de otras fuentes

Capítulo 6: Trabajar con archivos y directorios, explica las funciones del sistema de archivos de PHP, presenta las rutinas PHP disponibles para leer y escribir archivos, crear y modificar directorios y trabajar con rutas de acceso y atributos de archivo.

Capítulo 7: Trabajar con bases de datos y SQL, explica las bases de datos y el lenguaje estructurado de consultas (SQL, Structured Query Language) y después presenta una introducción a las dos bases de datos más utilizadas con PHP: MySQL y SQLite. Muestra cómo puede utilizarse PHP para construir aplicaciones Web que interactúen con bases de datos para ver, añadir y editar datos y también aborda las nuevas características de portabilidad de las bases de datos.

Capítulo 8: Trabajar con XML, explica los conceptos y las tecnologías básicas de XML, y cómo puede ser utilizado PHP para procesar datos XML con el uso de la extensión SimpleXML.

Capítulo 9: Trabajar con cookies, sesiones y encabezados, explica las funciones integradas de PHP para crear sesiones y cookies y muestra cómo pueden utilizarse estas funciones para crear aplicaciones Web más amigables para los usuarios.

Parte III: Seguridad y solución de problemas

Capítulo 10: Manejo de errores, se concentra en el sistema de manejo de errores de PHP. Explica el modelo de errores y excepciones de PHP y muestra la manera de crear rutinas personalizadas para el manejo de errores diseñadas a la medida para necesidades específicas.

Capítulo 11: Seguridad con PHP, aborda temas de seguridad y ataques comunes, además de sugerir medios para incrementar la seguridad en las aplicaciones PHP. Aborda las técnicas clave para consolidar una aplicación respecto a la validación de datos de entrada, fuga de datos de salida y la configuración de seguridad de PHP.

Capítulo 12: Extender PHP, presenta una introducción a dos de los más grades depósitos de código PHP gratuito en Internet: PEAR, el depósito de extensiones y aplicaciones de PHP; y PECL, la biblioteca de la comunidad de extensiones PHP. Explica la manera en que pueden utilizarse los componentes, gratuitos y disponibles, ubicados en estos depósitos para añadir rápidamente nuevas capacidades y características a PHP, con lo que el desarrollo de aplicaciones se hace más rápido y efectivo.

Parte IV: Apéndices

Los apéndices incluyen material de referencia para la información presentada en las primeras tres partes.

Apéndice A: Instalar y configurar los programas requeridos, aborda el proceso para obtener, instalar y configurar Apache, PHP, MySQL y SQLite.

Apéndice B: Respuestas a los autoexámenes, proporciona las respuestas a los autoexámenes que aparecen al final de cada capítulo del libro.

Contenido de los capítulos

- **Prueba esto** Cada capítulo contiene por lo menos un proyecto independiente y desarrollable que resulta relevante para el tema que se trata y que el lector puede utilizar para obtener entendimiento práctico sobre el material expuesto.
- **Pregunta al experto** Cada capítulo contiene una o dos secciones Pregunta al experto que proporcionan una guía experta e información sobre preguntas que pueden surgir acerca del material expuesto en el capítulo.
- **Autoexamen** Cada capítulo finaliza con un autoexamen, una serie de preguntas que miden la información y las habilidades que aprendiste en ese capítulo. Las respuestas a los autoexámenes se incluyen en el apéndice B, al final del libro.

Parte I

Entender las bases de PHP

Capítulo 1

Introducción a PHP

Habilidades y conceptos clave

- Conocer la historia de PHP
- Conocer las características únicas de PHP para desarrollar aplicaciones Web
- Apreciar la interacción de los componentes de un sistema PHP
- Comprender la gramática y la estructura básicas de un script PHP
- Crear y ejecutar un programa PHP sencillo
- Insertar PHP en una página HTML

PHP. Tres letras que juntas constituyen el nombre de uno de los lenguajes de programación más populares para el desarrollo de Web, el Preprocesador de Hipertexto PHP. Y mientras tal vez sonríes por lo insulso y reiterativo del acrónimo, te diré que las estadísticas indican que PHP no debe tomarse a la ligera: actualmente este lenguaje se utiliza en más de *20 millones* de sitios Web y en más de un tercio de los servidores Web en todo el mundo; no es algo despreciable, especialmente cuando se considera que el lenguaje ha sido desarrollado por completo por una comunidad de voluntarios repartida en todo el mundo y está disponible en Internet ¡sin costo alguno!

Durante los últimos años, PHP se ha convertido, *de facto*, en la opción para el desarrollo de aplicaciones Web orientadas a bases de datos, sobre todo por su escalabilidad, facilidad, uso y el amplio soporte para diferentes bases de datos y formatos de éstos. Este primer capítulo te presentará una introducción amigable al mundo de PHP con un recorrido relámpago por su historia y sus características, y luego te guiará por la escritura y ejecución de tu primer programa PHP. ¡Así que continúa leyendo y comencemos!

Historia

La versión actual de PHP, PHP 5.3, se ha elaborado durante más de 14 años; su linaje puede rastrearse hasta 1994, cuando un desarrollador de nombre Rasmus Lerdorf creó por primera vez un conjunto de scripts CGI para monitorear visitas a la página electrónica donde mantenía su currículum. Esta versión temprana de PHP, llamada PHP/FI, era muy primitiva; aunque tenía soporte para datos de entrada y la base de datos mSQL, carecía de muchas de las características de seguridad y adiciones que se encuentran en las versiones modernas de PHP.

Más tarde, Lerdorf mejoró la versión 1.0 de PHP/FI y la lanzó como PHP/FI 2.0, pero no fue sino hasta 1997 cuando los desarrolladores Andi Gutmans y Zeev Suraski reescribieron el analizador sintáctico PHP y lo lanzaron como PHP 3.0; fue entonces cuando en realidad

comenzó a sobresalir el movimiento PHP. No sólo la sintaxis de PHP 3.0 fue más potente y consistente, también introducía una nueva arquitectura extensible que animaba a los desarrolladores independientes a crear sus propias versiones mejoradas y extensiones del lenguaje. Sobra decirlo, pero esto incrementó la adopción del lenguaje y fue mucho antes de que PHP 3.0 comenzara a aparecer en muchos miles de servidores Web.

La siguiente iteración del árbol de código, PHP 4.0, fue lanzada en 2000; ofrecía un nuevo motor, mejor rendimiento, más confiabilidad y soporte integrado para sesiones y características orientadas a objetos. Una encuesta de Nexen, realizada en julio de 2007, reveló que esta versión de PHP, la 4.*x*, era aún la de uso dominante en los sitios Web de Internet, porque 80% de los servidores con capacidades PHP del total de los encuestados lo tenían instalado. De cualquier manera, en julio de 2007, el equipo de desarrollo de PHP anunció que la versión 4.*x* no recibiría soporte después de diciembre del mismo año, con el fin de que las nuevas versiones fueran adoptadas por un mayor número de desarrolladores.

PHP 5.0, liberada en 2004, fue un rediseño radical de PHP 4.0, ostentando un motor completamente reprogramado, un modelo de objetos perfeccionado y muchas mejoras de seguridad y rendimiento. De particular interés para los desarrolladores fue el nuevo modelo de objetos, el cual ahora incluye soporte para los robustos paradigmas de programación orientada a objetos, como clases abstractas, destructores, múltiples interfaces e indicaciones de tipos de clase. PHP 5.0 también introdujo varias nuevas e importantes herramientas: capa de acceso común a bases de datos, manejo de excepciones semejante a Java y un motor de base de datos integrado.

PHP 5.3, la versión más reciente (y la que utilizaremos a lo largo de este libro), fue liberada en enero de 2008. Mejora las nuevas características introducidas en PHP 5.0 y también intenta corregir algunos de los problemas anotados por los usuarios de versiones anteriores. Algunas de las mejoras más notables en esta versión son: soporte para los espacios de nombres; un ambiente más limpio y seguro para el manejo del espacio de variables; soporte integrado para SQLite 3 y un nuevo controlador nativo para MySQL. Hasta ahora, todos estos cambios han contribuido a hacer de PHP 5.3 el mejor lanzamiento de PHP en la historia de catorce años del lenguaje, hecho ampliamente ilustrado por la encuesta de Netcraft, en abril de 2008, que muestra que PHP se utiliza en más de 30 millones de sitios Web.

Características únicas

Si estás familiarizado con otros lenguajes que se ejecutan del lado del servidor, como ASP.NET o JSP, tal vez te preguntes qué tiene de especial PHP o qué lo hace tan diferente de esas opciones competidoras. Bien, he aquí algunas razones:

Rendimiento Los scripts escritos en PHP se ejecutan más rápido que los escritos en otros lenguajes de creación de scripts; numerosos estudios comparativos independientes ponen este lenguaje por encima de sus competidores como JSP, ASP.NET y Perl. El motor de PHP

5.0 fue completamente rediseñado con un manejo óptimo de memoria para mejorar su rendimiento y es claramente más veloz que las versiones previas. Además, están disponibles aceleradores de terceros que pueden mejorar aún más el rendimiento y el tiempo de respuesta.

Portabilidad PHP está disponible para UNIX, Microsoft Windows, Mac OS y OS/2 y los programas escritos en PHP se pueden transportar de una plataforma a otra. Como resultado, las aplicaciones PHP desarrolladas en Windows, por ejemplo, se ejecutarán en UNIX sin grandes contratiempos. Esta capacidad de desarrollar fácilmente para múltiples plataformas es muy valiosa, en especial cuando se trabaja en un ambiente corporativo de varias plataformas o cuando se intenta atacar diversos sectores del mercado.

Fácil de usar "La sencillez es la mayor sofisticación", dijo Leonardo da Vinci y, de acuerdo con ello, PHP es un lenguaje de programación extremadamente sofisticado. Su sintaxis es clara y consistente y viene con una documentación exhaustiva para las más de 5 000 funciones incluidas en la distribución principal. Esto reduce de manera importante la curva de aprendizaje tanto para los desarrolladores novatos como para los expertos, y es una de las razones por las que PHP es favorecido como una herramienta rápida para la creación de prototipos que permitan el desarrollo de aplicaciones basadas en Web.

Código libre PHP es un proyecto de código libre; el lenguaje es desarrollado por un grupo de programadores voluntarios distribuidos por todo el mundo, quienes ponen a disposición gratuita el código fuente a través de Internet, y puede ser utilizado sin costo, sin pagos por licencia y sin necesidad de grandes inversiones en equipo de cómputo ni programas. Con ello se reduce el costo del desarrollo de programas sin afectar la flexibilidad ni la confiabilidad de los productos. La naturaleza del código libre implica que cualquier desarrollador, dondequiera que se encuentre, puede inspeccionar el árbol de código, detectar errores y sugerir posibles correcciones; con esto se produce un producto estable y robusto, en que las fallas, una vez descubiertas, se corrigen rápidamente, en algunas ocasiones, ¡horas después de ser descubiertas!

Soporte comunitario Una de las mejores características de los lenguajes a los que da soporte una comunidad, como PHP, es el acceso que ofrece a la creatividad e imaginación de cientos de desarrolladores ubicados en diferentes partes del mundo. En la comunidad PHP, los frutos de esta creatividad pueden ser encontrados en PEAR (PHP Extension and Application Repository), el repositorio de extensiones y aplicaciones de PHP (http://pear.php.net), y en PECL (PHP Extension Community Library), la biblioteca de la comunidad de extensiones PHP (http://pecl.php.net), que contienen cientos de soluciones y extensiones que los desarrolladores pueden ocupar para añadir sin esfuerzo nuevas funcionalidades a sus aplicaciones PHP. Utilizar estas soluciones suele ser una mejor opción en tiempo y costo, en vez de desarrollar desde cero tu propio código.

Soporte a aplicaciones de terceros Una de las fortalezas históricas de PHP ha sido su soporte a una amplia gama de diferentes bases de datos, entre las cuales se incluyen MySQL,

PostgreSQL, Oracle y Microsoft SQL Server. PHP 5.3 soporta más de quince diferentes motores de bases de datos, e incluye una API (interfaz de programación de aplicaciones) común para el acceso a base de datos. El soporte para XML facilita la lectura (y escritura) de documentos XML como si fueran estructuras de datos nativas de PHP; es posible accesar a colecciones de nodos XML utilizando XPath y transformar código XML en otros formatos con las hojas de estilo XSLT.

Y no termina aquí. La arquitectura extensible de PHP permite que los desarrolladores escriban sus propias adiciones personalizadas al lenguaje, de manera que hoy en día los desarrolladores de PHP pueden hacer que sus aplicaciones lean y registren imágenes en formato GIF, JPEG y PNG; enviar y recibir correos electrónicos utilizando protocolos SMTP, IMAP y POP3; colaborar con servicios Web utilizando protocolos SOAP y REST; validar datos de entrada utilizando expresiones regulares de Perl, además de crear y manipular documentos PDF. Más aún, PHP puede acceder a las bibliotecas de C, las clases de Java y los objetos COM, ¡y aprovechar el código escrito en esos lenguajes!

Pregunta al experto

P: ¿Necesito compilar los programas PHP antes de ejecutarlos, como en Java o C++?

R: No, porque PHP es un lenguaje interpretado. Una de las ventajas de los lenguajes interpretados es que te permiten hacer cambios en el código fuente y ponerlos a prueba de inmediato, sin necesidad de compilar primero el código fuente en forma de números binarios. Saltarse el proceso de compilación hace que el proceso de desarrollo sea más rápido, y PHP incluye un administrador de memoria y caché incorporado, con el fin de anular el efecto de tiempo adicional de carga a la memoria asociado con el uso de un intérprete.

Conceptos básicos de desarrollo

Cuando se desarrolla una aplicación para Web, la manera más común de hacerlo es incrustar el código PHP en uno o más documentos HTML estándar utilizando “etiquetas” o delimitadores especiales. He aquí un ejemplo:

```
<html>
 <head></head>
 <body>
  <div>
   <?php echo sqrt(49); ?>
  </div>
 </body>
</html>
```

Cuando un documento HTML con esas características es solicitado por el usuario, el servidor Web con capacidades PHP puede reconocer y ejecutar los bloques de código PHP e insertar el resultado en los datos de salida dentro del documento HTML, antes de enviarlo al usuario que lo solicita. El resultado es una página Web o una aplicación que parecen casi vivas; responden de manera inteligente a las acciones del usuario en virtud del programa lógico PHP incrustado en ellas. La figura 1-1 ilustra el proceso y muestra los cuatro elementos del esquema LAMP, que se describen más adelante en esta misma sección.

He aquí lo que sucede en la figura 1-1:

1. Joe inicia su explorador Web en casa y escribe el URL de una estación Web. Después de buscar el dominio, el explorador de Joe (el cliente) envía una solicitud HTTP a la dirección IP correspondiente al servidor.
2. El servidor Web encargado de manejar solicitudes HTTP para el dominio correspondiente se percata de que la URL finaliza con un sufijo `.php`. Como el servidor está programado para redireccionar automáticamente todas estas solicitudes a una capa PHP, simplemente invoca al intérprete PHP y pasa el contenido del archivo que lleva el sufijo mencionado.
3. El intérprete PHP segmenta el archivo y ejecuta el código que se encuentra en las etiquetas especiales PHP. Dentro de estas etiquetas puedes realizar operaciones de cálculo, procesar datos de entrada de usuario, interactuar con bases de datos, leer y escribir archivos... ¡y la lista continúa! Una vez que el intérprete del script ha terminado de ejecutar las instrucciones PHP, regresa el resultado al navegador, se limpia y retorna al estado de hibernación.
4. El servidor Web transmite los resultados al navegador de Joe, enviados por el intérprete.

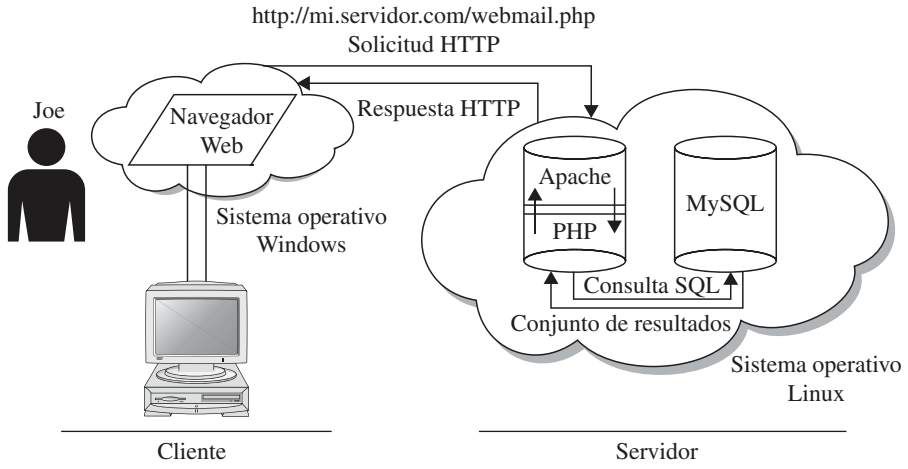


Figura 1-1 Esquema de desarrollo LAMP

De la anterior explicación debe quedar claro que para comenzar a construir aplicaciones PHP, tu ambiente de desarrollo debe contener por lo menos tres componentes:

- Un sistema operativo y un ambiente de servidor base (por lo general, Linux).
- Un servidor Web (por lo general Apache sobre Linux o ISS sobre Windows) para interceptar las solicitudes HTTP y procesarlas directamente o pasarlas al intérprete PHP para su correspondiente ejecución.
- Un intérprete PHP para segmentar y ejecutar el código PHP y regresar los resultados al servidor Web.

También existe por lo regular un cuarto componente, *opcional pero de enorme utilidad*:

- Un motor de base de datos (como MySQL) que almacena datos de la aplicación, acepta conexiones de la capa PHP y modifica o extrae datos de la base.

Un corolario importante de esta explicación es que el código PHP se ejecuta en el servidor y no en el explorador del cliente. Esto permite a los desarrolladores Web escribir código de programa que es completamente independiente y, por lo tanto, impermeable a los rasgos específicos del explorador cliente, importante ventaja sobre los lenguajes de creación de scripts que se ejecutan del lado del cliente, como JavaScript, los cuales suelen necesitar una lógica compleja para tomar en cuenta las diferencias específicas de cada explorador. Más aún, como el código se ejecuta completamente del lado del servidor y sólo el resultado es transmitido al cliente, resulta imposible que los usuarios vean el código fuente del programa PHP, importante ventaja de seguridad sobre lenguajes como JavaScript.

Pregunta al experto

P: ¿Cuánto cuestan los componentes del ambiente de desarrollo PHP?

R: Los cuatro componentes descritos en la sección anterior son, todos ellos, proyectos de código libre y, como tales, pueden descargarse de Internet sin costo alguno. Como principio general, tampoco existen cobros o cargos asociados con el uso de componentes para propósitos personales o comerciales, ni por desarrollar y distribuir aplicaciones que los utilizan. Sin embargo, si tu intención es desarrollar aplicaciones comerciales, es buena idea revisar los términos de la licencia de cada uno de estos componentes; por lo regular encontrarás las licencias mencionadas en el sitio Web del componente y también en el archivo de descarga.

Cuando están presentes los cuatro componentes (Linux, Apache, MySQL y PHP), el ambiente de desarrollo recibe el nombre de “plataforma LAMP”.

Crear tu primer script PHP

Ahora que sabes un poquito de PHP, demos un salto y comencemos a escribir algo de código. Los scripts que escribirás en las siguientes secciones serán, necesariamente, muy sencillos; pero no te preocupes, ¡las cosas se complicarán cuando aprendas más sobre el lenguaje!

Si no lo has hecho, es el momento propicio para encender tu computadora, descargar las versiones más recientes de Apache y PHP e instalarlas en tu ambiente de desarrollo. En el apéndice A de este libro encontrarás instrucciones detalladas para completar este procedimiento y para probar tu sistema de desarrollo con el fin de asegurarte de que todo funcione como debe después de la instalación. Así que manos a la obra y regresa cuando estés listo.

¿Ya terminaste? ¡Comencemos!

Escribir y ejecutar el script

Los scripts PHP son archivos de texto simple que contienen instrucciones PHP, en ocasiones combinadas con otros elementos, JavaScript, HTML y demás. Así, la manera más sencilla de escribir un script PHP consiste en abrir tu procesador de texto favorito y crear un archivo que contenga código PHP, como el siguiente:

```
<?php
// esta línea de código despliega una frase famosa
echo '¡Un caballo! ¡Un caballo! ¡Mi reino por un caballo!';
?>
```

Nombra este archivo *caballo.php* y guárdalo en un lugar de la raíz de documentos en tu servidor Web. Luego, inicia tu explorador Web y escribe el URL correspondiente a la ubicación del archivo. Debes ver algo semejante a lo que se presenta en la figura 1-2.

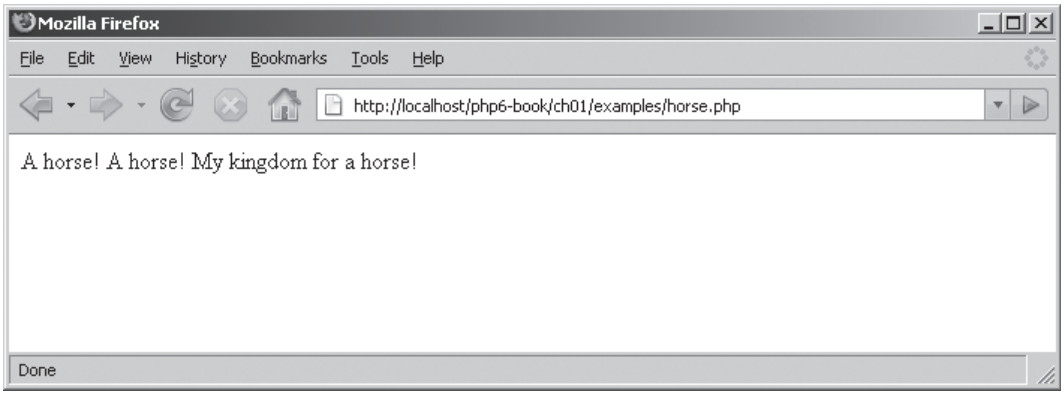


Figura 1-2 El resultado del script *caballo.php*

Entender el script

¿Qué sucede aquí? Bueno, cuando solicitaste el script *caballo.php*, el servidor Web Apache recibió tu solicitud, reconoció que el archivo era un script PHP (debido a la extensión `.php` del archivo) y lo envió al analizador sintáctico PHP y al intérprete para su procesamiento. El intérprete PHP leyó las instrucciones localizadas entre las etiquetas especiales `<?php ... ?>`, las ejecutó y regresó los resultados al servidor Web, el cual, a continuación, las envió a tu explorador cliente. Las instrucciones de esta instancia consisten en la invocación de la declaración PHP `echo`, encargada de presentar los datos de salida al usuario; los datos de salida que serán desplegados en el explorador cliente se encuentran encerrados entre comillas sencillas.

Incluso este sencillo script PHP contiene información valiosa que se debe analizar. Debe quedar claro, por ejemplo, que todo el código PHP debe estar encerrado entre las etiquetas `<?php ... ?>` y que toda declaración debe terminar con un punto y coma. Los saltos de línea dentro de las etiquetas PHP son ignorados por el analizador sintáctico.

Pregunta al experto

P: Escribí el siguiente script (omití el terminador punto y coma), y funcionó sin generar errores.

```
<?php
echo 'La reina ha muerto. ¡Viva la reina!'
?>
```

Esto contradice lo que dijiste antes, de que toda declaración PHP debe finalizar necesariamente con un punto y coma. Por favor explícamelo.

R: Omitir el punto y coma al final de una declaración PHP es uno de los errores más comunes que comenten los programadores PHP novatos, e invariablemente da como resultado un mensaje de error. Sin embargo, existe una situación de excepción (la que descubriste), en la cual no se genera mensaje de error a pesar de la omisión. El punto y coma no es necesario para finalizar la última línea de un segmento PHP, porque la etiqueta ?> incluye un punto y coma. Por eso el script que escribiste se ejecutó sin errores.

Advierte, sin embargo, que si bien funcionó en este caso, omitir el punto y coma de esta manera no es una buena práctica de programación. Después de todo, ¡nunca se sabe cuándo necesitarás añadir algo más al final del script!

Es posible incorporar cualquier tipo de comentarios dentro de un script PHP utilizando las mismas convenciones que JavaScript. Los comentarios de una sola línea deben ser precedidos por dos diagonales (//); los comentarios de varias líneas deben ir encerrados dentro de los caracteres para bloques de comentarios (/* . . . */). Estos comentarios son excluidos de los datos de salida del script PHP. He aquí algunos ejemplos:

```
<?php
// comentario de una sola línea
?>
<?php
/* comentario
   de varias
   líneas */
?>
```

Manejar los errores del script

El analizador sintáctico PHP tiene buen ojo. Si tu código incluye un error, desplegará un mensaje de advertencia o detendrá la ejecución del script en el punto donde aparece el error con una notificación de lo que salió mal; una u otra acción depende de la severidad del error. El capítulo 10 de este libro aborda con detalle los errores y el manejo de los mismos, pero en este momento resulta instructivo ver lo que sucede cuando el analizador sintáctico PHP localiza un error; así estarás mejor preparado para manejar situaciones similares cuando te ocurran.

Para generar un error de manera deliberada, regresa al script *caballo.php* que acabas de crear y escribe un punto y coma extra después de la palabra clave `echo`, de manera que el script se vea así:

```
<?php
// esta línea de código despliega una frase famosa
echo ; '¡Un caballo! ¡Un caballo! ¡Mi reino por un caballo!';
?>
```

Guarda este archivo y cárgalo en el explorador Web como lo hiciste con anterioridad. Esta vez debe aparecer algo semejante a lo que se muestra en la figura 1-3.

Como se ve en esa figura, el analizador sintáctico PHP es rápido para capturar errores en tu código. El mensaje de error generado por el analizador sintáctico es de gran utilidad: te indica en qué consistió el error y la línea donde ocurrió. Eso facilita enormemente (en casi todos los casos) localizar y corregir el error.

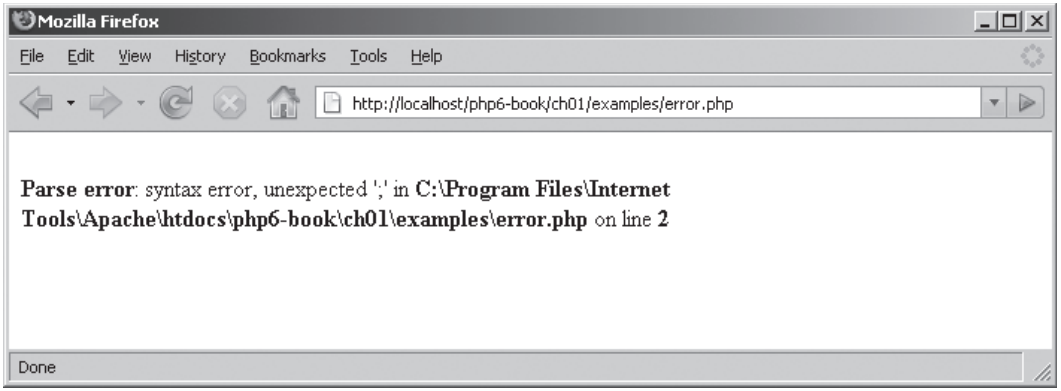


Figura 1-3 Los datos de salida generados por PHP cuando encuentra un error en el script

Prueba esto 1-1 Mezclar PHP con HTML

Cuando el analizador sintáctico PHP lee el script, ejecuta sólo el código que se encuentra entre las etiquetas PHP; ignora el resto y lo regresa "tal y como está". Gracias a ello, resulta muy fácil incrustar código dentro de un documento HTML para crear páginas Web que contengan todos los adornos propios del lenguaje HTML estándar, pero que además sean capaces de realizar cálculos complejos o leer y manipular datos provenientes de fuentes externas (como bases de datos o servicios Web).

Para ver cómo funciona esto en la práctica, considera el siguiente código:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Tabla de Color HTML</title>
    <style type="text/css">
    body {
        font-family: Verdana sans-serif;
    }
    td {
        border: solid 5px white;
    }
    </style>
  </head>
  <body>
    <h2>Colores con HTML y PHP</h2>
```

(*continúa*)

```
    <table>
      <tr>
        <td>Azul</td>
        <td style="width:40px; background-color:#0000ff"></td>
      </tr>
      <tr>
        <td><?php echo 'Rojo'; ?></td>
        <td style="width:40px; background-color:<?php echo '#ff0000';
?>"></td>
      </tr>
<?php
     // esta fila se genera por PHP
     echo "<tr>\n";
     echo "  <td>Verde</td>\n";
     echo "  <td style=\"width:40px; background-color:#00ff00\"></td>\n";
     echo "</tr>\n";
?>
    </table>
  </body>
</html>
```

Guarda este script como *colores.php* y despliégalo en tu explorador Web. Verás una página HTML con una tabla de tres filas y dos columnas; una de estas últimas contiene el color y la otra el nombre correspondiente (figura 1-4).

Utiliza el comando *Ver código fuente* de tu explorador para inspeccionar el código HTML de la página y verás que el analizador sintáctico PHP ha convertido el contenido de las declaraciones `echo` en código HTML para crear una página HTML completa. Ésta es una técnica

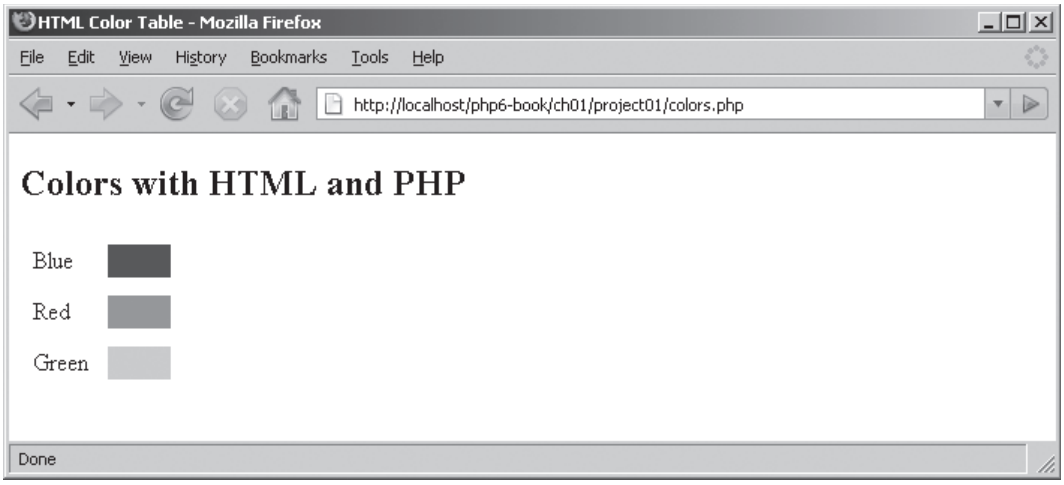


Figura 1-4 Una página Web que contiene colores y códigos de color, generada al mezclar PHP con HTML

muy común utilizada para construir aplicaciones Web con PHP, y la verás en casi todos los ejemplos que siguen.

Caracteres de escape especiales

Hay algo interesante en la página *colores.php* que merece ser explicado: las numerosas diagonales que se utilizan en el script. Mira la tercera fila de la tabla HTML generada con código PHP y luego su correspondiente código fuente HTML de la página desplegada y verás que ninguna de las diagonales aparece. ¿Adónde se fueron?

Hay una explicación sencilla a esto. Como ya has visto, los datos de salida que deben ser mostrados por PHP están encerrados entre comillas. ¿Pero qué sucede cuando los datos que deben desplegarse contienen sus propias comillas, como ocurre con el código HTML generado en *colores.php*? Si sólo encierras un juego de comillas dentro de otro, PHP se confundirá y no sabrá cuál debe ser presentado en pantalla y cuál es el que sirve para encerrar el valor de la cadena de caracteres, por lo que generará un error de segmentación. Por ello, para manejar este tipo de situaciones, PHP permite la identificación de ciertos caracteres de *escape* antecedidos por una diagonal invertida (\). Las llamadas *secuencias de escape* incluyen

Secuencia	Lo que representa
`\n`	carácter para insertar una línea
`\t`	un tabulador
`\r`	un salto de línea
`\"`	unas comillas dobles
`\'`	una comilla sencilla

Cuando el analizador sintáctico encuentra una de estas *secuencias de escape*, sabe que debe sustituirlas con su correspondiente valor antes de enviarlas al dispositivo de salida. Por ejemplo, considera la siguiente línea de código:

```
<?php
echo "Dijiste \"Hola\"";
?>
```

PHP sabe que las comillas precedidas por diagonales invertidas deben presentarse “tal y como están”, y no las confundirá con las comillas que marcan el principio y fin de una cadena de texto. Por esa razón ves diagonales invertidas que preceden a cada una de las comillas generadas por PHP e inserciones de línea en *colores.php*.

Pregunta al experto

P: ¿Por qué razón en el script *colores.php* se utilizan comillas sencillas en algunos lugares y dobles en otros?

R: Las secuencias de escape, como las utilizadas para insertar líneas (`\n`), saltos de línea (`\r`) y comillas dobles (`\"`), sólo pueden ser entendidas por el analizador sintáctico PHP cuando se encuentran encerradas entre comillas dobles. Si estas secuencias de escape se encuentran entre comillas sencillas, serán presentadas "tal y como aparecen".

Considera el siguiente fragmento de código y su presentación en pantalla, el cual ilustra la diferencia:

```
<?php
// datos de salida:
// Bienvenido
// a
// PHP
echo "Bienvenido\na\nPHP";
?>

<?php
// datos de salida: Bienvenido\na\nPHP
echo 'Bienvenido\na\nPHP';
?>
```

Ahora, el script *colores.php* es el responsable de generar dinámicamente el código HTML para la última fila de la tabla HTML. Para que este código se lea fácilmente, debe estar formado de manera que los elementos `<tr>` y `<td>` aparezcan en diferentes líneas. Para realizar esta tarea, es necesario utilizar la secuencia de escape que inserta una línea (`\n`), la cual, como ya se explicó, sólo es reconocida cuando se encuentra entre comillas dobles. Por ello, éstas son utilizadas en ciertos lugares dentro del script *colores.php*.

Aplicaciones de ejemplo

Por supuesto, en PHP existen muchas declaraciones, además de `echo`, y en los siguientes capítulos recibirás un curso rápido sobre las diferentes capacidades del lenguaje. Sin embargo, éste es un buen momento para darte un merecido descanso, tomar café y reflexionar sobre lo que acabas de aprender. Y sólo para aumentar tu interés sobre lo que viene, he aquí un pequeño ejemplo de las muchas aplicaciones para las que los desarrolladores quisieran que utilizaras PHP.

phpMyAdmin

La aplicación phpMyAdmin (www.phpmyadmin.net/) es una herramienta de administración basada en PHP para el sistema de administración de base de datos relacionales de MySQL. Uno de los más populares proyectos en la red SourceForge permite la creación y modificación de tablas y registros, administración de índices, ejecución de consultas *ad hoc* SQL, importación y exportación de datos y monitoreo del rendimiento de la base de datos.

phpBB

La aplicación phpBB (www.phpbb.com/) es una robusta implementación PHP de código libre de un sistema de boletines que es fácil de utilizar y de administración sencilla. Ofrece un tablero de discusión sencillo y amigable para los miembros del portal e incluye soporte para características como colocación de mensajes y sus respectivas respuestas, árboles de mensajes, búsquedas por tema y cuerpo de mensajes, temas, mensajes privados y muchas más.

Gallery

Gallery (http://gallery.menalto.com/) es un archivo de fotografías digitales muy configurable escrito en PHP. Soporta múltiples galerías de imágenes y numerosas palabras clave para cada foto e incluye soporte para características como creación automática de imágenes muestra, subtítulos y edición de imágenes, búsquedas por palabra clave y autenticación de nivel de galería.

Pregunta al experto

P: Entiendo el uso de las diagonales invertidas para marcar secuencias de escape. Pero, ¿qué sucede si necesito presentar una diagonal invertida en los datos de salida que aparecen en pantalla?

R: La solución es muy sencilla: ¡Utiliza una diagonal invertida doble!

```
<?php
echo "Esta es una diagonal invertida: \\";
?>
```

PoMMo

PoMMo (www.pommo.org/) es una aplicación de correo electrónico masivo escrita completamente en PHP. Es útil, sobre todo, para administrar y enviar mensajes de correo electrónico a uno o más suscriptores de listas de correo, y también soporta el uso de formularios personalizados para la recolección de datos de los suscriptores. Permite que los administradores importen y exporten datos de los suscriptores desde una base de datos MySQL, y que los usuarios manejen su suscripción a través de un panel de control en línea.

Smarty

Smarty (www.smarty.net/) es un esquema basado en PHP para separar la lógica de negocios de una aplicación PHP de la interfaz de usuario. Para conseguir esta separación utiliza hojas modelo que contienen espacios predeterminados para los datos de salida, los cuales son reemplazados por el contenido real durante la ejecución. Smarty soporta caché y anidamiento de hojas modelo, filtros previos y posteriores a la generación y funciones integradas para simplificar las tareas comunes. Se está convirtiendo rápidamente en uno de los motores de hojas modelo más populares para el desarrollo de aplicaciones basadas en PHP.

Squirrelmail

Squirrelmail (www.squirrelmail.org/) es un programa cliente de correo electrónico basado en Web y escrito en PHP; ofrece soporte para los protocolos SMTP e IMAP. Incluye la capacidad de enviar y recibir archivos adjuntos, administra libros de direcciones y manipula buzones electrónicos basados en el servidor. La aplicación es muy configurable y puede extender su uso para aceptar *plug-ins* y temas.

eZPublish

La aplicación eZPublish (www.ez.no/) es un sistema administrador de contenidos basado en PHP, útil para desarrollar tanto pequeñas páginas Web personales como grandes estaciones Web corporativas. Incluye soporte para crear y editar contenido, rastrear cambios, construir flujos de trabajo de edición personalizados y establecer una "tienda Web" con funciones de comercio electrónico integradas.

Mantis

Mantis (www.mantisbt.org/) es un sistema de rastreo de errores basado en Web y diseñado específicamente para rastrear y resolver problemas de software y otros proyectos. Soporta múltiples niveles de acceso a usuarios, diversos proyectos y varios niveles de prioridad e incluye un motor de búsqueda con características completas y varios reportes integrados para actualizar la fotografía del estatus del proyecto.

Wordpress

Wordpress (www.wordpress.org/) es una herramienta bien conocida para publicar *weblogs* (también conocidos como *blogs*). Permite que los usuarios publiquen y den mantenimiento a sus diarios en línea sobre sus actividades y soporta temas, etiquetas, revisión ortográfica automática, colocación de fotografías y videos, además de protección integrada contra correo electrónico no deseado. Es muy veloz, fácil de configurar y de uso sencillo, tres razones que lo han hecho famoso entre los usuarios de *blogs* de todo el mundo.

Resumen

Este capítulo ofreció una introducción amigable al mundo de PHP, se abordó la historia y evolución del lenguaje y se destacaron algunas de sus características únicas y sus ventajas frente a sus competidores. En él se explicaron varios componentes del ambiente de desarrollo típico de PHP y se mostró la manera en que interactúan entre sí. Finalmente, te condujo a comenzar a escribir código PHP, te mostró las reglas básicas de sintaxis propias del lenguaje y te explicó la manera de incrustar código PHP en documentos HTML utilizando las etiquetas especiales de PHP `<?php ... ?>` en proyectos sencillos. Estas habilidades básicas te serán de gran utilidad en los siguientes capítulos, donde se te ofrecerá un curso rápido de dos bloques fundamentales para la construcción de aplicaciones PHP: variables y operadores.

Si quieres saber más sobre los temas abordados en este capítulo, te serán útiles los siguientes vínculos:

- El sitio Web oficial de PHP, en **www.php.net**
- Estadísticas de uso de PHP, en **www.php.net/usage.php**
- Una breve historia de PHP, en **www.php.net/manual/en/history.php**
- La comunidad detrás de PHP, en **www.php.net/credits.php**

✓ Autoexamen Capítulo 1

1. ¿Cuáles son los cuatro componentes del esquema LAMP?
2. ¿Por qué PHP es superior a los lenguajes que se ejecutan del lado del cliente, como JavaScript?
3. ¿Qué hace la declaración `echo`?
4. ¿Qué sucede cuando el analizador sintáctico de PHP encuentra espacios o líneas en blanco en un script PHP?
5. ¿Qué carácter es obligatorio utilizar para terminar cada declaración PHP? Menciona una situación en la que omitir este terminador no produce un error.
6. ¿Qué es una secuencia de escape? Menciona tres secuencias de escape de uso común.
7. ¿Qué mostrarían en pantalla los siguientes scripts PHP?

A

```
<?php
                    echo            "Hoy amaneció\nsoleado y brillante"
          ;
?>
```

B

```
<?php
echo "Lo nuestro no es preguntarnos por qué;";
echo "Lo nuestro es hacer o morir";
?>
```

8. Encuentra el error en cada uno de los siguientes scripts PHP:

A

```
<?php
/* presenta una línea de salida /
echo 'Hola'
?>
```

B

```
<?php
echo '<p align="right">Obtén los derechos de autor para mí, 2008</p>';
?>
```

C

```
<?php
echo 'Línea 1;
?>
```

Capítulo 2

Utilizar variables y operadores

Habilidades y conceptos clave

- Crear y utilizar variables y constantes
- Comprender los tipos de datos simples de PHP
- Familiarizarse con algunas funciones integradas de PHP
- Realizar operaciones aritméticas
- Comparar y probar lógicamente las variables
- Manejar datos enviados a través de un formulario Web

En el capítulo anterior se dio una introducción amigable a PHP, y dejamos que te pusieras a trabajar un poco con dos proyectos sencillos. Sin embargo, como lo comprobarás dentro de poco, PHP sirve para mucho más que llenar los espacios en blanco de una página HTML. En este capítulo aprenderás sobre variables y operadores, los dos bloques de construcción básicos para cualquier programa PHP, y los utilizarás para desarrollar programas más sofisticados. También crearás tu primer *script* interactivo, que solicita datos de entrada del usuario y responde a lo enviado. Así que, sin más preámbulos, ¡pongamos manos a la obra!

Almacenar datos en variables

Una *variable* simplemente es un contenedor que se utiliza para almacenar información numérica y no numérica. Y como cualquier contenedor, puedes moverlo de un lugar a otro, añadirle cosas, vaciarlo en el piso y llenarlo con algo completamente diferente.

Para ampliar un poco más la analogía, así como es buena idea etiquetar todo contenedor, también debes darle un nombre a cada variable en tu programa. Como regla general, estos nombres deben tener un sentido y ser fáciles de entender. En el mundo real, esta práctica ayuda a encontrar rápidamente las cosas; en el mundo de la programación, hace que tu código sea más limpio y fácil de entender para los demás. Como alguien con la experiencia necesaria, te puedo decir que no hay nada más frustrante que pasar tres horas buscando en un montón de cajas la vajilla china preferida de mamá, sólo para descubrir que está en una caja etiquetada como “varios”, ¡junto con un hueso de plástico y bizcochos rancios!

En la práctica, los programadores suelen evitar los nombres de variables con acentos, diéresis y las eñes, por ser caracteres especiales del conjunto ISO-Latin-X; así se hace en los ejemplos de este libro.

PHP tiene algunas reglas sencillas para asignar nombre a las variables. Cada nombre de variable debe estar precedido por un signo de moneda (`$`) y debe comenzar con una letra o un guión bajo, seguido opcionalmente por más letras, números u otros guiones bajos. Los signos de puntuación comunes, como comas, comillas o puntos no son permitidos en los nombres de las variables; tampoco los espacios en blanco. Por ejemplo, `$root`, `$_num` y `$query2` son nombres de variable válidos, mientras que `$58%`, `$1day` y `email` no son válidos.

Asignar valores a variables

Asignar un valor a una variable en PHP es muy sencillo: se utiliza el símbolo de igual (=), que también es el operador de asignación de PHP. Este símbolo asigna el valor localizado a la derecha de la ecuación a la variable que se encuentra a la izquierda.

Para utilizar una variable en un script, simplemente se invoca su nombre en una expresión y PHP reemplaza este último por su valor cuando se ejecuta el *script*. He aquí un ejemplo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional //EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head><title /></head>
  <body>
   <?php
   // asignar valor a una variable
   $nombre = 'Juan';
   ?>
  <h2>Bienvenido al blog de <?php echo $nombre; ?></h2>
  </body>
</html>
```

En este ejemplo, se le asigna el valor `'Juan'` a la variable `$nombre`. La declaración `echo` se utiliza entonces para mostrar el valor de esta variable en la página Web.

También puedes asignar a una variable el valor de otra, o el resultado de un cálculo. El siguiente ejemplo muestra ambas situaciones:

```
<?php
// asignar valor a una variable
$fecha = 2008;

// asignar el valor de la variable a otra variable
$fechaActual = $fecha;

// realizar el cálculo
$fechaAnterior = $fechaActual - 1;

// datos de salida: '2007 ha terminado. ¡Bienvenido a 2008!'
echo "$fechaAnterior ha terminado. ¡Bienvenido a $fechaActual!";
?>
```

Pregunta al experto

P: ¿Es posible que un nombre de variable sea en sí una variable?

R: En raras situaciones, te será de utilidad asignar dinámicamente un nombre de variable, en el tiempo de ejecución. PHP te permite hacerlo al encerrar entre llaves la parte dinámica del nombre de la variable. El siguiente ejemplo ilustra este caso:

```
<?php
// definir una variable
$atributo = 'precio';

// crear una nueva variable
// su nombre surge dinámicamente
// del valor de la variable $atributo
${$atributo} = 678;

// dato de salida: 678
echo $precio;
?>
```

Destruir variables

Para destruir una variable, pasa la variable a la función PHP llamada apropiadamente `unset()`, como en el ejemplo siguiente:

```
<?php
// asignar un valor a la variable
$carro = 'Porsche';

// mostrar el valor de la variable
// datos de salida: 'Antes de unset(), mi carro es un Porsche'
echo "Antes de unset(), mi carro es un $carro";

// destruir la variable
unset($carro);

// mostrar el valor de la variable
// esto generará un error 'undefined variable' (variable indefinida)
// datos de salida:'después de unset(), mi carro es un '
echo "Después de unset(), mi carro es un $carro";
?>
```

NOTA

Tratar de utilizar o acceder a una variable que ha sido destruida con `unset()`, como en el ejemplo anterior, dará como resultado un mensaje de error *undefined variable* (variable indefinida). Este error puede o no ser visible en la página donde aparecen los datos de salida, dependiendo del nivel de reporte de errores en la configuración PHP. Para mayor información de la manera en que funcionan los mensajes de error, consulta el capítulo 10.

Como opción, es posible limpiar el contenido de la variable asignándole el valor especial PHP `NULL`. Puedes leer más acerca del tipo de dato PHP `NULL` en la siguiente sección, pero a continuación presento una revisión rápida del procedimiento para ilustrar la manera en que funciona:

```
<?php
// asignar un valor a la variable
$carro = 'Porsche';

// mostrar el valor de la variable
// datos de salida: 'Antes de unset(), mi carro es un Porsche'
echo "Antes de unset(), mi carro es un $carro";

// asignar un valor nulo a la variable
$carro = null;

// mostrar el valor de la variable
// datos de salida:'después de unset(), mi carro es un '
echo "Después de unset(), mi carro es un $carro";
?>
```

PRECAUCIÓN

Los nombres de variables en PHP son sensibles a las mayúsculas. Como resultado, `$ayuda` hace referencia a una variable diferente de `$AYUDA` y `$Ayuda`. Olvidar esta sencilla regla causa mucha frustración entre los programadores novatos de PHP.

Inspeccionar el contenido de la variable

PHP ofrece la función `var_dump()`, la cual acepta una variable y le aplica rayos X. He aquí un ejemplo:

```
<?php
// definir variables
$nombre = 'Fiona';
$edad = 28;

// mostrar el contenido de la variable
var_dump($nombre);
var_dump($edad);
?>
```

TIP

Existe la función `print_r()` que realiza una función semejante a `var_dump()` pero presenta menos información.

Comprender los tipos de datos de PHP

Los valores asignados a una variable PHP pueden corresponder a diferentes *tipos de datos*, que van desde sencillas cadenas de texto y datos numéricos hasta matrices y objetos complejos. En ejemplos previos ya has visto dos de ellos en acción: cadenas de texto y números. He aquí un ejemplo integral que introduce tres tipos de datos más:

```
<?php
// Booleano
$usuarioPermitido = true;

// entero
$capacidad = 15;

// punto flotante
$temporal = 98.6;

// cadena de texto
$gato = 'Siamés';

// nulo
$lugar = null;
?>
```

- Los *booleanos* son los tipos de datos más sencillos de PHP. Como un conmutador que sólo tiene dos estados: encendido y apagado, consiste en un solo valor que puede ser establecido como 1 (`true` [verdadero]) o 0 (`false` [falso]). En el ejemplo anterior `$usuarioPermitido` es una variable booleana establecida como verdadera (`true`).
- PHP también admite dos tipos de datos numéricos: *enteros* y *valores de punto flotante*. Los valores de punto flotante (también conocidos como *flotantes* o *dobles*) son números decimales o fracciones, mientras que los enteros son números naturales. Ambos pueden ser menores que, mayores que o iguales a cero. En el ejemplo, la variable `$capacidad` contiene un valor entero, mientras que la variable `$temporal` contiene un valor de punto flotante.
- Para datos diferentes a los numéricos, PHP ofrece el tipo de dato cadena de caracteres (*string*), que puede almacenar letras, números y caracteres especiales. Las cadenas de caracteres deben ir encerradas entre comillas dobles o sencillas. En el ejemplo anterior, `$gato` es una variable de cadena de texto que contiene el valor `'Siamés'`.

- También tenemos el tipo de dato `NULL` (nulo), que es un tipo de dato "especial" introducido por primera vez en PHP 4. `NULL` es utilizado en PHP para representar variables "vacías"; una variable de tipo `NULL` es una variable sin datos. En el ejemplo anterior, `$lugar` es una variable `NULL`.

Pregunta al experto

P: ¿PHP acepta números escritos en hexadecimal, octal o notación científica?

R: Sí, sí y sí. He aquí algunos ejemplos:

```
<?php
// 8, especificado como un valor octal
$a = 010;

// 1500, especificado como un valor hexadecimal
$b = 0x5dc;

// 690, en notación científica
$c = 6.9E+2;
?>
```

PRECAUCIÓN

Muchos desarrolladores novatos en PHP creen erróneamente que al asignar a una variable el valor de una cadena de caracteres vacía (`''`) automáticamente queda vacía. Esto no es cierto, porque PHP no considera equivalentes los valores de una cadena de texto vacía y un tipo de datos `NULL`. Para eliminar por completo el contenido de una variable, siempre debe declararse el tipo de datos `NULL`.

Establecer y verificar el tipo de datos de la variable

Contrariamente a otros lenguajes de programación, donde el tipo de dato el programador debe definir explícitamente la variable, PHP determina de manera automática el tipo de variable por el contenido que almacena. En caso de que el contenido de la variable cambie durante la duración del script, el lenguaje establecerá automáticamente el nuevo tipo de dato apropiado para la variable, de acuerdo con el cambio.

He aquí un ejemplo que ilustra este *malabarismo con los tipos de datos*:

```
<?php
// define una variable cadena de caracteres
$soy = 'Sara';

// datos de salida: 'cadena de caracteres'
echo gettype($soy);
```

```
// asigna un nuevo valor entero a la variable
$soy = 99.8;

// datos de salida: 'doble'
echo gettype($soy);

// destruye la variable
unset($soy);

// datos de salida: 'NULL'
echo gettype($soy);
?>
```

Este ejemplo presenta por primera vez el operador PHP `gettype()`, que es una herramienta útil y ligera que se utiliza para averiguar el tipo de una variable. Como lo muestran los datos de salida del script, la variable `$soy` inicia como una cadena de caracteres, con el valor `'Sara'`. Después se le asigna como valor el número 99.8, con lo que se convierte automáticamente en una variable de punto flotante. A continuación, la variable se invalida con el método `unset()`, el cual borra su valor y la transforma en una variable `NULL`. PHP es la mano invisible detrás de todas estas transformaciones, restableciendo internamente el tipo de dato de la variable `$soy` de una cadena de texto a un número de punto flotante y de ahí a un valor nulo.

Sin embargo, esto no significa que estés por completo a merced de PHP; es posible establecer específicamente el tipo de variable PHP al *convertir* la variable en un tipo específico antes de utilizarla. La conversión es una técnica de uso común para los programadores de Java; para utilizarla, simplemente especifica entre paréntesis, y del lado derecho de la ecuación, el tipo de dato que deseas asignar a una variable. Considera el siguiente ejemplo, que ilustra la conversión de un valor de punto flotante a un entero:

```
<?php
// define una variable de punto flotante
$velocidad = 501.789;

// conversión a entero
$nuevaVelocidad = (integer)$velocidad;

// datos de salida: 501
echo $nuevaVelocidad;
?>
```

Además de la función `gettype()`, PHP cuenta con otras más especializadas para probar si una variable corresponde a un tipo de datos específico. La tabla 2-1 tiene la lista de estas funciones.

TIP

¿Recuerdas la función `var_dump()` que vimos en la sección anterior? Si miras con cuidado los datos de salida que genera, notarás que además de decirte lo que contiene la variable, también muestra el tipo de datos correspondiente.

Función	Propósito
`is_bool()`	Prueba si la variable contiene un valor booleano
`is_numeric()`	Prueba si la variable contiene un valor numérico
`is_int()`	Prueba si la variable contiene un valor entero
`is_float()`	Prueba si la variable contiene un valor de punto flotante
`is_string()`	Prueba si la variable contiene un valor de cadena de texto
`is_null()`	Prueba si la variable contiene un valor NULL
`is_array()`	Prueba si el valor que contiene la variable es una matriz
`is_object()`	Prueba si el valor que contiene la variable es un objeto

Tabla 2-1 Funciones de PHP para probar los tipos de dato de las variables

Usar constantes

Hasta ahora el capítulo se ha concentrado en las variables, que son útiles para almacenar y cambiar valores durante el tiempo de vida del script PHP. Pero, ¿qué sucede si necesitas almacenar un valor fijo, que permanezca estático durante el curso del script? Bien, en esos casos se debe utilizar una *constante*.

Como su nombre lo sugiere, las constantes son contenedores de PHP para valores que permanecen constantes y que nunca cambian. Por lo regular se utilizan para valores bien conocidos de antemano y que son utilizados, sin cambio alguno, en diferentes lugares de la aplicación. Buenos candidatos para constantes son los niveles de depuración y registro, los números de versión, las marcas de configuración y las fórmulas.

Las constantes se definen en PHP con la función `define()`, la cual acepta dos argumentos: el nombre de la constante y su valor. Los nombres de las constantes deben seguir las mismas reglas que las variables, con una sola excepción: el prefijo `$` no se requiere para los nombres de constantes.

He aquí un ejemplo para definir y utilizar una constante dentro de un script:

```
<?php
// define constantes
define('PROGRAMA', 'The Matrix');
define('VERSION', 11.7);

// usar constantes
// datos de salida: 'Bienvenido a The Matrix (versión 11.7)'
echo 'Bienvenido a ' . PROGRAMA . ' (versión ' . VERSION . ')';
?>
```

NOTA

Por convención, los nombres de constantes se escriben en mayúsculas; esto se hace con el fin de identificarlos y diferenciarlos rápidamente de las variables "regulares" en un script.

Manipular variables con operadores

Por sí solas, las variables sólo son contenedores de información. Para realizar algo útil con ellas necesitas *operadores*. Los procesadores son símbolos que le indican al procesador PHP que realice ciertas acciones. Por ejemplo, el símbolo de suma (+) le dice a PHP que sume dos variables o valores, mientras que el símbolo mayor que (>) es un operador que le dice a PHP que compare dos valores.

PHP da soporte a más de 50 de estos operadores, que van desde los destinados a operaciones aritméticas hasta operadores para comparaciones lógicas y cálculos complejos. En esta sección se abordan los operadores de uso más común.

Pregunta al experto

P: ¿Cuándo debo usar una variable y cuándo una constante?

R: Las variables son para almacenamiento temporal; utilízalas para valores que tal vez cambiarán a lo largo del script. Las constantes son un poco más permanentes; utilízalas para valores que tal vez se mantendrán fijos y a los que se hacen múltiples referencias dentro del script. A diferencia de las variables, las constantes no pueden destruirse y no pueden generarse sus nombres dinámicamente.

Realizar operaciones aritméticas

PHP da soporte a todas las operaciones aritméticas estándar, como se muestra en la lista de operadores de la tabla 2-2.

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	Divide y regresa el cociente
%	Divide y regresa el residuo

Tabla 2-2 Operadores aritméticos comunes

He aquí un ejemplo que muestra estos operadores en acción:

```
<?php
// define variables
$x = 10;
$y = 5;
$z = 3;

// suma
$suma = $x + $y;
echo "$x + $y = $suma\n";

// resta
$resta = $x - $y;
echo "$x - $y = $resta\n";

// multiplica
$producto = $x * $y;
echo "$x * $y = $producto\n";

// divide y obtiene el cociente
$cociente = $x / $y;
echo "$x / $y = $cociente\n";

// divide y obtiene el módulo
$modulo = $x % $y;
echo "$x % $y = $modulo\n";
?>
```

Pregunta al experto

P: ¿Hay algún límite para la longitud de los enteros?

R: Sí lo hay, pero es muy alto: 2147483647. Este límite está definido por PHP en su constante `PHP_INT_MAX`, así que puedes comprobarlo tú mismo en cualquier momento.

Unir cadenas de texto

Para combinar cadenas de texto, utiliza el operador de unión de PHP, el cual resulta ser un punto (.). El siguiente ejemplo lo ilustra:

```
<?php
// define variables
$pais = 'Inglaterra';
$ciudad = 'Londres';
```

```
// combina ambas en una sola línea
// datos de salida: 'Bienvenido a Londres, la ciudad más fría de toda Inglaterra'
echo 'Bienvenido a ' . $ciudad . ', la ciudad más fría de toda ' . $pais;
?>
```

Comparar variables

PHP te permite comparar una variable o un valor con otro mediante su amplia variedad de operadores de comparación, presentados en la tabla 2-3.

He aquí un ejemplo para mostrar estos operadores en acción:

```
<?php
// define variables
$p = 10;
$q = 11;
$r = 11.3;
$s = 11;

// prueba si $q es mayor que $p
// regresa el valor de verdadero (true)
echo ($q > $p);

// prueba si $q es menor que $p
// regresa el valor de falso (false)
echo ($q < $p);

// prueba si $q es mayor que o igual a $s
// regresa el valor de verdadero (true)
echo ($q >= $s);

// prueba si $r es menor o igual que $s
// regresa el valor de falso (false)
echo ($r <= $s);
```

Operador	Descripción
==	Igual a
!=	Diferente de
>	Mayor que
>=	Mayor que o igual a
<	Menor que
<=	Menor que o igual a
===	Igual a y del mismo tipo

Tabla 2-3 Operadores de comparación comunes

```
// prueba si $q es igual a $s
// regresa el valor de verdadero (true)
echo ($q == $s);

// prueba si $q es igual a $r
// regresa el valor de falso (false)
echo ($q == $r);
?>
```

Mención especial merece aquí el operador `===`, excluido del ejemplo anterior. Este operador permite realizar una comparación estricta entre variables: sólo regresa el valor verdadero (`true`) si las dos variables o valores comparados tienen la misma información y son del mismo tipo de datos. De esta manera, en el ejemplo siguiente la comparación entre las variables `$bool` y `$num` regresará el valor verdadero (`true`) cuando se les compare con el operador `==`, pero falso (`false`) cuando se les compare con `===`:

```
<?php
// define variables de dos tipos
// pero con el mismo valor
$bool = (boolean) 1;
$num = (integer) 1;

// regresa el valor de verdadero (true)
echo ($bool == $num);

// regresa el valor de falso (false)
echo ($bool === $num);
?>
```

Realizar pruebas lógicas

Cuando se construyen expresiones condicionales complejas (tema que se abordará con detalle en el capítulo 3), constantemente te encontrarás con situaciones en las que es necesario combinar una o más pruebas lógicas. Los tres operadores lógicos más usados en PHP, listados en la tabla 2-4, están pensados específicamente para esas situaciones.

Operador	Descripción
`&&`	Y (AND)
`||`	O (OR)
`!`	NO (NOT)

Tabla 2-4 Operadores lógicos comunes

Los operadores lógicos en realidad muestran su poder cuando se combinan con pruebas condicionales; el siguiente ejemplo es sólo para ilustrar esta gran capacidad; en el capítulo 3 encontrarás mejor material para entretenerte.

```
<?php
// define variables
$precio = 100;
$tamano = 18;

// prueba lógica Y (AND)
// regresa el valor de verdadero (true) si ambas comparaciones son verdaderas
// en este caso regresa el valor de verdadero (true)
echo ($precio > 50 && $tamano < 25);

// prueba lógica O (OR)
// regresa el valor de verdadero (true) si cualquiera de las
comparaciones es verdadera
// en este caso regresa el valor de falso (false)
echo ($precio > 150 || $tamano > 75);

// prueba lógica NO (NOT)
// invierte la prueba lógica
// en este caso regresa el valor de falso (false)
echo !($tamano > 10);
?>
```

Otros operadores útiles

Hay unos cuantos operadores más que suelen ser útiles durante el desarrollo de PHP. Primero, el operador de asignación de suma, representado por el símbolo `+=`, permite sumar y asignar un nuevo valor a la variable simultáneamente. El siguiente ejemplo lo ilustra:

```
<?php
// define variable
$cuenta = 7;

// suma 2 y asigna el valor de la suma a la variable
$cuenta += 2;

// dato de salida: 9
echo $cuenta;
?>
```

En el ejemplo anterior, la expresión `$cuenta +=7` es equivalente a la expresión `$cuenta + 2`, una operación de suma seguida por la asignación del resultado a la misma variable. En la misma línea de acción, existen operadores para otras asignaciones matemáticas y cadenas de texto. La tabla 2-5 presenta la lista.

Operador	Descripción
`+=`	Suma y asigna
`-=`	Resta y asigna
`*=`	Multiplica y asigna
`/=`	Divide y asigna el cociente
`%=`	Divide y asigna el residuo
`.=`	Concatena y asigna (solamente cadenas de texto)

Tabla 2-5 Operadores de asignación comunes

He aquí algunos ejemplos de estas acciones:

```
<?php
// define variables
$cuenta = 7;
$edad = 60;
$saludo = 'Bien';

// resta 2 y reasigna el nuevo valor a la variable
// equivalente a $cuenta = $cuenta - 2
// dato de salida: 5
$cuenta -= 2;
echo $cuenta;

// divide entre 5 y reasigna el nuevo valor a la variable
// equivalente a $edad = $edad / 5
// dato de salida: 12
$edad /= 5;
echo $edad;

// añade una nueva cadena de texto y reasigna el nuevo valor a la
variable
// equivalente a $saludo = $saludo . 'venidos'
// dato de salida: 'Bienvenidos'
$saludo .= 'venidos';
echo $saludo;
?>
```

Más adelante, en este libro, también encontrarás los operadores de autoincremento y autodecremento, representados por los símbolos `++` y `--`, respectivamente.

Estos operadores suman 1 o restan 1 automáticamente a la variable en que se aplican. He aquí un ejemplo:

```
<?php
// define variable
$cuenta = 19;

// incremento
$cuenta++;

// dato de salida: 20
echo $cuenta;

// ahora el decremento
$cuenta --;

// dato de salida: 19
echo $cuenta;
?>
```

Estos operadores suelen encontrarse en contadores reiterativos, otro tema que abordaremos con detalle en el capítulo 3.

Comprender la precedencia de los operadores

Recordando las clases de matemáticas, tal vez te enseñaron el CODMSR, acrónimo que especifica el orden en el que una calculadora o una computadora realiza una secuencia de operaciones matemáticas: Corchete, Orden, División, Multiplicación, Suma y Resta. Pues bien, PHP sigue un conjunto de reglas similares para determinar cuáles operadores tienen precedencia sobre otros, y aprender estas reglas te ahorrará incontables horas de frustración depurando un cálculo que parece bien estructurado y en el que, sin embargo, algo anda mal, ¡porque siempre regresa un resultado erróneo!

La siguiente lista (una versión reducida de una mucho más larga que forma parte del manual de PHP) presenta las reglas de precedencia más importantes. Los operadores en el mismo nivel tienen precedencia equivalente.

- `++ --`
- `!`
- `* / %`
- `+ - .`
- `< <= > >=`

- `== != === !==`
- `&&`
- `||`
- `= += -= *= /= .= %= &= |= ^=`

Pregunta al experto

P: Las reglas de precedencia de PHP son difíciles de recordar. ¿Existe alguna otra manera de indicar a PHP el orden en que quiero que se realicen los cálculos?

R: Sí. Los paréntesis siempre tienen la más alta precedencia, por lo que encerrar una expresión entre paréntesis obligará a PHP a evaluarlos primero. Cuando utilices paréntesis anidados, recuerda que la evaluación comienza con el último conjunto de paréntesis de izquierda a derecha y sigue el orden de dentro hacia afuera (como si se quitaran capas de una cebolla desde adentro). A manera de ejemplo, considera la expresión `(((4 * 8) – 2 ) / 10)`, que da como resultado `3` con paréntesis y `31.8` sin ellos.

Prueba esto 2-1 Construir un convertidor dólar-euro

Ahora tomemos un breve descanso de toda esta teoría e intentemos aplicar algo de ella a un proyecto práctico y real: un convertidor monetario de dólares a euros. En este proyecto se aplicarán algunos conocimientos que has aprendido en secciones anteriores sobre variables, constantes y operadores aritméticos; ¡también será de utilidad la próxima vez que tomes tu avión para pasar las vacaciones en Europa!

He aquí el código (*convertidor.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 2-1: Convertidor Monetario USD/EUR</title>
  </head>
  <body>
    <h2>Proyecto 2-1: Convertidor Monetario USD/EUR</h2>
    <?php
    // define tasa de cambio
```

(*continúa*)

```
    // 1.00 USD = 0.70 EUR
    define ('TASA_DE_CAMBIO', 0.70);

    // define la cantidad de dólares
    $dolares = 150;

    // realiza la conversión y presenta el resultado
    $euros = $dolares * TASA_DE_CAMBIO;
    echo "$dolares USD americanos son equivalentes a: $euros EUR";
    ?>
  </body>
</html>
```

Si has seguido con cuidado los ejemplos, debes entender este script con gran facilidad. Comienza por definir una constante llamada `TASA_DE_CAMBIO` que (¡sorpresa!) almacena la tasa de cambio entre el dólar y el euro (aquí damos por hecho que un dólar equivale a 0.70 euros). A continuación, define una variable llamada `$dolares`, que almacena el monto que será convertido a euros, y después realiza una operación aritmética utilizando el operador `*` sobre la variable `$dolares` y la constante `TASA_DE_CAMBIO` para obtener el número equivalente en euros. El resultado se almacena en una nueva variable llamada `$euros`, misma que es presentada en la página Web.

La figura 2-1 ilustra la manera en que aparecen los datos de salida.

Para convertir una cantidad diferente de dólares, simplemente cambia el valor de la variable `$dolares`. ¡Vamos, inténtalo y compruébalo tú mismo!

Figura 2-1 Los datos de salida del convertidor dólares-euros

Manejar datos de entrada para formularios

Hasta ahora, todos los ejemplos que has visto tienen sus variables claramente definidas al inicio del script. Sin embargo, a medida que tus scripts PHP se hagan más complejos, esta feliz situación cambiará y tendrás que aprender a interactuar con datos de entrada que proporciona el usuario. La fuente más común para transmitir estos datos es un formulario Web, y PHP cuenta con un mecanismo sencillo para recuperar información enviada en estos formularios.

Como ejemplo, considera el siguiente formulario Web simple (*lista.html*), con el que se selecciona una marca de automóvil y se ingresa el color deseado:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head><title /></head>
  <body>
    <h2>Selecciona Tu Automóvil</h2>
    <form method="post" action="carro.php">
      Tipo: <br />
      <select name="tipo">
        <option value="Porsche 911">Porsche 911</option>
        <option value="Volkswagen Beetle">Volkswagen Beetle</option>
        <option value="Ford Taurus">Ford Taurus</option>
      </select><p />
      Color: <br />
      <input type="text" name="txtColor" /> <p />
      <input type="submit" />
    </form>
  </body>
</html>
```

Hasta aquí el formulario es muy sencillo: tiene una lista de selección y un recuadro de texto. La figura 2-2 muestra cómo aparece en el explorador Web.

Pon atención al atributo `'action'` del formulario Web: hace referencia al script PHP llamado *carro.php*. Éste es el script que recibe los datos del formulario una vez que éste ha sido enviado. También debes prestar atención al atributo `'method'` del formulario, el cual especifica que el envío de los datos será a través del método POST.

Con estos dos datos bien comprendidos, veamos ahora el script *carro.php*:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head><title /></head>
  <body>
    <h2>¡Éxito!</h2>
```

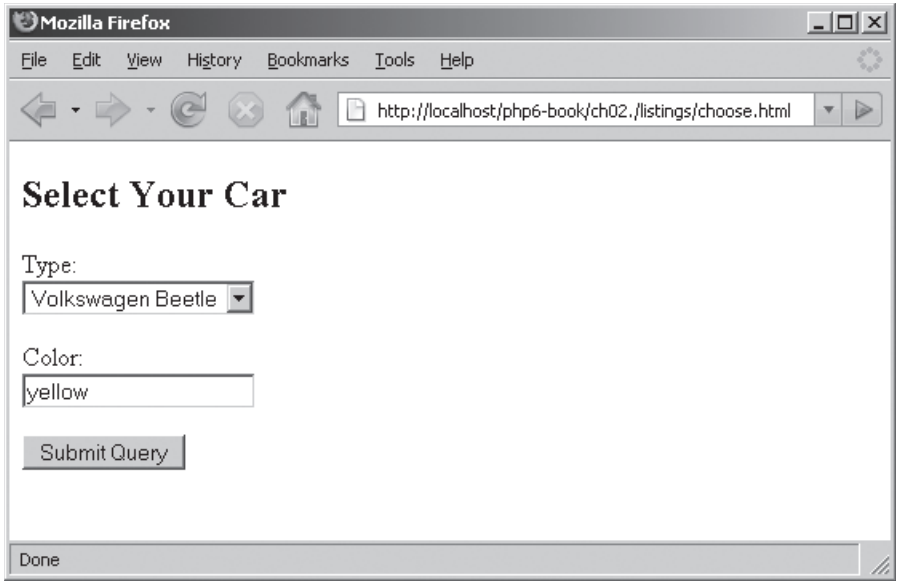


Figura 2-2 Un formulario sencillo

```
    <?php
    // obtiene los datos de entrada del formulario
    $tipo = $_POST['tipo'];
    $color = $_POST['txtColor'];

    // utiliza los datos de entrada del formulario
    echo "Tu $tipo $txtColor está listo. ¡Maneja con cuidado!";
    ?>
  </body>
</html>
```

¿Qué sucede aquí? Bien, cada vez que un formulario es enviado a un script PHP con el método POST, las variables internas del formulario y sus respectivos valores están disponibles para el script PHP a través de un contenedor de variables especial llamado `$_POST`. Accesar al valor introducido en un campo particular del formulario se convierte en una simple referencia a `$_POST` con el correspondiente nombre del campo, como se aprecia en el script anterior.

Considera, por ejemplo, la tarea de acceder al color escrito por el usuario en el formulario Web. En el código del formulario se puede ver que el campo para la inserción de datos designado para esta información lleva el nombre `'txtColor'`. Por lo tanto, dentro del script PHP, el valor ingresado en este campo de texto puede ser accesado utilizando la sintaxis `$_POST['txtColor']`.

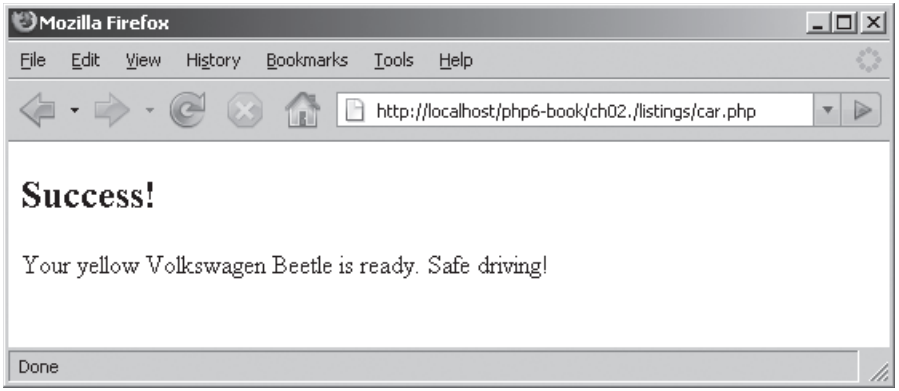


Figura 2-3 El resultado de enviar el formulario

Este valor puede ser utilizado de la manera convencional: imprimirse en una página Web, asignarse a otra variable o manipularse con los operadores presentados en las secciones anteriores.

La figura 2-3 muestra el resultado enviado por el formulario.

PHP también contempla los casos en que el formulario Web envía datos utilizando el método GET, en lugar de POST: los datos de entrada del formulario enviados con el método GET encuentran su equivalente en el contenedor de variables `$_GET`, el cual puede utilizarse al hacer referencia a `$_GET` en lugar de `$_POST`.

Pregunta al experto

P: Entendí en las secciones anteriores los aspectos de las variables y la manera en que funcionan, pero `$_GET` y `$_POST` no siguen las mismas reglas. ¿Qué sucede aquí?

R: Las variables `$_GET` y `$_POST` son diferentes a los tipos de variables numéricas y las cadenas de texto que hemos visto en este capítulo. Son un tipo de variables más complejas llamadas *matrices*, las cuales pueden contener más de un valor al mismo tiempo y también siguen reglas diferentes para almacenar y acceder a valores dentro de ellas. Las matrices se abordarán ampliamente en el capítulo 4; en ese punto todo lo que has leído sobre `$_GET` y `$_POST` cobrará sentido.

Prueba esto 2-2 Construir un muestrario HTML interactivo de colores

Ahora que has aprendido a acceder datos de entrada del formulario mediante un script PHP, elaboremos una aplicación que muestre esta característica de manera más específica. En este proyecto, construirás un muestrario de colores HTML, una herramienta que te permitirá ingresar valores RGB de colores y presentará una muestra del tono correspondiente en el explorador Web. Los valores RGB de los colores se introducirán en un formulario y PHP los procesará, con lo que crearemos un ejemplo aplicable a la realidad con los conocimientos que has adquirido en las secciones anteriores.

Primero el formulario Web (*color.html*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 2-2: Muestrario HTML Interactivo de Colores</title>
  </head>
  <body>
    <h2>Proyecto 2-2: Muestrario HTML Interactivo de Colores</h2>
    <form method="get" action="muestra.php">
      R: <input type="text" name="r" size="3" /> <p />
      G: <input type="text" name="g" size="3" /> <p />
      B: <input type="text" name="b" size="3" /> <p />
      <input type="submit" value="Muéstrame" />
    </form>
  </body>
</html>
```

Hasta ahora es muy convencional: un formulario Web con tres campos de datos de entrada, cada uno de ellos asignado a un componente específico de color (rojo [R], verde [G] y azul [B]). Advierte que este formulario envía sus datos a un script PHP llamado *muestra.php*, y utiliza el método GET (para variar).

La figura 2-4 muestra cómo se ve en el explorador Web.

A continuación, crearemos el script PHP que acepta los datos de entrada y los utiliza para mostrar los colores (*muestra.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 2-2: Muestrario HTML Interactivo de Colores</title>
  </head>
  <body>
```

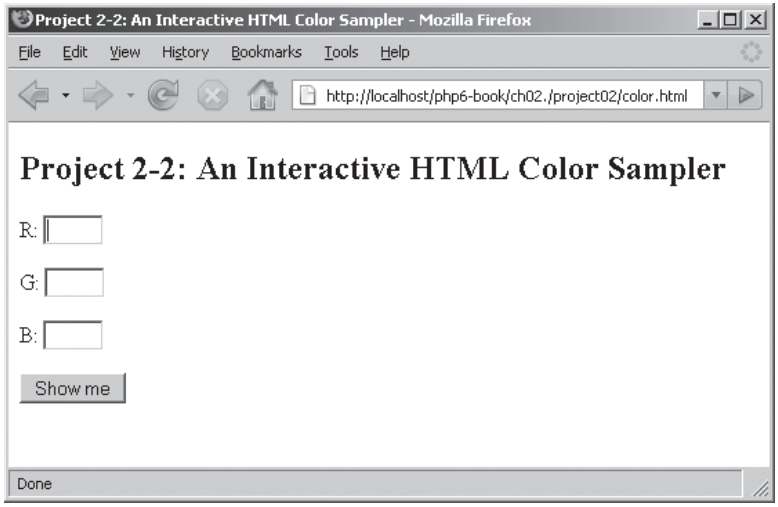


Figura 2-4 Un formulario Web con campos para los valores RGB de color

```
    <h2>Proyecto 2-2: Muestrario HTML Interactivo de Colores</h2>
    <?php
    // obtiene los valores de entrada
    $r = $_GET['r'];
    $g = $_GET['g'];
    $b = $_GET['b'];

    // genera la cadena de caracteres RGB para los datos de entrada
    $rgb = $r . ',' . $g . ',' . $b;
    ?>
      R: <?php echo $r; ?>
      G: <?php echo $g; ?>
      B: <?php echo $b; ?>
    <p />
    <div style="width:150px; height: 150px;
    background-color: rgb(<?php echo $rgb; ?>)" />
  </body>
</html>
```

Los datos insertados en el formulario quedan accesibles a través de la matriz `$_GET` (advierte que es `$_GET` porque el método utilizado en el formulario para enviar los datos fue GET). El script divide esta información en tres variables: `$r`, `$g` y `$b`. Estas variables son unidas en una sola línea utilizando el operador de unión de PHP (¿lo recuerdas?). Esta línea RGB es utilizada después para establecer el color de fondo para un elemento `<div>` en la parte inferior del documento HTML.

(*continúa*)

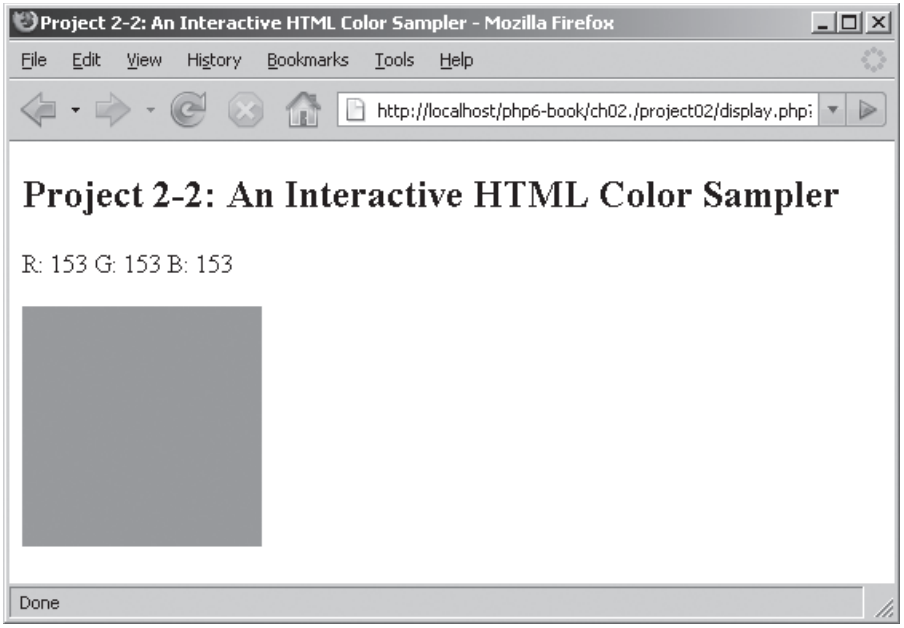


Figura 2-5 La muestra del color que se despliega al enviar el formulario

¿El resultado? Cuando el explorador Web construye la página, verás un bloque de 150 × 150 pixeles relleno con el color sólido correspondiente a los valores RGB seleccionados. Y si regresas al formulario e insertas un nuevo conjunto de código RGB, el color cambiará de acuerdo con esos valores... ¡De ahí lo "interactivo" que ostenta el título del proyecto!

La figura 2-5 muestra lo que sucede cuando se insertan los valores RGB 153, 153, 153 (un tono gris claro, porque este libro está impreso en blanco y negro).

La tabla 2-6 contiene algunas combinaciones de color RGB para que las pruebes.

Color	R	G	B
Rosa	250	75	200
Naranja	255	105	0
Amarillo	255	255	0
Verde	75	200	60
Café	100	75	25

Tabla 2-6 Muestra de combinaciones de color RGB

Resumen

Tal vez te tomó un poco más de tiempo terminar este capítulo en comparación con el anterior, pero al final tus conocimientos sobre la construcción básica de PHP debieron aumentar considerablemente. El capítulo inició presentando variables y constantes PHP, explicando la manera de darles nombre, asignarles valores, utilizarlas en un script y destruirlas cuando ya no se requieren. Te presentó una breve introducción a los tipos de datos simples de PHP y después te llevó en un viaje relámpago por los operadores aritméticos, las cadenas de texto, de comparación y lógicos, utilizando ejemplos comentados para explicar la manera en que cada uno de estos operadores puede ser utilizado para manipular y modificar el contenido de las variables.

Con estas bases bien asentadas, el capítulo prosiguió con una explicación sobre cómo procesar los datos de entrada de un formulario con PHP, tarea común que realizarás una y otra vez a lo largo de este curso y, en realidad, a través de tu carrera como desarrollador de PHP. Por último, dos proyectos mostraron la manera práctica de aplicar estos conocimientos: uno mostró el uso de operadores para realizar cálculos con las variables y el otro mostró lo fácil que es crear una aplicación interactiva PHP utilizando datos de entrada provenientes de un formulario.

El siguiente capítulo estará basado en todo lo que has aprendido hasta ahora, explicará cómo puedes añadir inteligencia a tus scripts PHP a través del uso de pruebas condicionales y te mostrará cómo los constructores de bucles de PHP pueden ayudarte a realizar acciones repetitivas. Hasta entonces, ocupa algo de tiempo revisando los siguientes vínculos Web del manual PHP, que ofrecen información más detallada de los temas abordados en este capítulo:

- Tipos de datos PHP, en **www.php.net/manual/en/language.types.php**
- Malabarismos con los tipos de datos y su conversión en PHP, **www.php.net/manual/en/language.types.type-juggling.php**
- Tablas de comparaciones de tipos de datos PHP, en **www.php.net/manual/en/types.comparisons.php**
- Conocimientos básicos sobre variables, en **www.php.net/manual/en/language.variables.php**
- Operadores PHP y precedencia de operadores, en **www.php.net/manual/en/language.operators.php**
- Accesar datos de un formulario con PHP, en **www.php.net/manual/en/language.variables.external.php**

Autoexamen Capítulo 2

1. La función PHP para detectar el tipo de variables es: _____

2. Identifica cuáles de los siguientes son nombres de variables no válidos: `$24`, `$SOYYO`, `$_error`, `$^b`, `${$var}`, `$yA_K`

3. Escribe una declaración PHP para crear el valor de una constante que almacene el nombre de tu helado favorito.

4. Escribe un script PHP para inicializar una variable y luego incrementar su valor de 3 en 3.

5. Marca como verdaderas o falsas las siguientes declaraciones:

A La función `unset()` borra una variable y la elimina del espacio de variables del programa.

B Las expresiones PHP `$c = ''` y `$c = null` son equivalentes.

C El resultado del cálculo `(56 - 1 * 36 % 7)` es `6`.

D El operador `==` compara el valor y el tipo de variable.

E El operador lógico `O` tiene una precedencia superior que el operador lógico `Y`.

F La función `is_numeric()` regresa el valor true (verdadero) si se aplica a un valor de punto flotante.

G Convertir un número de punto flotante a entero siempre da como resultado un valor redondeado.

H Los elementos tipo `'hidden'` del formulario se excluyen de `$_POST` y `$_GET`.

6. ¿Cuáles son los valores de `$x` y `ABC` al finalizar el siguiente script?

```
<?php
$x = 89;
define ('ABC', $x+1);
$x += ABC;
?>
```

7. ¿Cuáles son los posibles datos de salida del siguiente script PHP?

```
<?php
$boolean = (integer) true;
$numero = 1;
echo (integer) ($boolean === $numero);
?>
```

8. ¿Cuáles son los posibles datos de salida del siguiente script PHP?

```
<?php
define ('NUM', '7');
$a = NUM;
echo gettype ($a);
?>
```

9. Reescribe el código de Prueba esto 2-1, de tal manera que el usuario proporcione la tasa de cambio y la cantidad convertida, a través de un formulario Web.

10. Escribe un script PHP que acepte el valor de la temperatura en grados Celsius (C) mediante un formulario Web y que los convierta a la escala de grados Fahrenheit (F). La fórmula que debe usarse para la conversión es: `F = (9/5) * C + 32`.

11. Escribe un script PHP que muestre los valores insertados en un formulario Web que contenga:

- Un campo de texto.
- Un área de texto.
- Un campo oculto.
- Un campo de contraseña.
- Una lista de selección.
- Dos botones de opción.
- Dos casillas de verificación.

Capítulo 3

Controlar el flujo del programa

Habilidades y conceptos clave

- Aprender a utilizar declaraciones condicionales como `if-else` y `switch-case`
- Automatizar tareas repetitivas con los bucles `while`, `do-while` y `for`
- Ganar experiencia con las funciones numéricas y de cadenas de texto integradas en PHP

Los programas PHP que viste en el capítulo anterior fueron muy convencionales: aceptaban uno o más valores de entrada, realizaban cálculos o comparaciones con ellos y regresaban un resultado. Sin embargo, en la realidad los programas PHP no son tan sencillos: muchos necesitarán tomar decisiones complejas y ejecutar diferentes operaciones durante su ejecución, de acuerdo con las condiciones específicas marcadas por el programador.

En este capítulo aprenderás a crear programas PHP que son más "inteligentes" y pueden realizar diferentes acciones de acuerdo con los resultados obtenidos en pruebas lógicas y comparativas. También aprenderás a volver automáticas acciones repetitivas utilizando bucles y aprenderás más sobre las funciones integradas de PHP para trabajar con cadenas de texto y números. Para asegurar que puedes aplicar estos conocimientos en la realidad, este capítulo también te permite aplicar tus conocimientos recién adquiridos en cuatro proyectos prácticos.

Escribir declaraciones condicionales sencillas

Además de almacenar y recuperar valores en variables, PHP también permite que los programadores evalúen diferentes condiciones durante el curso del programa y que tomen decisiones basándose en el resultado verdadero o falso de la evaluación. Estas condiciones y las acciones asociadas con ellas se expresan mediante un constructor de programación llamado *declaración condicional*. PHP da soporte a diferentes tipos de declaraciones condicionales, cada una de ellas diseñada para un uso específico.

La declaración if

La declaración condicional más sencilla de PHP es `if`. Funciona de manera muy semejante a su correspondiente en el lenguaje común: "**si** (if) sucede X, haz Y". He aquí un sencillo ejemplo; contiene una declaración condicional que verifica si el valor de la variable `$numero` es menor que 0 y envía un mensaje de notificación en caso positivo.

```
<?php
//si el número es menor que cero
//presenta el mensaje
$numero = -88;
if ($numero < 0) {
    echo 'Este número es negativo';
}
?>
```

De esta manera, la clave para la declaración `if` es la condición que habrá de evaluarse, que siempre debe ir encerrada entre paréntesis. Si la condición evaluada es verdadera, se ejecuta el código encerrado entre las llaves ({ }); en caso de que la evaluación resulte falsa, se omite el código entre las llaves. Esta prueba verdadero/falso se realiza utilizando los operadores de comparación PHP, que conociste en el capítulo anterior; la tabla 3-1 hace una recapitulación rápida de ellos.

La declaración if-else

La declaración `if` es muy básica; sólo te permite definir lo que sucede cuando la condición especificada se evalúa como verdadera. Pero PHP también ofrece la declaración `if-else`, una versión mejorada del constructor `if` que te permite definir un conjunto opcional de acciones que el programa debe ejecutar cuando la condición especificada se evalúa como falsa. Por lo general, el uso de esta declaración trae como resultado código más compacto y legible, porque te permite combinar dos acciones en un solo bloque de código unificado. En lenguaje común, esta declaración podría leerse como: "si sucede X, haz Y; de otra manera, haz Z".

Operador	Descripción
==	Igual a
!=	Diferente de
>	Mayor que
>=	Mayor que o igual a
<	Menor que
<=	Menor que o igual a
===	Igual a y del mismo tipo

Tabla 3-1 Operadores de comparación comunes

Como ejemplo, examina esta revisión del ejemplo anterior:

```
<?php
//cambia el mensaje dependiendo de
//si el número es menor que cero o no lo es
$numero = -88;
if ($numero < 0) {
    echo 'Este número es negativo';
} else {
    echo 'Este número es positivo o igual a cero';
}
?>
```

Aquí, la declaración `if-else` se utiliza para manejar dos posibles resultados: un número menor que cero y todos los demás números. Para verlo en acción, intenta ejecutar el script una vez tal y como aparece aquí y vuelve a ejecutarlo después de cambiar el valor de la variable `$numero` por uno positivo.

Pregunta al experto

P: ¿Existe una manera más compacta de escribir la declaración `if-else`?

R: Sí. Requiere un pequeño truco llamado *operador ternario*. Este operador, representado por el signo de cierre de interrogación (`?`), te permite representar la declaración condicional `if-else` en una sola y compacta línea de código. Para verlo en acción, examina los siguientes scripts; ambos son equivalentes:

El bloque if-else estándar	El bloque equivalente utilizando el operador ternario

`<?php`
`if ($X < 10) {`
`  echo 'X es menor que 10';`
`} else {`
`  echo 'x es mayor que 10';`
`}`
`?>`
	`<?php`
	`echo ($X < 10) ? 'X es menor que 10' :`
	`  'X es mayor que 10';`
	`?>`

Aquí, el operador ternario selecciona el código a la izquierda de los dos puntos (`:`) si la evaluación de la condición es verdadera, y el código a la derecha de los dos puntos en caso de que la condición sea falsa.

Prueba esto 3-1 Probar números pares y nones

Ahora que ya conoces las bases de las declaraciones condicionales, veamos un ejemplo de cómo pueden utilizarse. El siguiente programa solicitará al usuario que ingrese un número en un formulario Web, revisará si es un par o non y regresará el mensaje correspondiente.

He aquí el código (*paresnones.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 3-1: Verificador de Números Pares y Nones</title>
  </head>
  <body>
    <h2>Proyecto 3-1: Verificador de Números Pares y Nones</h2>
<?php
    // si el formulario aún no ha sido enviado
    // muestra el formulario
    if (!isset ($_POST['submit'])) {
?>
    <form method="post" action="paresnones.php">
      Ingrese un número: <br />
      <input type="text" name="num" size="3" />
      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
<?php
    // si el formulario ha sido enviado
    // procesa los datos de entrada del formulario
    } else {
      //recupera el número enviado en el formulario
      $num = $_POST['num'];

      // prueba el valor para los números nones
      // muestra el mensaje apropiado
      if (($num % 2) == 0) {
        echo 'Usted ha escrito ' . $num . ', que es un número par.';
      } else {
        echo 'Usted ha escrito ' . $num . ', que es un número non.';
      }
    }
?>
  </body>
</html>
```

(*continúa*)

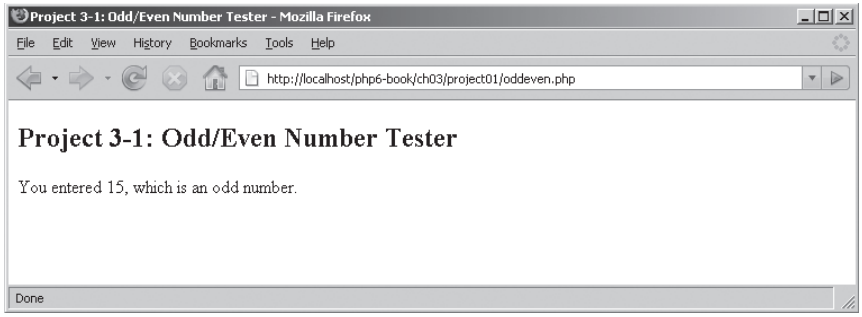


Figura 3-1 Probar números pares y nones con PHP

Este programa consta de dos secciones: la primera mitad genera un formulario Web para que el usuario inserte un número y la segunda averigua si el número es par o non y presenta el mensaje correspondiente. En casi todos los casos, estas dos secciones estarían en archivos separados, pero han sido combinadas en un solo script PHP gracias a la magia de las declaraciones condicionales.

¿Cómo funciona esto? Bien, cuando se envía el formulario Web, la variable `$_POST` contendrá una entrada para el elemento `<input type='submit'...>`. Después, una prueba condicional comprueba la presencia o ausencia de esta variable: en caso de estar ausente, el programa "sabe" que el formulario Web no ha sido enviado todavía y ejecuta el código HTML del formulario; en caso de estar presente, el programa "sabe" que el formulario ha sido enviado y procede a probar el valor enviado.

La prueba para verificar si el valor enviado es un número non es realizada por una segunda declaración condicional `if-else`. Aquí, la expresión condicional consiste en dividir el valor de entrada entre 2 y comprobar que el residuo sea igual a cero. Si la prueba regresa un valor verdadero, se presenta en pantalla el mensaje de los números pares; de lo contrario, se presenta el mensaje correspondiente a los números nones.

La figura 3-1 muestra un ejemplo de los datos de salida.

Escribir declaraciones condicionales más complejas

La declaración `if-else` te permite definir acciones para dos posibilidades: una condición verdadera y una falsa. Sin embargo, es posible que en realidad tu programa tenga que decidir entre más de estos dos sencillos resultados. Para tales situaciones, PHP ofrece dos constructores que permiten al programador manejar múltiples posibilidades: la declaración `if-elseif-else` y la declaración `switch-case`.

La declaración if-elseif-else

La declaración `if-elseif-else` te permite unir varias declaraciones tipo `if-else`, permitiendo que el programador pueda definir acciones para más de dos resultados posibles. Examina el siguiente ejemplo, que ilustra su uso:

```
<?php
// maneja varias posibilidades
// define un mensaje diferente para cada día
$hoy = 'Martes';
if ($hoy == 'Lunes'){
    echo 'El lunes la cara del niño está limpia.';
} elseif ($hoy == 'Martes'){
    echo 'El martes el niño está lleno de gracia.';
} elseif ($hoy == 'Miércoles'){
    echo 'El miércoles el niño está lleno de preocupaciones.';
} elseif ($hoy == 'Jueves'){
    echo 'El jueves el niño se tiene que ir.';
} elseif ($hoy == ‘Viernes’){
    echo 'El viernes el niño es amoroso y dadivoso.';
} elseif ($hoy == 'Sábado'){
    echo 'El sábado el niño trabaja duro.';
} else {
    echo 'No hay información disponible para este día';
}
?>
```

En este caso, el programa mostrará diferentes mensajes para cada día de la semana (dependiendo de lo que se establezca en la variable `$hoy`). Advierte también la última rama `else`: se trata de una rama para “captar el resto”, que será accionada en caso de que ninguna de las declaraciones condicionales anteriores resulten verdaderas; es una manera muy útil para cubrir situaciones imprevisibles.

Hay algo muy importante para recordar sobre el constructor `if-elseif-else`: en cuanto la primera declaración condicional resulte verdadera, PHP ejecutará el código correspondiente, se saltará el resto de las pruebas condicionales e irá directo a las líneas posteriores del bloque entero `if-elseif-else`. Así pues, aunque más de una declaración sea verdadera, PHP sólo ejecutará el código correspondiente a la primera que sea evaluada de manera positiva.

La declaración switch-case

Una opción a `if-elseif-else` es `switch-case`, que realiza casi la misma acción: coteja una variable contra una serie de valores hasta que localiza una coincidencia, para luego

ejecutar el código correspondiente a la misma. Examina el siguiente código, equivalente al ejemplo inmediato anterior:

```
<?php
// maneja varias posibilidades
// define un mensaje diferente para cada día
$hoy = 'Martes';
switch ($hoy){
    case 'Lunes':
        echo 'El lunes la cara del niño está limpia.';
        break;
    case 'Martes':
        echo 'El martes el niño está lleno de gracia.';
        break;
    case 'Miércoles':
        echo 'El miércoles el niño está lleno de preocupaciones.';
        break;
    case 'Jueves':
        echo 'El jueves el niño se tiene que ir.';
        break;
    case 'Viernes':
        echo 'El viernes el niño es amoroso y dadivoso.';
        break;
    case 'Sábado':
        echo 'El sábado el niño trabaja duro.';
        break;
    default:
        echo 'No hay información disponible para este día';
        break;
}
?>
```

El constructor `switch-case` presenta una importante diferencia en comparación con el constructor `if-elseif-else`: una vez que PHP encuentra una declaración `case` que es evaluada como verdadera, ejecuta no sólo el código correspondiente, sino todo el código de las demás declaraciones `case`. Si esto no es lo que quieres, debes añadir la declaración `break` al final de cada bloque `case` (como se hizo en el ejemplo anterior), para indicar a PHP que termine la ejecución del constructor una vez que haya ejecutado el código correspondiente al primer caso evaluado como verdadero.

Advierte también el caso `'default'`: como su nombre lo sugiere, especifica el conjunto de acciones por defecto que PHP debe tomar cuando ninguno de los anteriores casos se evalúe como verdadero. Este caso por defecto, al igual que la rama `else` del bloque `if-elseif-else`, es útil para "captar el resto" en situaciones imprevistas.

Prueba esto 3-2 Asignar niños exploradores a su tienda de campaña

Ahora utilizaremos la declaración `if-elseif-else` para crear una pequeña aplicación para los líderes de niños exploradores de cualquier parte: una herramienta Web que asigna automáticamente la tienda de campaña correcta a cada niño explorador durante el campamento, basándose en su edad. La aplicación muestra a los exploradores un formulario Web en el cual pueden escribir su edad y luego les asigna una de cuatro tiendas de campaña: Roja, Verde, Azul y Negra, que compartirán con otros exploradores de casi la misma edad.

He aquí el código (*tienda.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 3-2: Asignación de tiendas de campaña</title>
  </head>
  <body>
    <h2>Proyecto 3-2: Asignación de tiendas de campaña</h2>
<?php
    // si el formulario no ha sido enviado
    // muestra el formulario
    if (!isset($_POST['submit'])) {
?>
    <form method="post" action="tiendas.php">
      Escribe tu edad: <br />
      <input type="text" name="age" size="3" />
      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
<?php
    // si el formulario ha sido enviado
    // procesa los datos ingresados
    } else {
      // recupera la edad enviada por el método POST
      $edad = $_POST['age'];

      // asigna a una de cuatro tiendas
      // de acuerdo con el "binario" de edad al que pertenece
      if ($edad <= 9){
          echo "Estás en la tienda de campaña Roja.";
      } elseif ($edad > 9 && $edad <= 11) {
          echo "Estás en la tienda de campaña Azul.";
      } elseif ($edad > 11 && $edad <= 14) {
          echo "Estás en la tienda de campaña Verde.";
```

(*continúa*)

Figura 3-2 La página resultante, que asigna al usuario una tienda de campaña

```
        } elseif ($edad > 14 && $edad <= 17) {
            echo "Estás en la tienda de campaña Negra.";
        } else {
            echo "Mejor comunícate con el líder de los niños exploradores.";
        }
    }
?>
  </body>
</html>
```

Como el proyecto anterior de este mismo capítulo, éste también combina el formulario Web y su página resultante en un solo script, separado por la declaración condicional `if-else`. Una vez que el explorador ingresa su edad en el formulario Web y lo envía, un bloque `if-elseif-else` se encarga de definir cuatro rangos de edad, prueba la edad enviada contra estos rangos y decide cuál tienda de campaña es más apropiada para el explorador. Los rangos de edad son 0-9 (tienda de campaña Roja); 10-11 (tienda Azul); 12-14 (Verde); 15-17 (Negra). Los exploradores mayores de 17 años reciben un mensaje que les indica comunicarse con el líder para arreglar su acomodo.

La figura 3-2 muestra la página resultante.

Combinar declaraciones condicionales

PHP permite que una declaración condicional se anide dentro de otra, con el fin de tomar decisiones más complejas. Para ilustrarlo, examina el siguiente código:

```
<?php
// los empleados con ingresos mensuales <= 15000
// y con un rendimiento >= 3 reciben un bono de $5000
```

```
// el resto recibe un bono de $3000
if ($rendimiento >= 3) {
    if ($salario < 15000) {
        $bono = 5000;
    }
} else {
    if ($salario > 15000){
        $bono = 3000;
    }
}
?>
```

También puedes combinar declaraciones condicionales utilizando operadores lógicos, como && o ||. Los estudiaste en el capítulo anterior; la tabla 3-2 hace una recapitulación rápida de ellos.

He aquí un ejemplo de la manera en que estos operadores pueden utilizarse con una declaración condicional:

```
<?php
$fecha = 2008;
// los años bisiestos son divisibles entre 400
// o entre 4, pero no entre 100
if (($fecha % 400 == 0) || (($fecha % 100 != 0) && ($fecha % 4 == 0))) {
    echo "$fecha es año bisiesto.";
} else {
    echo "$fecha no es año bisiesto.";
}
?>
```

Repetir acciones con bucles

Como cualquier buen lenguaje de programación, PHP da soporte a *bucles*: en esencia, con este nombre se designa la capacidad de repetir una serie de acciones hasta que se cumple una condición especificada. Los bucles son una herramienta importante que ayuda a volver automáticas tareas repetitivas dentro del programa. PHP da soporte a cuatro tipos diferentes

Operador	Descripción
&&	Y (AND)
||	O (OR)
!	NO (NOT)

Tabla 3-2 Operadores lógicos comunes

de bucles, tres de los cuales son presentados en la siguiente sección (el cuarto se explica en el siguiente capítulo).

El bucle while

El bucle más sencillo de entender es el `while`, que se repite continuamente mientras una condición específica sea verdadera. A continuación presentamos un ejemplo, que utiliza un bucle para escribir una `'x'` de manera repetida en la página de salida.

```
<?php
// repite constantemente hasta que el contador llega a 10
// datos de salida: 'xxxxxxxxx'
$contador = 1;
while ($contador < 10) {
  echo 'x';
  $contador++;
}
?>
```

Observa la condición encerrada entre paréntesis: mientras sea verdadera, se ejecutará el código entre las llaves. En cuanto la condición sea falsa, el bucle se detiene y las líneas que le siguen se ejecutan de la manera usual.

El bucle do-while

Con el bucle `while`, la condición que habrá de evaluarse se prueba al principio de cada repetición del bucle. Existe una variante, el bucle `do-while`, que evalúa la condición al final de cada repetición. He aquí una revisión del ejemplo anterior para ilustrar lo dicho:

```
<?php
// repite constantemente hasta que el contador llega a 10
// datos de salida: 'xxxxxxxxx'
$contador = 1;
do {
  echo 'x';
  $contador++;
} while ($contador < 10);
?>
```

La diferencia en la estructura también debe ser evidente: con un bucle `do-while`, la condición que habrá de evaluarse ahora aparece en la parte inferior del bloque de código, en lugar de hacerlo al principio.

NOTA

Las diferencias en el comportamiento entre un bucle `while` y uno `do-while` tienen una importante implicación: en un bucle `while`, si la expresión condicional es evaluada como falsa en la primera pasada, el bucle nunca se ejecutará. En cambio, el bucle `do-while` siempre se ejecutará por lo menos una vez, aunque la expresión condicional sea falsa, porque la condición se evalúa al final de la repetición del bucle y no al principio.

El bucle for

Los bucles `while` y `do-while` son muy sencillos: se repiten mientras la condición especificada sea verdadera. Pero PHP también da soporte a un tipo de bucle más complejo: `for`, que es útil cuando necesitas ejecutar un conjunto de declaraciones un número determinado de veces.

La mejor manera de comprender el bucle `for` es analizando el código donde aparece. He aquí un ejemplo, que lista los números del 1 al 10:

```
<?php
// repite constantemente hasta que el contador llega a 10
// datos de salida:
for ($x=1; $x<10; $x++) {
  echo "$x ";
}
?>
```

En este listado, el bucle comienza asignando a la variable contador `$x` el valor 1; después ejecuta la declaración que conforma el bucle. Una vez que alcanza el final de la primera repetición, actualiza el contador añadiéndole 1, revisa la expresión condicional para asegurarse de que aún no ha alcanzado el valor predeterminado (10), y ejecuta el bucle una vez más. El proceso continúa hasta que el contador alcanza el valor de 10 y la expresión condicional se vuelve falsa.

Como se aprecia en el ejemplo anterior, un bucle `for` típico requiere el uso de tres expresiones, encerradas entre paréntesis y separadas por puntos y comas:

- La primera es una expresión de asignación, que inicializa el bucle con cierto valor; en este caso le asigna el valor 1 a la variable `$x`.
- La segunda es una expresión condicional, que debe evaluarse como verdadera o falsa; el bucle seguirá ejecutándose mientras esta condición sea verdadera. Una vez que se vuelva falsa, el bucle dejará de ejecutarse.
- La tercera es, de nuevo, una expresión de asignación, que se ejecuta al final de cada repetición del bucle, y que actualiza el contador con un nuevo valor; en este caso añade 1 al valor de `$x`.

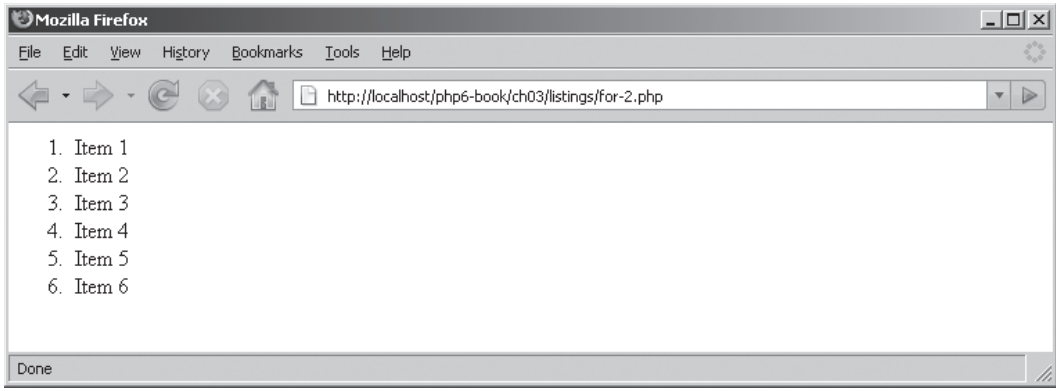


Figura 3-3 Una lista generada dinámicamente

He aquí otro ejemplo; éste presenta el uso del bucle `for` para generar una lista ordenada en HTML:

```
<?php
// genera una lista ordenada con 6 elementos
echo "<ol>";
for ($x=1; $x<7; $x++) {
  echo "<li>Elemento $x</li>";
}
echo "</ol>";
?>
```

La figura 3-3 muestra cómo se presentan los datos de salida.

Combinar bucles

Al igual que con las declaraciones condicionales, también es posible anidar un bucle dentro de otro. Para ilustrar esto, examina el siguiente ejemplo, que anida un bucle `for` dentro de otro para generar dinámicamente una tabla HTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title></title>
  </head>
  <body>
<?php
// genera una tabla HTML
// 3 filas, 4 columnas
echo "<table border=\"1\">";
```

```
for ($fila=1; $fila<4; $fila++) {
  echo "<tr>";
  for ($col=1; $col<5; $col++) {
    echo "<td>Fila $fila, Columna $col</td>";
  }
  echo "</tr>";
}
echo "</table>";
?>
  </body>
</html>
```

Este script utiliza dos bucles `for` anidados. El bucle externo es el responsable de generar las filas de la tabla y se ejecuta tres veces. En cada repetición de este bucle externo, también se dispara el bucle interno, que es responsable de generar las celdas dentro de cada fila (equivalente a las columnas) y se ejecuta cuatro veces. El resultado final es una tabla con tres filas, cada una de ellas con cuatro celdas. La figura 3-4 muestra el resultado.

Interrumpir y omitir bucles

Ya que estamos en el tema de los bucles, es interesante tratar dos declaraciones que te permiten interrumpir u omitir una iteración particular de un bucle. La declaración PHP `break` permite romper un bucle en cualquier punto. Para ilustrarlo, examina el siguiente bucle que normalmente se repetiría cinco veces, pero que se detiene después de la segunda iteración debido a la declaración `break`:

```
<?php
$cuenta = 0;
// recorre el bucle 5 veces
```

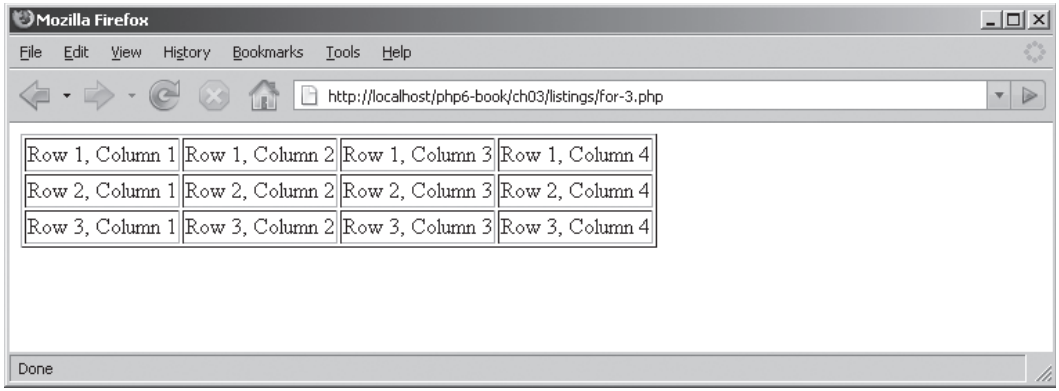


Figura 3-4 Tabla generada dinámicamente

```
while ($cuenta <= 4) {
    $cuenta++;
    // cuando el contador llega a 3
    // rompe el bucle
    if ($cuenta == 3) {
        break;
    }
    echo "Esta es la iteración #$cuenta <br/>";
}
?>
```

A diferencia de `break`, `continue` no detiene el procesamiento del bucle; simplemente "salta una" iteración. Para ver cómo trabaja esto, considere el siguiente bucle, que se itera cinco veces pero omite la tercera iteración debido a la intervención de `continue`:

```
<?php
$cuenta = 0;
// recorre el bucle 5 veces
while ($cuenta <= 4) {
    $cuenta++;
    // cuando el contador llega a 3
    // omite la siguiente iteración
    if ($cuenta == 3) {
        continue;
    }
    echo "Esta es la iteración #$cuenta <br/>";
}
?>
```

Prueba esto 3-3 Construir una calculadora factorial

Una aplicación sencilla y real que puedes probar por ti mismo para comprender mejor la manera en que funcionan los bucles, es una calculadora factorial. En caso de que te hayas dormido en la clase de matemáticas el día que se explicó, el factorial de un número *n* es simplemente el producto de la multiplicación de todos los números entre *n* y 1. Así, por ejemplo, el factorial de 4 es 4 * 3 * 2 * 1 = 24.

La calculadora de factoriales que construirás en esta sección es interactiva: solicita que el usuario escriba un número en un formulario Web y luego calcula el factorial de ese número. He aquí el código (*factorial.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 3-3: Calculadora Factorial</title>
```

```
  </head>
  <body>
    <h2>Proyecto 3-3: Calculadora Factorial</h2>
<?php
    // si el formulario no se ha enviado
    // presenta el formulario
    if (!isset ($_POST['submit'])) {
?>
    <form method="post" action="factorial.php">
      Ingresa un número: <br />
      <input type="text" name="num" size="3" />
      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
<?php
    // si el formulario ya se ha enviado
    // procesa los datos de entrada
    } else {
      // recupera el número del formulario
      $num = $_POST['num'];

      // verificar que el número sea positivo
      if ($num <= 0) {
        echo 'Error: por favor ingrese un número superior a 0';
        exit();
      }

      // calcula el factorial
      // multiplicando el número por
      // todos los números entre él y el 1
      $factorial = 1;
      for ($x=$num; $x>=1; $x--){
        $factorial *= $x;
      }
      echo "El factorial de $num es: $factorial";
    }
?>
  </body>
</html>
```

La mayor parte del trabajo es realizada por el bucle `for`. Este bucle contador se inicia con el número ingresado por el usuario en el formulario Web y luego comienza a contar hacia atrás, disminuyendo el contador a razón de 1 en cada iteración. Cada vez que el bucle se ejecuta, el producto previo calculado se multiplica por el valor actual que contiene el contador. El resultado final es el factorial del número ingresado por el usuario.

La figura 3-5 muestra el resultado después de enviar el formulario.

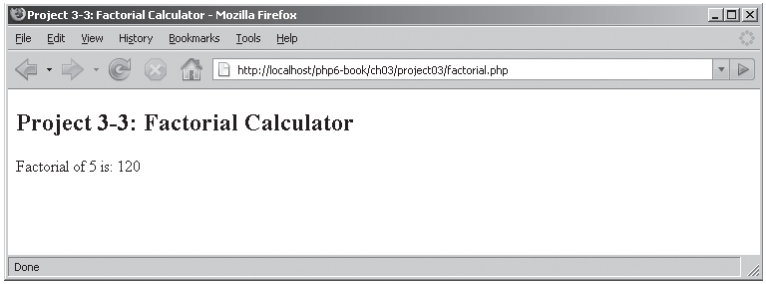


Figura 3-5 La página de salida, que muestra el factorial del número

Trabajar con funciones de cadenas de texto y numéricas

En el capítulo anterior aprendiste un poco sobre los tipos de datos de PHP, incluidos sus operadores para manipular cadenas de texto y números. Pero PHP te permite hacer mucho más que una simple unión de caracteres y sumar valores; el lenguaje incluye una biblioteca completa de funciones integradas que te permite hacer casi cualquier cosa, desde dividir cadenas de texto e invertir el orden de las letras hasta calcular valores logarítmicos. Este capítulo presenta una introducción a algunas de esas funciones.

Utilizar funciones de cadena de texto

PHP tiene más de 75 funciones integradas para la manipulación de texto, que dan soporte a operaciones que van desde repetición e inversión de cadenas de texto hasta comparaciones y operaciones de búsqueda y sustitución. La tabla 3-3 presenta una lista con algunas de estas funciones.

Buscar cadenas de texto vacías

La función `empty()` regresa un valor verdadero si una variable de cadena de texto está "vacía". Las variables de cadena de texto vacías son las que tienen valores `''`, `0`, `'0'` o `NULL`. La función `empty()` también regresa un valor verdadero cuando se utiliza con una variable inexistente. A continuación unos ejemplos:

```
<?php
// prueba si la cadena de caracteres está vacía
// datos de salida: true
$cadena = '';
echo (boolean) empty($cadena);

// datos de salida: true
$cadena = null;
```

```
echo (boolean) empty($cadena);
// datos de salida: true
$cadena = '0';
echo (boolean) empty($cadena);

// datos de salida: true
unset($cadena);
echo (boolean) empty($cadena);
?>
```

Función	Lo que hace
empty()	Prueba si una cadena de caracteres está vacía
strlen()	Calcula la cantidad de caracteres en una cadena
strrev()	Invierte una cadena de caracteres
str_repeat()	Repite una cadena de caracteres
substr()	Recupera una sección de la cadena de caracteres
strcmp()	Compara dos cadenas de caracteres
str_word_count()	Calcula la cantidad de palabras en una cadena de caracteres
str_replace()	Reemplaza partes de la cadena de caracteres
trim()	Elimina espacios vacíos al principio y al final de una cadena de caracteres
strtolower()	Convierte a minúsculas los caracteres de una cadena
strtoupper()	Convierte a mayúsculas los caracteres de una cadena
ucfirst()	Convierte a mayúsculas el primer carácter de una cadena
ucwords()	Convierte a mayúsculas el primer carácter de cada palabra en una cadena
addslashes()	Crea caracteres de escape especiales en una cadena con diagonales invertidas
stripslashes()	Elimina diagonales invertidas de una cadena de caracteres
htmlentities()	Codifica en HTML dentro de una cadena de caracteres
htmlspecialchars()	Codifica caracteres especiales HTML dentro de una cadena
nl2br()	Reemplaza saltos de línea con elementos
html_entity_decode()	Decodifica entidades HTML dentro de una cadena de caracteres
htmlspecialchars_decode()	Decodifica caracteres especiales HTML dentro de una cadena
strip_tags()	Elimina el código PHP y HTML de una cadena de caracteres

Tabla 3-3 Funciones comunes de cadena de caracteres en PHP

Invertir y repetir cadenas de caracteres

La función `strlen()` regresa la cantidad de caracteres en una cadena. He aquí un ejemplo de ésta en acción:

```
<?php
// calcula la longitud de una cadena de caracteres
// datos de salida: 17
$cadena = 'Bienvenido a Xanadu';
echo strlen($cadena);
?>
```

Invertir una cadena de caracteres es tan sencillo como invocar la función `strrev()`, como en el siguiente ejemplo:

```
<?php
// invertir cadena de caracteres
// datos de salida: 'osap oñeuqep nU'
$cadena = 'Un pequeño paso';
echo strrev($cadena);
?>
```

En caso de que necesites repetir una cadena de caracteres, PHP ofrece la función `str_repeat`, que acepta dos argumentos: la cadena de caracteres y la cantidad de veces que será repetida. He aquí un ejemplo:

```
<?php
// repetir cadena de caracteres
// datos de salida: 'yoyoyo'
$cadena = 'yo';
echo str_repeat ($cadena, 3);
?>
```

Trabajar con subcadenas de caracteres

PHP también te permite dividir una cadena de caracteres en partes más pequeñas con la función `substr()`, que acepta tres argumentos: la cadena de caracteres original, la posición (*offset*) donde debe comenzar la división y el número de caracteres que debe mostrar a partir de la posición donde comienza. El siguiente ejemplo lo ilustra:

```
<?php
// extrae subcadena
// datos de salida: 'venido'
$cadena = 'Bienvenido a ninguna parte';
echo substr($cadena, 3, 4);
?>
```

NOTA

Cuando utilices la función `substr()`, el primer carácter de la cadena se considera como offset0, el segundo como offset1, etcétera.

Para extraer una subcadena desde el final de la cadena original (en lugar de hacerlo desde el principio), pasa un valor negativo de offset en `substr()`, como se muestra en la siguiente revisión del ejemplo anterior:

```
<?php
// extrae una subcadena
// datos de salida: 'un arte'
$cadena = 'Bienvenido a ninguna parte';
echo substr($cadena, 3, 5) . substr($cadena, -4, 4);
?>
```

Pregunta al experto

P: ¿Puedo recuperar un solo carácter de una cadena? ¿Cómo?

R: Existen dos maneras de recuperar un solo carácter de una cadena. El camino largo utiliza la función `substr()`, como se muestra en el siguiente ejemplo:

```
<?php
// datos de salida: 'r'
$cadena = 'abracadabra';
echo substr($cadena, 2, 1);
?>
```

La función `substr()` acepta tres argumentos: la cadena de caracteres original, la posición donde debe comenzar la extracción y el número de caracteres a extraer comenzando por la posición especificada. Ten en mente que la cuenta de la posición comienza en 0 y no en 1.

De cualquier manera, PHP también da soporte a una abreviatura de sintaxis para completar la misma tarea. Consiste en encerrar entre llaves la posición específica del carácter que quieres recuperar; las llaves se colocan justo después del nombre de la variable. El siguiente ejemplo lo ilustra:

```
<?php
// datos de salida: 'r'
$cadena = 'abracadabra';
echo $cadena{2};
?>
```

Comparar, contar y reemplazar cadenas de caracteres

Si necesitas comparar dos cadenas de caracteres, la función `strcmp()` realiza una comparación sensible a mayúsculas, regresa un valor negativo en caso de que la primera sea "menor" que la segunda; en caso contrario regresa un valor positivo y regresa un cero si ambas cadenas son "iguales". He aquí unos ejemplos de cómo funciona:

```
<?php
// compara cadenas de caracteres
$a ='hola';
$b ='hola';
$c ='hOla';

// datos de salida: 0
echo strcmp($a, $b);

// datos de salida: 1
echo strcmp($a, $c);
?>
```

La función PHP `str_word_count()` proporciona una manera fácil de contar el número de palabras en una cadena de caracteres. El siguiente ejemplo ilustra su uso:

```
<?php
// cuenta palabras
// datos de salida: 6
$cadena = "Mi nombre es Bond, James Bond";
echo str_word_count($cadena);
?>
```

Si necesitas sustituir caracteres dentro de una cadena, PHP también cuenta con la función `str_replace()`, diseñada especialmente para realizar operaciones de búsqueda y sustitución. Esta función acepta tres argumentos: el término que se busca, el término que lo reemplaza y la cadena de caracteres donde se debe realizar la sustitución. He aquí un ejemplo:

```
<?php
// reemplaza '@' por 'arroba'
// datos de salida: 'juan arroba dominio.net'
$cadena = 'juan@dominio.net';
echo str_replace('@', ' arroba ', $cadena);
?>
```

Dar formato a cadenas de caracteres

La función PHP `trim()` puede utilizarse para eliminar espacios en blanco al principio o al final de una cadena de caracteres; esto es muy útil cuando se procesan datos de entrada provenientes de un formulario Web. He aquí un ejemplo:

```
<?php
// eliminar espacios en blanco al principio y al final
// datos de salida: 'a b   c'
$cadena = '   a b   c   ';
echo trim($cadena);
?>
```

Cambiar a mayúsculas o minúsculas los caracteres de una cadena es tan sencillo como invocar las funciones `strtolower()` para minúsculas y `strtoupper()` para mayúsculas, como se muestra en los siguientes ejemplos:

```
<?php
// cambia las mayúsculas por minúsculas
$cadena = 'Yabba Dabba Doo';

// datos de salida: 'yabba dabba doo'
echo strtolower($cadena);

// datos de salida: 'YABBA DABBA DOO'
echo strtoupper($cadena);
?>
```

También puedes cambiar a mayúsculas el primer carácter de una cadena con la función `ucfirst()`, o bien el primer carácter de cada palabra de la cadena con la función `ucwords()`. El siguiente ejemplo muestra ambas funciones:

```
<?php
// cambia mayúsculas/minúsculas
$cadena = 'las brigadas amarillas';

// datos de salida: 'Las Brigadas Amarillas'
echo ucwords($cadena);

// datos de salida: 'Las brigadas amarillas'
echo ucfirst($cadena);
?>
```

Trabajar con cadenas de caracteres HTML

PHP cuenta con funciones exclusivas para trabajar con cadenas de caracteres HTML. Antes que nada, está la función `addslashes()`, la cual busca signos especiales y los convierte automáticamente en caracteres de escape, añadiéndoles una diagonal invertida. He aquí un ejemplo:

```
<?php
// convierte signos especiales en caracteres de escape
// datos de salida: las décadas de los 80\'s y 90\'s
$cadena = "las décadas de los 80's y 90's";
echo addslashes($cadena);
?>
```

Puedes revertir el trabajo hecho por `addslashes()` con la función `stripslashes()`, que elimina todas las diagonales invertidas de una cadena de caracteres. Examina el siguiente ejemplo que ilustra lo escrito:

```
<?php
// elimina las diagonales invertidas
// datos de salida: 'Juan D'Souza dijo "Nos vemos luego".'
$cadena = "Juan D\'Souza dijo \"Nos vemos luego\".";
echo stripslashes($cadena);
?>
```

Las funciones `htmlentities()` y `htmlspecialchars()` convierten automáticamente los símbolos especiales HTML (como < y >) en su correspondiente código (`<` y `&<hairline #>gt;`). De manera similar, la función `nl2br()` reemplaza automáticamente los caracteres correspondientes a saltos de línea en una cadena con la etiqueta HTML correspondiente: `<br />`. He aquí un ejemplo:

```
<?php
// reemplazar con entidades HTML
// datos de salida: '&lt;div width=&quot;200&quot;&gt;
//                   Esta es una etiqueta div&lt;/div&gt;'
$html = '<div width="200">Esta es una etiqueta div</div>';
echo htmlentities($html);

// reemplaza saltos de línea con elementos <br />
// datos de salida: 'Esta es una línea ro<br>
//                     ta'
$lineas = 'Esta es una línea ro
             ta';
echo nl2br($lineas);
?>
```

Puedes revertir los efectos de las funciones `htmlentities()` y `htmlspecialchars()` con las funciones `html_entity_decode()` y `html_specialchars_decode()`, respectivamente.

Por último, la función `strip_tags()` busca todo el código HTML y PHP dentro de una cadena de caracteres y lo elimina para generar una cadena "limpia". He aquí un ejemplo:

```
<?php
// elimina las etiquetas HTML de una cadena de caracteres
// datos de salida: 'Por favor inicie sesión de nuevo'
$html = '<div width="200">Por favor <strong>inicie sesión de nuevo</
strong></div>';
echo strip_tags($html);
?>
```

Utilizar funciones numéricas

No pienses que el poder de PHP se limita a las cadenas de texto: el lenguaje cuenta con más de 50 funciones integradas para trabajar con números, desde sencillas funciones de formato hasta funciones aritméticas, logarítmicas y manipulaciones trigonométricas. La tabla 3-4 lista algunas de estas funciones.

Hacer cálculos

Una tarea común cuando se trabaja con números es redondearlos hacia arriba o hacia abajo. PHP ofrece las funciones `ceil()` y `floor()` para realizar estas tareas, respectivamente, y se muestran en el siguiente ejemplo:

```
<?php
// redondea números
// datos de salida: 19
$num = 19.7;
echo floor($num);
```

Función	Lo que hace
`ceil()`	Redondea un número hacia arriba
`floor()`	Redondea un número hacia abajo
`abs()`	Encuentra el valor absoluto de un número
`pow()`	Eleva un número a la potencia de otro
`log()`	Encuentra el logaritmo de un número
`exp()`	Encuentra el exponente de un número
`rand()`	Genera un número aleatorio
`bindec()`	Convierte un número de binario a decimal
`decbin()`	Convierte un número de decimal a binario
`decoct()`	Convierte un número de decimal a octal
`dechex()`	Convierte un número de decimal a hexadecimal
`hexdec()`	Convierte un número de hexadecimal a decimal
`octdec()`	Convierte un número de octal a decimal
`number_format()`	Forma un número con miles y decimales agrupados
`printf()`	Forma un número utilizando especificaciones personalizadas

Tabla 3-4 Funciones numéricas PHP comunes

```
// redondea un número hacia arriba
// datos de salida: 20
echo ceil($num);
?>
```

También tenemos la función `abs()`, que regresa el valor absoluto de un número. He aquí un ejemplo:

```
<?php
// regresa el valor absoluto de un número
// datos de salida: 19.7
$num = -19.7;
echo abs($num);
?>
```

La función `pow()` regresa el valor de un número elevado a la potencia de otro:

```
<?php
// calcula 4 ^ 3
// datos de salida: 64
echo pow(4,3);
?>
```

La función `log()` calcula el logaritmo natural o base 10 de un número, mientras que la función `exp()` calcula el exponente de un número. He aquí un ejemplo de ambas en acción:

```
<?php
// calcula el logaritmo natural de 100
// datos de salida: 2.30258509299
echo log(10);

// calcula el logaritmo de 100, base 10
// datos de salida: 2
echo log(100,10);

// calcula el exponente de 2.30258509299
// datos de salida: 9.99999999996
echo exp(2.30258509299);
?>
```

Generar números aleatorios

Generar números aleatorios con PHP es también una tarea sencilla: la función integrada al lenguaje, `rand()`, genera automáticamente un entero aleatorio mayor que 0. Puedes restringir la generación a cierto rango de números proporcionando límites opcionales como argumentos. El siguiente ejemplo lo ilustra:

```
<?php
// genera un número aleatorio
```

```
echo rand();

// genera un número aleatorio entre 10 y 99
echo rand(10,99);
?>
```

Convertir entre bases numéricas

PHP cuenta con funciones integradas para hacer conversiones entre diferentes bases numéricas: binario, decimal, octal y hexadecimal. He aquí un ejemplo que muestra en acción las funciones `bindec()`, `decbin()`, `decoct()`, `dechex()`, `hexdec()` y `octdec()`:

```
<?php
// convertir de decimal a binario
// datos de salida: 1000
echo decbin(8);

// convertir de decimal a hexadecimal
// datos de salida: 8
echo dechex(8);

// convertir de decimal a octal
// datos de salida: 10
echo decoct(8);

// convertir de octal a decimal
// datos de salida: 8
echo octdec(10);

// convertir de hexadecimal a decimal
// datos de salida: 101
echo hexdec(65);

// convertir de binario a decimal
// datos de salida: 8
echo bindec(1000);
?>
```

Formar números

Cuando se trata de formar números, PHP ofrece la función `number_format()`, que acepta cuatro argumentos: el número que se formará, la cantidad de espacios decimales que se deben presentar, el carácter que se utilizará en lugar del punto decimal y el carácter que se empleará para separar los millares agrupados (por lo general se utiliza una coma). Examina el siguiente ejemplo que ilustra lo escrito:

```
<?php
// formar un número (con valores por defecto)
```

```
// datos de salida: 1,106,483
$num = 1106482.5843;
echo number_format($num);

// formar un número (con separadores personalizados)
// datos de salida: 1?106?482*584
echo number_format($num, 3, '*', '?');
?>
```

Para tener mayor control sobre el formato de los números, PHP ofrece las funciones `printf()` y `sprintf()`. Estas funciones, aunque muy útiles, pueden intimidar a los usuarios novatos; por ello, la mejor manera de entenderlas es mediante ejemplos. Examina el siguiente código, que los muestra en acción:

```
<?php
// dar formato de número decimal
// datos de salida: 00065
printf("%05d", 65);

// dar formato de número de punto flotante
// datos de salida: 00239.000
printf("%09.3f", 239);

// dar formato de número octal
// datos de salida:  10
printf("%4o", 8);

// formar un número
// incorporado a una cadena de caracteres
// datos de salida: 'Veo 8 manzanas y 26.00 naranjas'
printf("Veo %4d manzanas y %4.2f naranjas", 8, 26);
?>
```

Ambas funciones aceptan dos argumentos: una serie de *especificadores de formato* y la cadena o el número sin trabajar que va a recibir el formato. Los datos de entrada se forman de acuerdo con las especificaciones anotadas y el resultado se despliega directamente con la función `printf()` o asignándolo a una variable con la función `sprintf()`.

Algunos de los especificadores de formato más comunes se muestran en la tabla 3-5.

También puedes combinar éstos con *especificadores de precisión*, que indican la cantidad de dígitos que habrá de mostrarse para los valores de punto flotante, por ejemplo, `%1.2f` implica que el número debe formarse como valor de punto flotante con dos dígitos desplegados después del punto decimal. En el caso de números más pequeños, es posible añadir también un *especificador de relleno*, que indica a la función que rellene los números hasta cierta longitud específica utilizando un carácter personalizado. Puedes ver estos dos tipos de especificadores en el ejemplo anterior.

Especificador	Lo que significa
`%s`	Cadena de caracteres
`%d`	Número decimal
`%x`	Número hexadecimal
`%o`	Número octal
`%f`	Número de punto flotante

Tabla 3-5 Especificadores de formato para las funciones `printf()` y `sprintf()`

Prueba esto 3-4 Procesar un formulario para registro de miembros

Ahora que ya conoces algunas funciones PHP integradas, utilicemos algo de lo que has aprendido en una aplicación práctica: un formulario de solicitud de membresías para un club deportivo. Este formulario solicitará al aplicante que ingrese información personal; después validará esta información y, de ser aceptable, formulará y enviará un correo electrónico con la información del solicitante al administrador del club.

He aquí el formulario HTML (*registro.html*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 3-4: Registro de Miembros</title>
  </head>
  <body>
    <h2> Proyecto 3-4: Registro de Miembros</h2>
    <form method="POST" action="registro.php">
      Nombre: <br />
      <input type="text" name="nombre" size="50" />
      <p>
      Dirección: <br />
      <textarea name="direccion" rows="5" cols="40"></textarea>
      <p>
      Edad: <br />
      <input type="text" name="edad" size="3" />
      <p>
      Profesión: <br />
      <input type="text" name="profesion" size="50" />
```

(*continúa*)

```
      <p>
      Estatus residencial: <br />
      <input type="radio" name="residencia" value="yes" checked="true" />
Residente
      <input type="radio" name="residencia" value="no" /> No Residente
      <p>
      <input type="submit" name=»submit" value="Enviar" />
    </form>
  </body>
</html>
```

Este formulario tiene cinco campos de entrada: nombre del solicitante, dirección, edad, profesión y estatus residencial. La figura 3-6 muestra cómo se ve en el explorador Web.

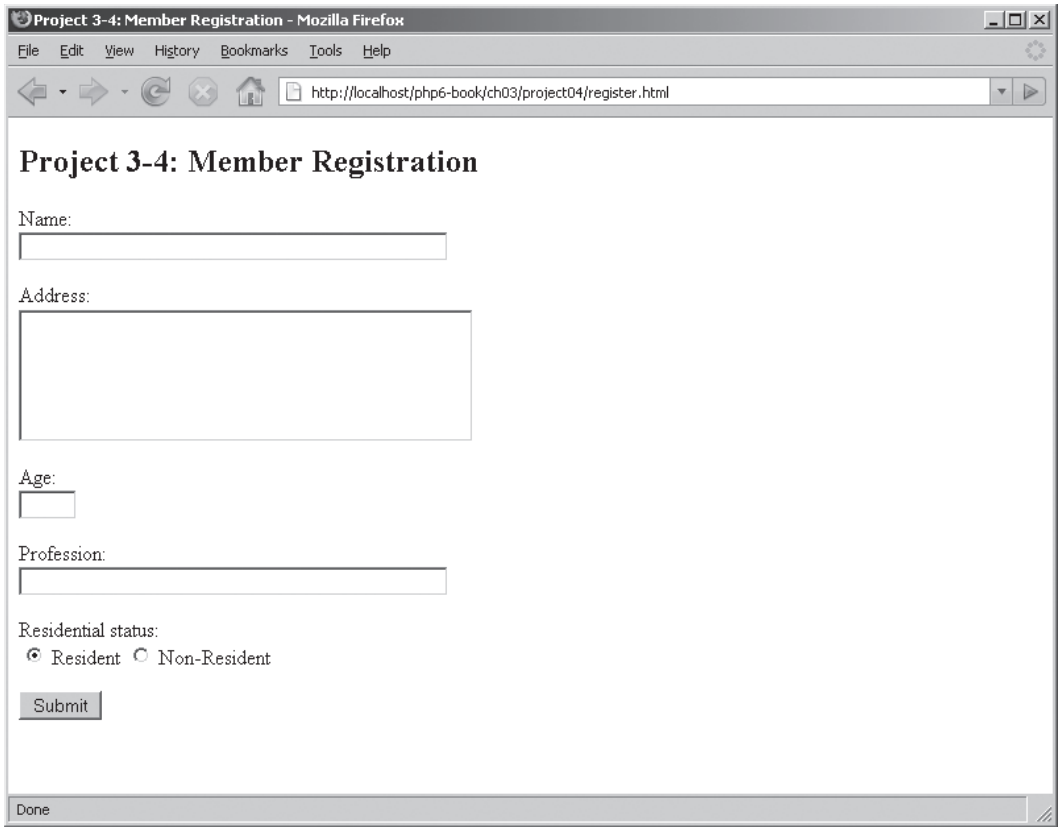


Figura 3-6 Un formulario de aplicación basado en Web

NOTA

Con el fin de enviar con éxito un correo electrónico en que PHP utilice la función `mail()`, tu archivo de configuración *php.ini* debe incluir cierta información sobre el servidor de correo electrónico o el agente de correo que va a utilizarse. Los usuarios de Windows necesitarán establecer las opciones `'SMTP'` y `'smtp_port'`, mientras que los usuarios de *NIX tal vez necesiten establecer la opción `'sendmail_path'`. Para obtener más información sobre estas opciones puedes consultar el manual de PHP en www.php.net/mail.

Una vez que se ha enviado el formulario, los datos ingresados por el solicitante son transmitidos al script de procesamiento (*registro.php*) con el método POST. El siguiente listado muestra el contenido del script:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 3-4: Registro de Miembros</title>
  </head>
  <body>
    <h2> Proyecto 3-4 Registro de Miembros</h2>
<?php
      // recupera detalles del envío POST
      $nombre = $_POST['nombre'];
      $direccion = $_POST['direccion'];
      $edad = $_POST['edad'];
      $profesion = $_POST['profesion'];
      $residencia = $_POST['residencia'];

      // valida los datos enviados
      // verifica nombre
      if(empty ($nombre)){
        die ('ERROR: Por favor proporcione su nombre.');
      }

      // verifica dirección
      if(empty ($direccion)) {
      die ('ERROR: Por favor proporcione su dirección.');
      }

      // verificar edad
      if(empty ($edad)) {
        die ('ERROR: Por favor proporcione su edad');
      } else if ($edad < 18 || $edad >60){
        die ('ERROR: Las membresías sólo están disponibles para mayores
de 18 y menores de 60 años.');
      }

      // verificar profesión
      if(empty ($profesion)) {
```

(*continúa*)

```
        die ('ERROR: Por favor proporcione su profesión.');
      }
      // verificar estatus residencial
      if(strcmp ($residencia, 'no') == 0) {
       die ('ERROR: Las membresías sólo están abiertas para residentes.');
      }

      // si llegamos a este punto
      // todos los datos de entrada han pasado la validación
      // crea y envía el mensaje de correo electrónico
      $to = 'registro@algun.dominio.com';
      $from = 'webmaster@algun.dominio.com ';
      $subject = 'Solicitud de membresía';
      $body = "Nombre: $nombre\r\nDirección: $direccion\r\n
        Edad: $edad\r\nProfesión: $profesion\r\n
        Estatus residencial: $residencia\r\n";
      if (mail($to, $subject, $body, "From: $from")){
        echo 'Gracias por enviar su solicitud.';
      } else {
        die ('ERROR: Error al enviar el mensaje');
      }
?>
  </body>
</html>
```

NOTA

Recuerda cambiar el valor de la variable `$to` en el script *registro.php*, con tu dirección de correo electrónico.

Este script principia por recuperar los valores enviados por el usuario de `$_POST` y asignando estos valores a variables regulares PHP. Después, se utiliza la función `empty()` para verificar si los campos vienen vacíos; en caso de que cualquier campo no contenga información, se genera un mensaje de error y el script termina de inmediato. Dos pruebas condicionales adicionales están presentes en esta sección: la primera es la edad del solicitante, que filtra solicitantes menores de 18 años y mayores de 60; y la segunda para el estatus residencial del solicitante, que filtra a quienes no son residentes.

Si todo es correcto, el script procede a crear un mensaje de correo electrónico, estableciendo las variables para el remitente, el destinatario, el asunto y el cuerpo del mensaje. Después, estas variables son enviadas a la función PHP `mail()`, que se encarga en realidad de hacer el trabajo pesado para enviar el mensaje de correo electrónico. Si la transmisión de éste tiene éxito, aparece la notificación correspondiente; en caso contrario, se genera un mensaje de error.

La función `mail()` es nueva para ti, por lo que merece una revisión más detallada. La función `mail()` está integrada a PHP para enviar mensajes de correo electrónico y acepta cuatro parámetros: la dirección de correo electrónico del destinatario, el asunto, el cuerpo del mensaje

y una lista adicional de encabezados del mensaje (de los cuales el encabezado `'From'` es obligatorio). Utiliza estos parámetros para construir un mensaje de correo electrónico, conectarse a un servidor de correo específico y entregar el mensaje para su envío. Si la entrega tiene éxito, `mail()` regresa un valor verdadero (true); en cualquier otro caso regresa un valor falso (false).

PRECAUCIÓN

Es importante entender el significado del valor que regresa de la función `mail()`. Si regresa un valor verdadero (true), significa que el mensaje se entregó correctamente al servidor de correo para su envío posterior. *No significa* que el mensaje haya sido transmitido al destinatario, o que éste lo haya recibido, porque PHP no tiene medios para rastrear el mensaje una vez que ha sido entregado al servidor de correo. La falta de comprensión de esta diferencia es un error común entre los programadores novatos en PHP.

La función `die()` presentada en este script también es nueva para ti: proporciona una manera muy conveniente de terminar de inmediato la ejecución del script, por lo general cuando ocurre un error. También puedes pasar un mensaje opcional a `die()`; este mensaje será presentado por PHP en el punto de terminación del script, por lo que sirve como explicación útil para que el usuario sepa qué fue lo que salió mal.

Pregunta al experto

P: ¿Puedo añadir encabezados personalizados a un mensaje de correo electrónico enviado por PHP?

R: Sí. El cuarto argumento de la función `mail()` es una cadena de caracteres que contiene tu encabezado personalizado, en formato `header:value`. Si tienes más de un encabezado personalizado para añadir, separa los encabezados con `\r\n`. El ejemplo siguiente lo ilustra:

```
<?php
// define mensaje
$to = 'bacchus@vsnl.com';
$subject = 'Hola';
$body = "Esta es una prueba";
// define encabezados personalizados
$headers = "From:webmaster@my.domain.com\r\n
 Organización:MyOrg Inc.\r\nX-Mailer:PHP";
if(mail($to, $subject, $body, $headers)){
  echo 'Su mensaje ha sido enviado.';
} else {
  die('ERROR: Error al enviar el mensaje');
}
?>
```

Resumen

El objetivo de este capítulo fue que avanzaras en tu curva de aprendizaje, desde la construcción de programas sencillos y lineales hasta la creación de aplicaciones PHP más complejas. Comenzó explicando cómo puedes añadir inteligencia a tus scripts PHP mediante el uso de pruebas condicionales y proporcionó ejemplos del uso de las declaraciones `if`, `if-else`, `if-elseif-else` y `switch-case`. Después fueron presentados los constructores bucle de PHP, mostrándote lo fácil que es automatizar acciones repetitivas con PHP y se presentaron tres tipos comunes de bucles: `while`, `do-while` y `for`.

La segunda mitad del capítulo te llevó por un viaje relámpago a través de las funciones de cadenas de caracteres y numéricas integradas en PHP; se utilizaron ejemplos comentados para explicar la manera de realizar diferentes tareas, desde sencillas comparaciones y extracciones en cadenas de caracteres hasta tareas más complejas de conversión y formato de números. Y como si eso no fuera suficiente, en este capítulo se te guió en el desarrollo de cuatro proyectos, cada uno de ellos diseñado para mostrar un uso práctico de los conceptos aprendidos. Declaraciones condicionales, bucles y funciones integradas… todos ellos fueran aplicados en los proyectos, que van desde una sencilla prueba para identificar números pares y nones hasta una aplicación Web completa que incluyó funciones para la validación de datos de entrada de un formulario y transmisión de un mensaje de correo electrónico.

Al finalizar este capítulo, ahora sabes lo suficiente acerca de la gramática y la sintaxis básica de PHP para comenzar a escribir tus propios scripts PHP con un grado de complejidad razonable. El siguiente capítulo te ayudará a expandir tu arsenal de trucos, al presentarte un nuevo tipo de variable de PHP y al enseñarte a manipular funciones de fecha y hora. Hasta entonces, ocupa un poco de tu tiempo revisando las direcciones Web que se presentan a continuación, que ofrecen información más detallada sobre los temas abordados en este capítulo:

- Declaraciones condicionales y bucles, en **www.php.net/manual/en/language.control-structures.php**
- Sobre las funciones de cadenas de caracteres de PHP, en **www.php.net/manual/en/ref.strings.php** y **www.melonfire.com/community/columns/trog/article.php?id=88**
- Funciones matemáticas PHP, en **www.php.net/manual/en/ref.math.php**
- Envío de mensajes de correo electrónico con PHP, en **www.php.net/manual/en/ref.mail.php**
- Operador ternario de PHP, en **www.php.net/manual/en/language.operators.php**

Autoexamen Capítulo 3

1. ¿Cuál es la diferencia entre una declaración `if-else` y una `if-elseif-else`?
2. Escribe una declaración condicional que verifique el valor de `$ciudad` y que despliegue un mensaje si el valor es igual a `'Minneapolis'`.
3. Explica la diferencia entre los bucles `while` y `do-while`. Ilustra tu respuesta con un ejemplo.
4. Utilizando el bucle `while`, escribe un programa que presente en pantalla la tabla de multiplicar del 8.
5. Rescribe el programa de la pregunta 4 utilizando el bucle `for`.
6. Menciona las funciones que utilizarías para
   - **A** Decodificar entidades HTML
   - **B** Poner en mayúsculas una cadena de caracteres
   - **C** Redondear un número hacia abajo
   - **D** Eliminar espacios en blanco de una cadena de caracteres
   - **E** Generar un número aleatorio
   - **F** Invertir una cadena de caracteres
   - **G** Contar las palabras de una cadena de caracteres
   - **H** Contar los caracteres de una cadena
   - **I** Terminar el procesamiento del script con un mensaje para el usuario
   - **J** Comparar dos cadenas de caracteres
   - **K** Calcular el exponente de un número
   - **L** Convertir un número decimal en un valor hexadecimal
7. ¿Cuáles serían los datos de salida de la siguiente línea de código PHP?:

   ```
   <?php printf("%09.6f", 7402.4042); ?>
   ```
8. Dada la línea `'Marco tuvo un día agradable'`, crea una nueva cadena de caracteres que diga `'Marco tuvo un helado'`.

Capítulo 4

Trabajar con matrices

Habilidades y conceptos clave

- Comprender los diferentes tipos de matrices soportados por PHP
- Procesar el contenido de matrices con el bucle `foreach`
- Utilizar matrices con formularios Web
- Ordenar, fusionar, añadir, modificar y dividir matrices utilizando las funciones integradas de PHP
- Trabajar con fechas y horas

Por ahora, los tipos simples de datos PHP deben ser pan comido para ti, porque has pasado los últimos capítulos utilizándolos en diferentes permutaciones y combinaciones, para crear aplicaciones de complejidades y usos diversos. Pero hay muchos tipos de datos en PHP, además de cadenas de caracteres, números y booleanos; PHP también da soporte a *matrices*, que te permiten agrupar y manipular más de un valor a la vez.

En este capítulo aprenderás sobre las matrices: cómo crearlas, cómo añadirles elementos y editarlas, cómo manipular los valores que contienen. También conocerás un nuevo tipo de bucle, diseñado exclusivamente para utilizarse con matrices, recibirás un curso rápido sobre algunas funciones integradas de PHP para matrices y aplicarás tus conocimientos en proyectos prácticos.

Almacenar datos en matrices

Hasta ahora, todas las variables que has utilizado contienen un solo valor. Las variables de matrices son "especiales" porque pueden contener más de un valor a la vez. Eso las hace muy útiles para almacenar valores relacionados (por ejemplo, un conjunto de nombres de frutas), como en el siguiente ejemplo:

```
<?php
// define una matriz
$frutas = array('manzana', 'plátano', 'piña', 'uva');
?>
```

Una vez que tienes una matriz como la anterior, surge una pregunta natural: ¿cómo recuperar un valor específico? Es muy sencillo: se usan números indexados para acceder a los diferentes valores almacenados en la matriz, y el cero representa el primer valor. De esta manera, para acceder al valor `'manzana'` de la matriz anterior se utiliza la notación `$frutas[0]`, mientras que el valor `'uva'` puede recuperarse con la notación `$frutas[3]`.

PHP también da soporte a un tipo de matriz un poco diferente, en que los números son reemplazados con cadenas de caracteres o "palabras clave" definidas por el usuario. He aquí una versión revisada del ejemplo anterior, que utiliza palabras clave en lugar de números indexados:

```
<?php
// define una matriz
$frutas = array(
  'm' => 'manzana'
  'p' => 'plátano'
  'i' => 'piña'
  'u' => 'uva'
);
?>
```

A este tipo de matriz se le conoce como *matriz asociada*. Las palabras clave de la matriz deben ser únicas; cada una de las palabras clave hace referencia a un solo valor y la relación palabra clave-valor está expresada por el símbolo =>. Para acceder al valor `'manzana'` de la matriz, se utiliza la notación `$frutas['m']`, mientras que `'plátano'` puede ser recuperado utilizando la notación `$frutas['p']`.

Asignar valores a matrices

Las reglas de PHP para dar nombre a las matrices son las mismas que las que se usan para las variables regulares: los nombres de las variables deben ir antecedidos con un signo de moneda (`$`) y deben iniciar con una letra o un guión bajo, opcionalmente seguidos por más letras, números o guiones bajos. No se permiten signos de puntuación ni espacios en blanco en los nombres de variables de las matrices.

Una vez que hayas decidido el nombre de tu matriz, hay dos maneras de asignarle valores. La primera es el método que viste en la sección anterior, donde los valores son asignados uno por uno y separados por comas. Este método crea una matriz estándar indexada por números. He aquí un ejemplo:

```
<?php
// define una matriz
$carros = array(
  'Ferrari',
  'Porsche',
  'Jaguar’',
  'Lamborghini',
  'Mercedes'
);
?>
```

La segunda manera de crear una matriz semejante es estableciendo valores individuales utilizando notaciones indexadas. He aquí un ejemplo, equivalente al anterior:

```
<?php
// define una matriz
$carros[0] = 'Ferrari';
$carros[1] = 'Porsche';
$carros[2] = 'Jaguar';
$carros[3] = 'Lamborghini';
$carros[4] = 'Mercedes';
?>
```

Pregunta al experto

P: En todos estos ejemplos has especificado explícitamente el número de índice de la matriz para cada declaración de asignación. ¿Qué sucede si ignoro cuál es el número de índice correcto? ¿Puedo hacer que PHP asigne de manera automática el siguiente número de índice disponible en la matriz a un nuevo valor?

R: Para asignar automáticamente el siguiente número de índice disponible en la matriz, omite los números de índice en la declaración de asignación de la matriz, como se muestra a continuación:

```
<?php
// define una matriz
$carros[] = 'Ferrari';
$carros[] = 'Lamborghini';
?>
```

También puedes utilizar ambas técnicas con matrices asociativas. Para asignar todos los valores de este tipo de matriz en una sola declaración, establece una palabra clave para cada valor y vincula ambas utilizando el conector =>; recuerda separar cada par palabra clave-valor con comas. He aquí un ejemplo:

```
<?php
// define una matriz
$datos = array(
  'nombredeusuario' => 'juan’,
  'contrasena' => 'secreta',
  'servidor' => '192.168.0.1'
);
?>
```

También puedes asignar valores a palabras clave uno por uno, como en el siguiente ejemplo:

```
<?php
// define una matriz
$datos['nombredeusuario'] = 'juan';
$datos['contrasena'] = 'secreta';
$datos['servidor'] = '192.168.0.1';
?>
```

Para acceder a un valor de la matriz en el script, simplemente utiliza el nombre y el índice/palabra clave en una expresión y PHP lo reemplazará con el valor correspondiente cuando se ejecute el script, igual que en una variable estándar. He aquí un ejemplo de cómo funciona:

```
<?php
// define una matriz
$datos = array(
  'nombredeusuario' => 'juan',
  'contrasena' => 'secreta',
  'servidor' => '192.168.0.1',
);

// usa un valor de la matriz
echo 'La contraseña es: ' . $datos['contrasena'];
?>
```

NOTA

¿Recuerdas los contenedores especiales de variables llamados `$_GET` y `$_POST` que viste por primera vez en el capítulo 2? Pues bien, ¡también son matrices! Cuando se envía un formulario, PHP convierte automáticamente cada elemento de él en un miembro de la matriz, ya sea `$_GET` o `$_POST`, dependiendo del método de envío.

Modificar valores de matrices

Modificar el valor de una matriz es tan sencillo como el de cualquier otra variable: simplemente asigna un nuevo valor al elemento utilizando su número de índice o su palabra clave. Examina el siguiente ejemplo, que modifica el segundo elemento de la matriz `$carnes`; cambiará el valor `'jamón'` y pondrá en su lugar el valor `'pavo'`:

```
<?php
// define una matriz
$carnes = array(
  'pescado',
  'pollo',
  'jamón',
  'cordero'
);
```

```
// cambia 'jamón' por 'pavo'
$carnes[2] = 'pavo';
?>
```

Para eliminar un elemento de la matriz, utiliza la función `unset()` en el correspondiente número de índice o palabra clave:

```
<?php
// define una matriz
$carnes = array(
  'pescado',
  'pollo',
  'jamón',
  'cordero'
);

// elimina 'pescado'
unset($carnes[0]);
?>
```

Pregunta al experto

P: Si elimino un elemento en medio de la matriz, ¿qué sucede con los valores anteriores y posteriores?

R: Si eliminas un elemento con la función `unset()`, PHP lo declarará nulo (`NULL`), pero no volverá a indexar automáticamente la matriz. Si se trata de una matriz indexada numéricamente, puedes volver a indexarla, para eliminar los huecos en la secuencia, utilizando la función PHP `array_multisort()` sobre tu matriz.

También puedes eliminar un elemento de una matriz utilizando la función `array_pop()` o `array_push()`; ambas son abordadas más adelante, en este mismo capítulo.

Recuperar el tamaño de la matriz

Saber cuántos valores contiene una matriz es una tarea importante cuando se trabaja con ellos, en especial cuando se combinan con bucles. Esto se resuelve fácilmente con la función PHP `count()`, que acepta la variable de una matriz como parámetro y regresa un número entero que indica cuántos elementos contiene. He aquí un ejemplo:

```
<?php
// define una matriz
$datos = array('lunes', 'martes', 'miércoles');
```

```
// obtiene el tamaño de la matriz
echo 'La matriz tiene ' . count($datos) . ' elementos';
?>
```

TIP

En lugar de la función `count()`, puedes utilizar `sizeof()`, que hace exactamente lo mismo.

Pregunta al experto

P: ¿Existe un límite para los elementos que puede contener una matriz en PHP?

R: No. La cantidad de elementos en una matriz está limitada por la memoria disponible, como se define en la variable de configuración `'memory_limit'`, en el archivo de configuración de PHP.

Matrices anidadas

PHP también te permite combinar matrices, colocando una dentro de otra sin límite de profundidad. Esto es útil cuando se trabaja con información ordenada de manera estructurada y jerárquica. Para ilustrarlo, examina el siguiente ejemplo:

```
<?php
// define matrices anidadas
$directorio = array(
  array(
    'nombre' => 'Raymundo Rabbit',
    'tel' => '1234567',
    'correo' => 'ray@planetaconejo.in',
    ),
  array(
    'nombre' => 'David Duck',
    'tel' => '8562904',
    'correo' => 'dduck@lagodepatos.corp',
    ),
  array(
    'nombre' => 'Omar Horse',
    'tel' => '5942033',
    'correo' => 'reyomar@mercadodelgranjero.cosasdecaballos.com',
  )
);
?>
```

En este ejemplo, `$directorio` es una matriz anidada con dos niveles de profundidad. El primero está indexado numéricamente, y cada elemento representa una pieza del directorio telefónico. Cada uno de estos elementos es, a su vez, una matriz asociativa que contiene información específica de los atributos correspondientes a cierta pieza del directorio, como el nombre del contacto, su número telefónico y su dirección de correo electrónico.

Para acceder a un valor que se encuentra en algunos de los niveles internos de la matriz, utiliza la secuencia jerárquica correcta de números de índice y palabras clave para obtenerlo. He aquí un ejemplo, que presenta el número telefónico del contacto `'David Duck'`:

```
<?php
// define matrices anidadas
$directorio = array(
  array(
    'nombre' => 'Raymundo Rabbit',
    'tel' => '1234567',
    'correo' => 'ray@planetaconejo.in',
    ),
  array(
    'nombre' => 'David Duck',
    'tel' => '8562904',
    'correo' => 'dduck@lagodepatos.corp',
    ),
  array(
    'nombre' => 'Omar Horse',
    'tel' => '5942033',
    'correo' => 'reyomar@mercadodelgranjero.cosasdecaballos.com',
  )
);

// accede al valor anidado
echo "El número telefónico de David Duck es: " . $directorio[1] ['tel'];
?>
```

TIP

¿Quieres mirar dentro de una matriz para ver lo que contiene? Las funciones `var_dump()` y `print_r()` abordadas en el capítulo 2 funcionan tan bien en las matrices como lo hacen en las variables estándar. Compruébalo tú mismo.

Procesar matrices con bucles e iteradores

Muchas veces tu programa PHP necesitará recorrer una matriz, realizando una operación o un cálculo con cada valor que encuentre. La mejor manera de realizar esta tarea es con bucles, los cuales conociste en el capítulo anterior. Examina el siguiente ejemplo, que establece una matriz y luego realiza operaciones iterativas utilizando el bucle `for`:

```
<?php
// define una matriz
$ciudades = array('Londres', 'París', 'Madrid', 'Los Ángeles', 'Bombay',
'Yakarta');

// hace iteraciones sobre la matriz
// presenta cada valor
for ($i=0; $i<count($ciudades); $i++) {
  echo $ciudades[$i] . "\r\n";
}
?>
```

En este ejemplo, el bucle `for` hace las iteraciones sobre la matriz `$ciudades`, y presenta en pantalla cada valor encontrado. El bucle se ejecuta para cada elemento de la matriz; esta información se averigua rápidamente invocando la función `count()`.

El bucle foreach

Tal vez recuerdes que en el capítulo anterior el tema de los bucles quedó inconcluso, y que aún faltaba por explicar un tipo de ellos. El bucle al que nos referimos es `foreach`, que fue introducido en PHP por primera vez en la versión 4.0 y representa la manera más sencilla de realizar iteraciones sobre matrices, ¡nada sorprendente dado que fue diseñado con ese fin específico!

Con el bucle `foreach`, cada vez que se ejecuta un bucle, el elemento en uso de la matriz es asignado a una variable temporal, la cual puede ser procesada de la forma que más te plazca: mandarlo a pantalla, copiarlo a otra variable, usarlo en cálculos y demás. A diferencia del bucle `for`, `foreach` no utiliza un contador, porque automáticamente "sabe" la posición que ocupa dentro de la matriz en un determinado momento y se desplaza hacia adelante en forma continua hasta que alcanza el final de la matriz; en ese punto se detiene automáticamente.

La mejor manera de comprender esta maravilla de la automatización es con un ejemplo. El siguiente listado lo muestra en acción; es una revisión del ejemplo anterior en que se utiliza el bucle `foreach` en lugar de `for`:

```
<?php
// define una matriz
$ciudades = array('Londres', 'París', 'Madrid', 'Los Ángeles', 'Bombay',
'Yakarta');

// hace iteraciones sobre la matriz
// presenta cada valor
foreach($ciudades as $c) {
  echo "$c \r\n";
}
?>
```

El bucle `foreach` también funciona con matrices asociativas, sólo que en esos casos utiliza dos variables temporales (una para la palabra clave y otra para el valor). El siguiente ejemplo ilustra la diferencia:

```
<?php
// define una matriz
$ciudades = array(
  "Reino Unido" => "Londres",
  "Estados Unidos" => "Washington",
  "Francia" => "París",
  "India" => "Delhi"
);

// hace iteraciones sobre la matriz
// presenta cada valor
foreach($ciudades as $clave => $valor){
  echo "$valor es la capital de $clave. \r\n";
}
?>
```

El iterador de matrices

Como opción, tal vez quieras utilizar un iterador de matrices llamado precisamente ArrayIterator (nuevo en PHP 5.0), que proporciona una herramienta extensible, lista para utilizarse y recorrer en bucle los elementos de una matriz. He aquí un ejemplo:

```
<?php
// define una matriz
$ciudades = array(
  "Reino Unido" => "Londres",
  "Estados Unidos" => "Washington",
  "Francia" => "París",
  "India" => "Delhi"
);

// crea un objeto ArrayIterator
$iterador = new ArrayIterator($ciudades);

// regresa al principio de la matriz
$iterador ->rewind();

// hace iteraciones sobre la matriz
// presenta cada valor
while($iterador->valid()){
    print $iterador->current() . " es la capital de " . $iterador->key()
. ". \r\n";
    $iterador->next();
}
?>
```

En este ejemplo, el objeto ArrayIterator se inicializa con una variable de matriz, y el método `rewind()`, perteneciente al objeto, se utiliza para enviar el puntero interno de la matriz al primer elemento de éste. Un bucle `while`, que se ejecuta mientras exista un elemento `valid()`, puede utilizarse entonces para hacer las iteraciones sobre la matriz. Las palabras clave individuales de la matriz son recuperadas por el método `key()`, y sus valores correspondientes son recuperados con el método `current()`. El método `next()` avanza el puntero interno de la matriz al siguiente elemento del mismo.

NOTA

No te preocupes si la sintaxis utilizada en el objeto ArrayIterator no te queda completamente clara. Los objetos son abordados con cierta profundidad en el capítulo 5, y los ejemplos anteriores tendrán mayor sentido conforme avances por el presente capítulo.

Prueba esto 4-1 Promediar las calificaciones de un grupo

Ahora que has comprendido los aspectos básicos del trabajo con matrices, hagamos una pequeña aplicación que demuestre su uso práctico. En esta sección escribirás un pequeño script que acepte una matriz de valores; éstos representan las calificaciones numéricas individuales de un grupo de estudiantes de cierto curso, y luego calcularás varias estadísticas de resumen, incluido el promedio general y la cantidad de estudiantes que se encuentran dentro de los rangos del 20% superior y el 20% inferior dentro del grupo.

He aquí el código (*calificaciones.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 4-1: Promedio de Desempeño</title>
  </head>
  <body>
    <h2>Proyecto 4-1: Promedio de Desempeño</h2>
<?php
// define una matriz de calificaciones
// que van de 1 a 100
$calificaciones = array(
   25, 64, 23, 87, 56, 38, 78, 57, 98, 95,
   81, 67, 75, 76, 74, 82, 36, 39,
   54, 43, 49, 65, 69, 69, 78, 17, 91
);

// obtiene la cantidad de calificaciones
$cuenta = count($calificaciones);
```

(*continúa*)

```
// hace iteraciones sobre la matriz
// calcula el total y el 20% superior e inferior
$total = $sup = $inf = 0;
foreach ($calificaciones as $g){
  $total += $g;
  if($g <= 20){
    $inf++;
  }

  if ($g >= 80){
    $sup++;
  }
}

// calcula el promedio
$prom = round($total / $cuenta);

// presenta en pantalla las estadísticas
echo "Promedio del grupo: $prom <br />";
echo "Número de estudiantes en el 20% inferior: $inf <br />";
echo "Número de estudiantes en el 20% superior: $sup <br />";
?>
  </body>
</html>
```

Este script comienza definiendo una matriz, que contiene las calificaciones de cada estudiante en un curso. Luego, un bucle `foreach` hace las iteraciones sobre la matriz, con lo que genera un acumulativo total; este total es dividido entre el número de estudiantes y es recuperado por la función `count()`, para calcular el promedio del grupo.

Como las calificaciones van del 1 al 100, no es difícil calcular la cantidad de estudiantes que se encuentran en el 20% superior e inferior del grupo. Dentro del bucle `foreach` cada valor es verificado para probar si es menor a 20 o superior a 80 y sus respectivos contadores se incrementan de acuerdo con ello. De esta manera, el resumen de los datos se presenta en una página Web, como se muestra en la figura 4-1.

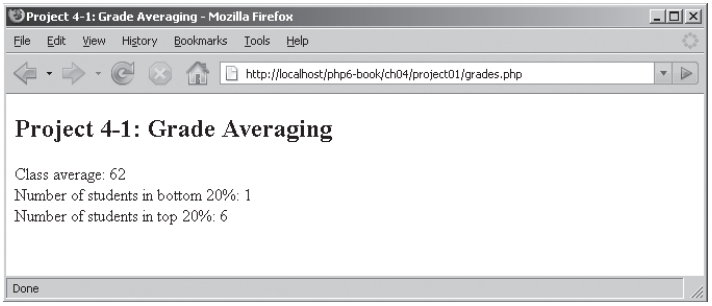


Figura 4-1 Las estadísticas de resumen generadas por el programa que promedia el desempeño del grupo

Utilizar matrices con formularios

Las matrices son muy poderosas cuando se combinan con elementos de formularios Web que dan soporte a más de un valor, como listas de selección múltiple o casillas de verificación agrupadas. Para capturar en una matriz los datos proporcionados por el usuario, simplemente añade un juego de corchetes al elemento `'name'` del formulario para convertirlo automáticamente en una matriz de PHP cuando se envíe.

La manera más fácil de ilustrarlo es con un ejemplo. Examina el siguiente formulario, que contiene una lista de selección múltiple con nombres de músicos populares:

```
<form method="post" action="matriz-formulario.php">
    Selecciona el nombre de tu artista favorito: <br />
    <select name="artistas[]" multiple="true">
      <option value="Britney Spears">Britney Spears</option>
      <option value="Aerosmith">Aerosmith</option>
      <option value="Black-Eyed Peas">Black-Eyed Peas</option>
      <option value="Diana Ross">Diana Ross</option>
      <option value="Foo Fighters">Foo Fighters</option>
    </select>
    <p>
    <input type="submit" name="submit" value="Enviar" />
  </form>
```

Pon atención al atributo `'name'` del elemento `<select>`, que tiene el nombre `artistas[]`. Esto le indica a PHP que, cuando se envíe el formulario, todos los valores seleccionados de la lista deben transformarse en elementos de una matriz. El nombre de ésta será `$_POST['artistas']`, y tendrá más o menos este aspecto:

```
Array
(
    [0] => Britney Spears
    [1] => Black-Eyed Peas
    [2] => Diana Ross
)
```

Prueba esto 4-2 Seleccionar sabores de pizzas

Una aplicación práctica hará que sea más fácil comprender la manera en que PHP interactúa con variables de matrices y formularios. La siguiente aplicación presenta al usuario un formulario que contiene varios sabores de pizzas y le solicita que seleccione sus favoritas, mediante casillas de verificación. He aquí el formulario (*pizza.html*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
```

(*continúa*)

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 4-2: Selector de Pizzas Favoritas</title>
  </head>
  <body>
    <h2>Proyecto 4-2: Selector de Pizzas Favoritas</h2>
    <form method="post" action="pizza.php">
      Seleccione su pizza favorita: <br />
      <input type="checkbox" name="favorita[]" value="tomate">Tomate</
input>
      <input type="checkbox" name="favorita[]" value="cebolla">Cebolla</
input>
      <input type="checkbox" name="favorita[]" value="jalapeños">Jalapeño
s</input>
      <input type="checkbox" name="favorita[]" value="aceitunas">Aceituna
s</input>
      <input type="checkbox" name="favorita[]" value="suiza">Suiza</
input>
      <input type="checkbox" name="favorita[]" value="piña">Piña</input>
      <input type="checkbox" name="favorita[]" value="tocino">Tocino</
input>
      <input type="checkbox" name="favorita[]" value="pollo">Pollo</
input>
      <input type="checkbox" name="favorita[]" value="jamón">Jamón</
input>
      <input type="checkbox" name="favorita[]" value="anchoas">Anchoas</
input>
      <input type="checkbox" name="favorita[]" value="extraqueso">Extra
queso</input>
<p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
  </body>
</html>
```

La figura 4-2 muestra cómo aparece el formulario.

Y a continuación, el código del script que capta los datos del formulario (*pizza.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 4-2: Selector de Pizzas Favoritas</title>
  </head>
  <body>
    <h2>Proyecto 4-2: Selector de Pizzas Favoritas</h2>
     Usted ha seleccionado las siguientes pizzas como sus favoritas: <br />
      <ul>
```

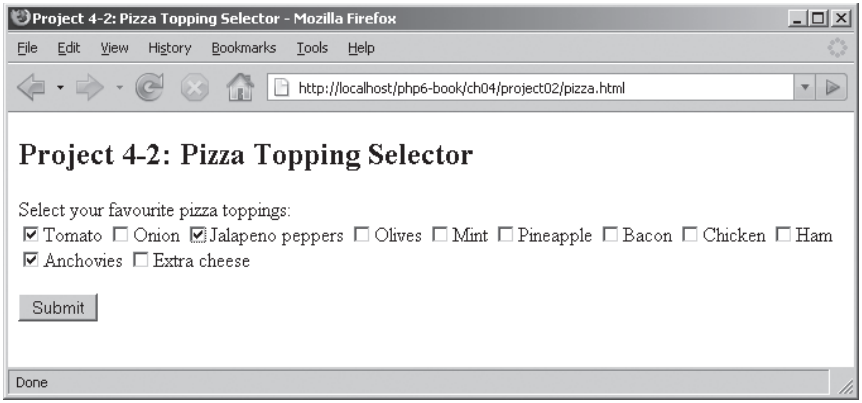


Figura 4-2 Un formulario Web para seleccionar pizzas favoritas

```
      <?php
      foreach ($_POST['favorita'] as $f) {
        echo "<li>$f</li> \r\n";
      }
      ?>
      </ul>
  </body>
</html>
```

Cuando se envía el formulario Web, automáticamente PHP coloca todos los valores seleccionados en la matriz `$_POST['favorita']`. Esta matriz puede procesarse como cualquiera otra, en este caso con un bucle `foreach` que presenta en pantalla las pizzas seleccionadas en una lista. La figura 4-3 muestra los datos de salida.

Figura 4-3 El resultado del formulario enviado

Trabajar con funciones de matrices

PHP cuenta con numerosas funciones integradas para manipular matrices; da soporte a operaciones que van desde la búsqueda y comparación, hasta la organización y conversión de elementos dentro de la matriz. La tabla 4-1 muestra algunas de esas funciones.

Conversión entre cadenas de caracteres y matrices

PHP te permite convertir una cadena de caracteres en una matriz, al tratar la cadena con separadores definidos por el usuario y asignar los segmentos resultantes a una matriz. La función

Función	Lo que hace
`explode()`	Divide una cadena de caracteres en elementos de una matriz
`implode()`	Conjunta elementos de una matriz en una cadena de caracteres
`range()`	Genera un rango de números como matriz
`min()`	Encuentra el valor menor dentro de la matriz
`max()`	Encuentra el valor mayor dentro de la matriz
`shuffle()`	Reordena aleatoriamente una secuencia de elementos dentro de la matriz
`array_slice()`	Extrae un segmento de la matriz
`array_shift()`	Elimina un elemento del principio de la matriz
`array_unshift()`	Añade un elemento al principio de la matriz
`array_pop()`	Elimina un elemento del final de la matriz
`array_push()`	Añade un elemento al final de la matriz
`array_unique()`	Elimina elementos repetidos dentro de la matriz
`array_reverse()`	Invierte la secuencia de los elementos dentro de la matriz
`array_merge()`	Combina dos o más matrices
`array_intersect()`	Calcula los elementos comunes entre dos o más matrices
`array_diff()`	Calcula las diferencias entre dos matrices
`in_array()`	Verifica si un valor existe dentro de la matriz
`array_key_exists()`	Verifica si una clave en particular existe dentro de la matriz
`sort()`	Ordena una matriz
`asort()`	Ordena una matriz asociativa por valor
`ksort()`	Ordena una matriz asociativa por palabra clave
`rsort()`	Revierte el orden de una matriz
`krsort()`	Revierte el orden de una matriz asociativa por valor
`arsort()`	Revierte el orden de una matriz asociativa por palabra clave

Tabla 4-1 Funciones de matriz comunes en PHP

PHP que realiza esta tarea lleva el nombre de `explode()`, acepta dos argumentos: el separador y la cadena de caracteres fuente, y regresa una matriz. He aquí un ejemplo:

```
<?php
// define una cadena de caracteres
$cadena = 'policía,sastre,soldado,espía';

// convierte una cadena en matriz
// datos de salida: ('policía', 'sastre', 'soldado', 'espía')
$matriz = explode(',', $cadena);
print_r($matriz);
?>
```

También es posible revertir el proceso, conjuntar elementos de una matriz en una sola cadena de caracteres utilizando el "pegamento" proporcionado por el usuario. La función PHP para realizar esta tarea recibe el nombre de `implode()` y se muestra en el siguiente ejemplo:

```
<?php
// define una matriz
$matriz = array('uno', 'dos', 'tres', 'cuatro');

// convierte matriz en cadena de caracteres
// datos de salida: 'uno y dos y tres y cuatro'
$cadena = implode(' y ', $matriz);
print_r($cadena);
?>
```

Trabajar con rangos de números

Si tratas de llenar una matriz con un rango de números, la función `range()` es una mejor opción que ingresar manualmente cada valor. Esta función acepta dos extremos y regresa una matriz que contiene todos los números existentes entre los extremos establecidos. He aquí un ejemplo; genera una matriz que contiene todos los valores entre 1 y 1 000:

```
<?php
// define una matriz
$matriz = range(1,1000);
print_r($matriz);
?>
```

Como opción, si ya cuentas con una matriz de números y quieres calcular el mínimo y el máximo de la serie, las funciones PHP `min()` y `max()` serán de utilidad; aceptan una matriz

numérica y regresan el valor menor y mayor, respectivamente, de los elementos contenidos en la matriz. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz
$matriz = array(7, 36, 5, 48, 28, 90, 91, 3, 67, 42);

// obtiene mínimos y máximos
// datos de salida: 'El mínimo es 3 y el máximo es 91'
echo 'El mínimo es ' . min($matriz) . ' y el máximo es ' . max($matriz);
?>
```

Extraer segmentos de la matriz

PHP te permite cortar una matriz en partes pequeñas con la función `array_slice()`, que acepta tres argumentos: la matriz original, la posición del índice (*offset*) donde debe comenzar el corte y la cantidad de elementos que debe regresar a partir de la posición de inicio. El siguiente ejemplo ilustra esta acción:

```
<?php
// define una matriz
$arcoiris = array('violeta', 'índigo', 'azul', 'verde', 'amarillo',
    'naranja', 'rojo');

// extrae 3 valores centrales
// datos de salida: ('azul', 'verde', 'amarillo')
$matriz = array_slice($arcoiris, 2, 3);
print_r($matriz);
?>
```

Para extraer un segmento a partir del final de la matriz (en lugar de hacerlo desde el principio), inserta un valor negativo para la posición del índice en la función `array_slice()`, como se presenta en la siguiente revisión del ejemplo anterior:

```
<?php
// define una matriz
$arcoiris = array('violeta', 'índigo', 'azul', 'verde', 'amarillo',
    'naranja', 'rojo');

// extrae 3 valores centrales
// a partir del final
// datos de salida: ('azul', 'verde', 'amarillo')
$matriz = array_slice($arcoiris, -5, 3);
print_r($matriz);
?>
```

Añadir y eliminar elementos de la matriz

PHP contiene cuatro funciones que te permiten añadir o eliminar elementos del principio o el final de la matriz: `array_unshift()` añade un elemento al principio; `array_shift()` elimina el primer elemento; `array_push()` añade un elemento al final; `array_pop()` elimina el último elemento de la matriz. El siguiente ejemplo los muestra en acción:

```
<?php
// define una matriz
$filmes = array('El Rey León', 'Cars', 'Bichos');

// elimina el primer elemento de la matriz
array_shift($filmes);

// elimina el último elemento de la matriz
array_pop($filmes);

// añade un elemento al final de la matriz
array_push($filmes, 'Ratatouille');

// añade un elemento al principio de la matriz
array_unshift($filmes, 'Los Increíbles');

// muestra la matriz
// datos de salida: ('Los Increíbles', 'Cars', 'Ratatouille')
print_r($filmes);
?>
```

NOTA

Las funciones `array_unshift()`, `array_shift()`, `array_push()` y `array_pop()` sólo deben utilizarse con matrices indexadas numéricamente y no con asociativas. Cada una de estas funciones indexa de nuevo la matriz para llevar la cuenta del valor o los valores añadidos o eliminados durante su operación.

Eliminar elementos duplicados en la matriz

PHP te permite limpiar una matriz de valores duplicados con la función `array_unique()`, que acepta la matriz completa y regresa una nueva que sólo contiene valores únicos. El siguiente ejemplo la muestra en acción:

```
<?php
// definir una matriz
$duplicados = array('a', 'b', 'a', 'c', 'e', 'd', 'e');
```

```
// elimina duplicados
// datos de salida: ('a', 'b', 'c', 'e', 'd')
$originales = array_unique($duplicados);
print_r($originales);
?>
```

Ordenar aleatoriamente e invertir la matriz

La función PHP `shuffle()` transforma el orden actual de los elementos de la matriz para darles un orden aleatorio, mientras que la función `array_reverse()` invierte el orden de sus elementos. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz
$arcoiris = array('violeta', 'índigo', 'azul', 'verde', 'amarillo',
    'naranja', 'rojo');

// da orden aleatorio a la matriz
shuffle($arcoiris);
print_r($arcoiris);

// invierte los elementos de la matriz
// datos de salida: ('rojo', 'naranja', 'amarillo', 'verde', 'azul',
//  'índigo', 'violeta')
$matriz = array_reverse($arcoiris);
print_r($matriz);
?>
```

Realizar búsquedas en la matriz

La función `in_array()` revisa la matriz en busca de un valor específico y regresa una respuesta verdadera (true) en caso de localizarlo. He aquí un ejemplo, que busca la cadena de caracteres `'Barcelona'` en la matriz `$ciudades`:

```
<?php
// define una matriz
$ciudades = array('Londres', 'París', 'Barcelona', 'Lisboa', 'Zurich');

// busca un valor dentro de la matriz
echo in_array('Barcelona', $ciudades);
?>
```

Si en lugar de valores quieres buscar palabras clave de una matriz asociativa, PHP también puede realizar esa acción: la función `array_key_exists()` busca una coincidencia entre las palabras clave de la matriz y el término específico que se busca. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz
$ciudades = array(
  "Reino Unido" => "Londres",
  "Estados Unidos" => "Washington",
  "Francia" => "París",
  "India" => "Delhi"
);

// busca palabra clave en la matriz
echo array_key_exists('India', $ciudades);
?>
```

Ordenar matrices

PHP cuenta con varias funciones integradas diseñadas para ordenar matrices de muy diversas maneras. La primera es la función `sort()`, que te permite disponer las matrices indexadas numéricamente por orden alfabético o numérico, de menor a mayor. He aquí un ejemplo:

```
<?php
// define una matriz
$datos = array(15, 81, 14, 74, 2);

// ordena y presenta la matriz
// datos de salida: (2, 14, 15, 74, 81)
sort($datos);
print_r($datos);
?>
```

Sin embargo, cuando quieres organizar una matriz asociativa, es mejor utilizar la función `asort()`, que mantiene la correlación entre palabras clave y valores mientras realiza la organización. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz
$perfil = array(
  "nombre" => "Susana",
  "apellido" => "De Tal",
  "sexo" => "femenino",
  "sector" => "Administración de recursos"
);

// ordena por valor
// datos de salida: ('sector' => 'Administración de recursos'
// 'apellido' => 'De Tal',
```

```
// 'nombre' => 'Susana'
// 'sexo' => 'femenino')
asort($perfil);
print_r($perfil);
?>
```

La función `ksort()` también se aplica a matrices asociativas; utiliza las palabras clave en lugar de los valores para ordenar la matriz. He aquí un ejemplo:

```
<?php
// define una matriz
$perfil = array(
  "nombre" => "Susana",
  "apellido" => "De Tal",
  "sexo" => "femenino",
  "sector" => "Administración de recursos"
);

// organiza por palabra clave
// datos de salida: ('apellido' => 'De Tal',
// 'nombre' => 'Susana',
// 'sector' => 'Administración de recursos',
// 'sexo' => 'femenino')
ksort($perfil);
print_r($perfil);
?>
```

TIP

Para invertir la secuencia ordenada que generaron `sort()`, `asort()` y `ksort()`, utiliza las funciones `rsort()`, `arsort()` y `krsort()`, respectivamente.

Combinar matrices

PHP te permite combinar dos o más matrices con la función `array_merge()`, que acepta variables de una o más matrices. El siguiente ejemplo y los datos de salida muestran su uso:

```
<?php
// define matrices
$oscuro = array('negro', 'café', 'azul');
$claro = array('blanco', 'plateado', 'amarillo');

// combina matrices
// datos de salida:('negro', 'café', 'azul',
//                  'blanco', 'plateado', 'amarillo')
$colores = array_merge($oscuro, $claro);
print_r($colores);
?>
```

Comparar matrices

PHP proporciona dos funciones para comparar matrices: `array_intersect()`, que regresa los valores comunes entre dos matrices y `array_diff()` que regresa los valores de la primera matriz que no existen en la segunda. He aquí un ejemplo que ilustra ambas en acción:

```
<?php
// define matrices
$naranja = array('rojo', 'amarillo');
$verde = array('amarillo', 'azul');

// encuentra elementos comunes
// datos de salida: ('amarillo')
$comunes = array_intersect($naranja, $verde);
print_r($comunes);

// encuentra elementos de la primera matriz que no están en la segunda
// datos de salida: ('rojo')
$unico = array_diff($naranja, $verde);
print_r($unico);
?>
```

TIP

También puedes comparar matrices con el operador de comparación de PHP (==), de manera muy similar a la comparación de variables.

Prueba esto 4-3 Verificar números primos

Ahora que has visto las muchas maneras en que PHP te permite trabajar con matrices, hagamos un pequeño ejemplo práctico que muestre algunas de estas funciones integradas en acción. La siguiente aplicación solicita al usuario que ingrese una serie de números en un formulario Web, y regresa un mensaje indicando cuáles son números primos.

He aquí el código (*primos.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 4-3: Prueba de Números Primos</title>
  </head>
  <body>
    <h2>Proyecto 4-3: Prueba de Números Primos</h2>
```

(*continúa*)

```
<?php
    // si el formulario no ha sido enviado
    // presenta el formulario
    if (!isset ($_POST['submit'])) {
?>
    <form method="post" action="primos.php">
      Escriba una lista de números, separados por comas: <br />
      <input type='text' name='num' />
      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
<?php
    // si el formulario ya ha sido enviado
    // procesa los datos de entrada
    } else {
     // recupera los números del envío POST
     // los convierte en una matriz de acuerdo con la separación por
comas
      $cadNums = $_POST['num'];
      $matrizNums = explode(',', $_POST['num']);
      $primos = array();
      $primosMarca = 0;

      // itera sobre la matriz
      // obtiene el valor absoluto de cada número
      foreach ($matrizNums as $n){
        $n = trim(abs ($n));

        // verifica si cada número es primo:
        // prueba cada número dividiéndolo
        // por todos los números existentes entre él mismo y el 2
        // si no es perfectamente divisible por ninguno
        // el número es primo
        for ($i=2; $i<$n; $i++){
          $primosMarca = 0;
          if (($n%$i) == 0){
            break;
          }
          $primosMarca = 1;
        }

        // si es primo
        // se añade a la matriz de salida
        if ($primosMarca == 1){
          array_push($primos, $n);
        }
```

```
        }
        // verifica si se encontraron números primos
        // de ser así, los organiza y elimina duplicados de la matriz
        // presenta el mensaje en pantalla
        if (count ($primos) > 0) {
          $primos = array_unique($primos);
          sort ($primos);
         echo 'Los siguientes números son primos: ' . implode($primos, ' ');
        } else {
          echo 'No se encontraron números primos';
        }
    }
?>
  </body>
</html>
```

Utilizando una técnica que ahora debe serte familiar, esta aplicación combina un formulario Web con la página de resultados en un mismo script, separando ambas con la declaración condicional `if-else`. El formulario Web permite que el usuario ingrese una serie de números separados por comas. La figura 4-4 presenta cómo se ve el formulario en el explorador Web.

Una vez que el usuario ingresa los números y envía el formulario, se dispara la segunda parte del script. Debe ser claro que esta parte del script utiliza muchas de las funciones de matrices abordadas en la sección anterior. Primero, la función `explode()` se encarga de extraer los números individuales enviados por el usuario y colocarlos en una matriz, utilizando

Figura 4-4 Formulario Web para ingresar los números

(*continúa*)

Figura 4-5 Los resultados del formulario enviado, que muestran los números primos encontrados

la coma como separador. A continuación, un bucle `foreach` itera sobre la matriz, calculando primero el valor absoluto de cada número y luego dividiéndolo entre todos los números existentes entre él y el 2 para determinar si es un número primo o no.

Si un número no se puede dividir por lo menos entre otro número diferente de sí mismo o de la unidad, sin que quede un residuo, se le considera primo, y se utiliza la función `array_push()` para añadirlo como elemento a una nueva matriz llamada `$primos`. Una vez que todos los números enviados por el usuario pasan por este proceso, se utiliza la función `count()` para averiguar si se encontraron números primos en el proceso; de ser así, la matriz `$primos` pasa por una verificación que elimina los duplicados, ordena los valores de menor a mayor y envía los resultados a la página como datos de salida.

La figura 4-5 muestra los datos de salida obtenidos de la secuencia enviada.

Trabajar con fechas y horas

Además de la capacidad de almacenar múltiples valores en una matriz, PHP también cuenta con un conjunto completo de funciones para trabajar con fechas y horas. Aunque el lenguaje no cuenta con un tipo de datos dedicado exclusivamente a valores de fecha y hora, ofrece a los programadores gran flexibilidad para darles formato y manipularlos.

Pregunta al experto

P: ¿Qué es el sello cronológico (timestamp) de UNIX?

R: Muy sencillo, el valor de sello cronológico de UNIX para cualquier punto en el tiempo, representa la cantidad de segundos que han transcurrido desde la medianoche del 1 de enero de 1970 y el momento actual. De esa manera, por ejemplo, el punto del tiempo ubicado el 5 de enero de 2008 a las 10:15:00 a.m. se representa en formato de sello cronológico de UNIX con el formato 1199508300.

PHP puede convertir automáticamente un valor de fecha en el formato de sello cronológico de UNIX con la función `mktime()`, que acepta argumentos para día, mes, año, hora, minuto y segundo, para regresar el sello cronológico correspondiente a ese instante.

Generar fechas y horas

Como muchos otros lenguajes de programación, PHP representa valores de fecha/hora en el formato de sello cronológico de UNIX. La generación del sello cronológico se realiza generalmente con la función `mktime()`, que acepta una serie de parámetros de fecha y hora convencionales para convertirlos en su correspondiente valor de sello cronológico. Para ilustrarlo, examina el siguiente ejemplo, que regresa el sello cronológico correspondiente al 5 de enero de 2008 a las 10:15:00 a.m.

```
<?php
// regresa el sello cronológico del 5 de enero de 2008 a las 10:15
// datos de salida: 1199508300
echo mktime(10,15,00,1,5,2008);
?>
```

La invocación de la función `mktime()` sin argumentos regresa el sello cronológico de UNIX correspondiente a la hora actual:

```
<?php
// regresa el sello cronológico del momento actual
echo mktime();
?>
```

Otra forma de obtener la fecha y hora actuales es con la función PHP `getdate()`, que regresa una matriz asociativa que contiene información sobre la fecha y hora actuales. He aquí un ejemplo de esa matriz:

```
Array
(
    [seconds] => 33
    [minutes] => 27
```

```
    [hours] => 19
    [mday] => 12
    [wday] => 1
    [mon] => 11
    [year] => 2007
    [yday] => 315
    [weekday] => Monday
    [month] => November
    [0] => 1194875853
)
```

He aquí un ejemplo de esta función en acción:

```
<?php
// obtiene la fecha y horas actuales como una matriz
$hoy = getdate();

// datos de salida: 'Fecha y hora actual 19:26:23 del 12-11-2007'
echo 'Fecha y hora actual ' . $hoy ['hours'] . ':' . $hoy ['minutes'] .
':' . $hoy ['seconds'] . ' del ' . $hoy ['mday'] . '-' . $hoy ['mon'] .
'-' . $hoy ['year'];
?>
```

TIP

Advierte que la matriz regresada por `getdate()` incluye la representación de la fecha en formato de sello cronológico de UNIX en la posición 0 del índice.

Formar fechas y horas

Casi siempre, generar el sello cronológico es sólo la mitad de la batalla: también necesitas encontrar una forma de presentarlo que sea legible para los humanos. Ahí es donde la función PHP `date()` entra en juego: esta función te permite transformar la horrible y larga cadena de sello cronológico en algo mucho más informativo. Esta función acepta dos argumentos: una cadena de formato y el valor de sello cronológico. La cadena de formato está conformada por una secuencia de caracteres, cada uno de los cuales tiene un significado especial. La tabla 4-2 presenta una lista con los más utilizados.

Utilizando los caracteres especiales de la tabla 4-2, es posible dar formato al sello cronológico de UNIX con la función `date()`, para personalizar la manera en que los valores de fecha y hora son presentados en pantalla. He aquí algunos ejemplos:

```
<?php
// datos de salida: "Hora y fecha actual: 12:28 pm 20 Mar 2008"
echo ' Hora y fecha actual: ' . date("h:i a d M Y",
mktime(12,28,13,3,20,2008));
```

Carácter	Significado
d	Día del mes (numérico)
D	Día de la semana (cadena de caracteres)
l	Día de la semana (cadena de caracteres)
F	Mes (cadena de caracteres)
M	Mes (cadena de caracteres)
m	Mes (numérico)
Y	Año
h	Hora (en formato de 12 horas)
H	Hora (en formato de 24 horas)
a	a.m. o p.m.
i	Minuto
s	Segundo

Tabla 4-2 Códigos de formato para la función `date()`

```
// datos de salida: " Hora y fecha actual: 8:15 14 Feb 2008"
echo ' Hora y fecha actual: ' . date("H:i d F Y",
mktime(8,15,0,2,14,2008));

// datos de salida: "La fecha de hoy es Oct-05-2007"
echo 'La fecha de hoy es ' . date("M-d-Y", mktime(0,0,0,10,5,2007));
?>
```

Funciones de fecha y hora útiles

PHP también da soporte a muchas otras funciones de manipulación para fechas y horas, que te permiten verificar si la fecha es válida o hacer conversiones entre zonas horarias. La tabla 4-3 presenta algunas de ellas.

Función	Lo que hace
`checkdate()`	Verifica si es una fecha válida
`strtotime()`	Crea sellos cronológicos para descripciones en inglés
`gmdate()`	Expresa un sello cronológico en GMT

Tabla 4-3 Funciones comunes de fecha y hora de PHP

Validar una fecha

Una función integrada de gran utilidad es `checkdate()`, que acepta combinaciones de mes, día y año y regresa un valor *true/false* (verdadero/falso) indicando si la fecha es válida o no. Examina el siguiente ejemplo, que prueba la fecha 30 de febrero de 2008.

```
<?php
// datos de salida: 'Fecha no válida'
if(checkdate(2,30,2008)) {
  echo 'Fecha válida';
} else {
  echo 'Fecha no válida';
}
?>
```

Convertir cadenas de caracteres en sellos cronológicos

Otra función PHP de gran utilidad es `strtotime()`, que acepta una cadena de caracteres que contenga una fecha o una hora y la convierte en un sello cronológico de UNIX. He aquí un ejemplo:

```
<?php
// define una cadena de caracteres que contiene un valor de fecha
// la convierte en un sello cronológico de UNIX
// aplica formato utilizando date()
// datos de salida: '07 Jul 08'
$cadena = 'July 7 2008’;
echo date('d M y', strtotime($cadena));
?>
```

Resulta interesante apuntar que la función `strtotime()` reconoce descripciones familiares de tiempo como `"now"` (ahora), `"3 hours ago"` (hace tres horas), `"tomorrow"` (mañana) o `"next Friday"` (el próximo viernes). El siguiente ejemplo ilustra tan útil característica:

```
<?php
// datos de salida: '12 Mar 09'
echo date ('d M y', strtotime('now'));

// datos de salida: '13 Mar 09'
echo date ('d M y', strtotime('tomorrow'));

// datos de salida: '16 Mar 09'
echo date ('d M y', strtotime('next Friday'));
```

```
// datos de salida: '10 Mar 09'
echo date ('d M y', strtotime('48 hours ago'));
?>
```

Traducir números de día y mes a nombres

La función `date()` abordada en la sección anterior no sólo es útil para dar formato a los sellos cronológicos y convertirlos en cadenas legibles; también sirve para encontrar el día de la semana correspondiente a una fecha determinada. Para ello, simplemente utiliza el carácter de formato `'D'` con el sello cronológico, como se ilustra en el siguiente ejemplo, que muestra el día de la semana correspondiente al 5 de octubre de 2008:

```
<?php
// datos de salida: 'Sun'
echo date ('D', mktime(0,0,0,10,5,2008));
?>
```

También puedes hacerlo para los nombres de los meses, utilizando el parámetro `'F'` de la función `date()`:

```
<?php
// muestra una lista con los nombres de los meses
// datos de salida: 'January, February, ... December'
foreach (range(1,12) as $m){
  $meses[] = date('F', mktime(0,0,0,$m,1,0));
}
echo implode($meses, ', ');
?>
```

Calcular la hora GMT (hora del Meridiano de Greenwich)

La función `gmdate()` trabaja exactamente como `date()`, salvo que a partir de la cadena con datos de fecha regresa la hora GMT en lugar de la hora local. Para verlo en acción, examina los siguientes ejemplos, que regresan el GMT equivalente a dos horas locales:

```
<?php
// muestra el GTM relativo a 'now'
echo gmdate('H:i:s d-M-Y', mktime());

// muestra el GMT relativo a '18:01 30-Nov-2007'
// datos de salida: '00:01:00 01-Dec-2007'
echo gmdate('H:i:s d-M-Y', mktime(18,1,0,11,30,2007));
?>
```

Prueba esto 4-4 Construir una calculadora de edad

Ahora que sabes un poco sobre la manera en que PHP maneja las fechas y los horarios, pongamos este conocimiento en uso con un proyecto práctico: una aplicación Web que permite escribir tu fecha de nacimiento y calcula la edad que tienes en este momento, en años y meses.

He aquí el código (*calculaedad.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 4-4: Calculadora de Edades</title>
  </head>
  <body>
    <h2>Proyecto 4-4: Calculadora de Edades</h2>
<?php
    // si el formulario no ha sido enviado
    // muestra el formulario
    if (!isset ($_POST['submit'])){
?>
    <form method="post" action="calculaedad.php">
      Escribe tu fecha de nacimiento, en formato mm/dd/aaaa: <br />
      <input type="text" name="fdn" />
      <p>
      <input type="submit" name="submit" value="Enviar" />
      </form>
<?php
    // si el formulario ha sido enviado
    // procesa los datos enviados
    } else {
      // divide el valor de la fecha en sus componentes
      $fechaArr = explode('/', $_POST['fdn']);

      // calcula el sello cronológico correspondiente al valor de la fecha
      $fechaTs = strtotime($_POST['fdn']);

      // calcula el sello cronológico correspondiente al día de hoy, 'today'
      $now = strtotime('today');

      // verifica si los datos han sido enviados con el formato correcto
      if (sizeof($fechaArr) != 3){
        die('ERROR: Por favor escriba una fecha válida');
      }
      // verifica si los datos insertados son una fecha válida
      if (!checkdate($fechaArr[0], $fechaArr[1], $fechaArr[2])) {
```

```
        die ('ERROR: Por favor escriba una fecha de nacimiento válida');
      }
      // verifica que la fecha sea anterior a hoy, 'today'
      if($fechaTs >= $now) {
        die('ERROR: Por favor escriba una fecha anterior al día de hoy');
      }

      // calcula la diferencia entre la fecha de nacimiento y el día de
hoy en días
      // convierte en años
      // convierte los días restantes en meses
      // presenta los datos de salida
      $edadDias = floor(($now - $fechaTs) / 86400);
      $edadAnos = floor($edadDias / 365);
      $edadMeses = floor(($edadDias – ($edadAnos * 365)) / 30);
      echo "Su edad aproximada es $edadAnos años y $edadMeses meses.";
    }
?>
  </body>
</html>
```

Una vez que el usuario envía su fecha de nacimiento en el formulario Web (figura 4-6), siguiendo el formato `MM/DD/AAAA`, se dispara la segunda parte del script. La primera parte del programa se concentra especialmente en buscar errores. Primero, se divide la fecha ingresada en el formulario en sus valores constituyentes de mes, día y año; la función `checkdate()` se utiliza para verificar si se trata de una fecha válida. A continuación, la función `strtotime()` se utiliza para convertir la fecha ingresada en sello cronológico de UNIX; la cadena que surge es comparada con el valor "today" (hoy) para verificar que se trata de una fecha pasada.

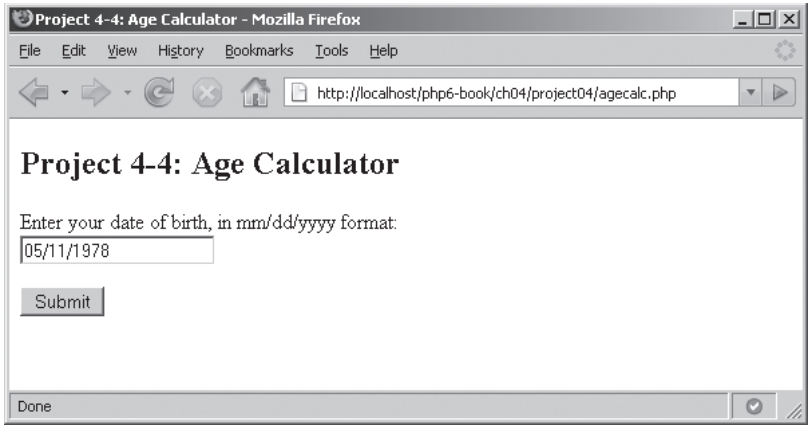


Figura 4-6 Formulario Web para que el usuario escriba su edad

(*continúa*)

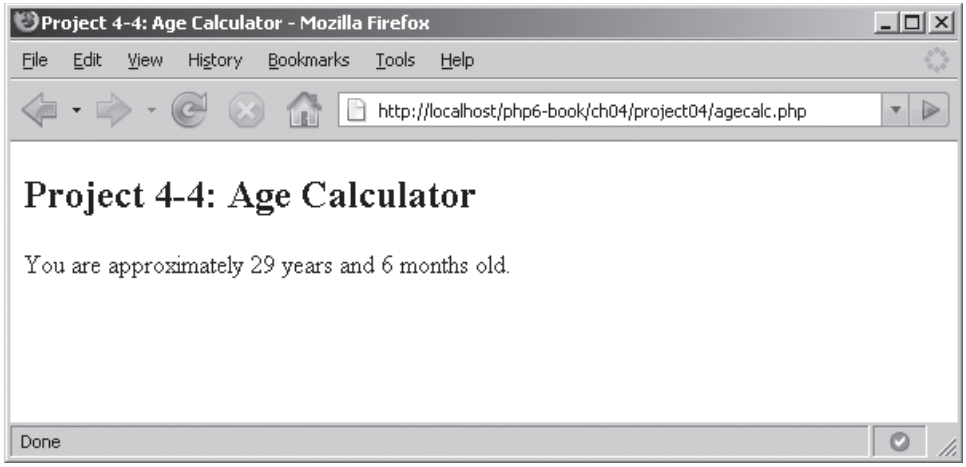


Figura 4-7 El resultado del formulario enviado muestra el cálculo de la edad

Dando por hecho que la fecha enviada aprueba la revisión de errores, el script pasa a realizar los cálculos. Primero, el sello cronológico correspondiente a "today" (hoy) se resta del sello correspondiente a la fecha de nacimiento; como ambos valores están expresados en segundos, el resultado de la operación regresa la edad del usuario en segundos. Este resultado es dividido entre 86 400, la cantidad de segundos que tiene un día, y regresa la edad del usuario en días. Al dividir el número de días entre 365 se obtiene la edad del usuario en años. Por último, la parte restante de la edad (meses) se calcula dividiendo los días restantes entre 30.

La figura 4-7 muestra el resultado después de que se envía el formulario.

Resumen

Este capítulo te presentó un nuevo tipo de variable de PHP, las matrices, que te permiten agrupar múltiples variables relacionadas y trabajar en ellas como una sola entidad. Además de enseñarte a crear, editar y procesar matrices, también te mostró algunas de las funciones de PHP para manipularlas, te enseñó la manera de utilizarlas en el contexto de los formularios Web y pusiste en práctica las matrices mediante tres aplicaciones sencillas. Por último, te explicó cómo trabajar con datos de tiempo en PHP, enseñándote las funciones de fecha y hora más comunes, que fueron utilizadas para construir una sencilla calculadora de edades.

Las matrices son una importante adición a tu caja de herramientas; a medida que tus scripts se vuelvan más complejos, comenzarás a apreciar el poder y la flexibilidad de las matrices. Para aprender más al respecto, puedes visitar las siguientes direcciones Web:

- Matrices, en **www.php.net/manual/en/language.types.array.php**
- Operadores de matrices, en **www.php.net/manual/en/language.operators.array.php**
- Funciones para manipulación de matrices, en **www.php.net/manual/en/ref.array.php**
- Los matrices especiales `$_POST` y `$_GET`, en **www.php.net/manual/en/reserved.variables.php#reserved.variables.get**
- Funciones de fecha y hora, en **www.php.net/manual/en/ref.datetime.php**

Autoexamen Capítulo 4

1. ¿Cuáles son los dos tipos de matriz PHP y en qué se diferencian?

2. Menciona las funciones que utilizarías para realizar las siguientes tareas:

A Eliminar elementos duplicados en una matriz

B Añadir un elemento al inicio de la matriz

C Invertir el orden de una matriz

D Contar la cantidad de elementos de una matriz

E Buscar un valor dentro de una matriz

F Mostrar el contenido de una matriz

G Revolver el contenido de una matriz

H Combinar dos matrices en una sola

I Encontrar los elementos comunes entre dos matrices

J Convertir una cadena de caracteres en una matriz

K Extraer un segmento de una matriz

3. ¿Cuáles serían los datos de salida del siguiente código?

```
<?php
sort(array_slice(array_reverse(array_unique(array('b','i','g','f','o',
'o','t'))), 2, 3));
?>
```

4. Utilizando únicamente una matriz y el bucle `foreach`, escribe un programa que presente los nombres de los días de la semana.
5. Escribe un programa que lea una matriz de números y regrese una lista de las cantidades menores a 15.
6. Escribe un programa que lea una matriz y regrese un mensaje indicando si la matriz contiene sólo valores únicos.

Capítulo 5

Usar funciones y clases

Habilidades y conceptos clave

- Aprender los beneficios de encapsular el código en funciones y clases
- Definir y utilizar tus propias funciones, argumentos y valores regresados
- Entender las variables locales y globales
- Aprender acerca de recursión y listas de argumentos de longitud variable
- Definir y utilizar tus propias clases, métodos y propiedades
- Comprender los conceptos de visibilidad, extensibilidad y reflejo propios de la programación orientada a objetos

Conforme tus programas de PHP se vayan haciendo más complejos, es posible que te encuentres repitiendo el mismo código varias veces para realizar la misma tarea (por ejemplo: verificar si un número es primo o revisar el campo de un formulario para saber si está vacío). En estas situaciones, tiene sentido convertir ese código en un componente reciclable, que pueda manejarse independientemente y que sea "invocado" en diferentes programas a medida que se necesite. Esta práctica no sólo reduce la cantidad de código duplicado que debes escribir, también hace que tus scripts sean más limpios, eficientes y fáciles de mantener.

PHP te ofrece dos constructores que te ayudan a reducir la duplicación de código en tus programas: *funciones definidas por el usuario*, que son segmentos de código reciclable creados por el programador para realizar tareas específicas; y las *clases*, que proporcionan un acercamiento más formal, orientado a objetos, al reciclaje y la encapsulación del código. Este capítulo te brindará una introducción a ambos constructores y presentará ejemplos prácticos para ejemplificar su empleo en la realidad.

Crear funciones definidas por el usuario

En capítulos anteriores se te han presentado muchas funciones integradas de PHP para manipular cadenas de caracteres, números y matrices. Sin embargo, PHP no restringe su uso a las funciones integradas, también te permite diseñar tus propias funciones. Estas funciones *definidas por el usuario* son una manera nítida de crear páginas independientes de código reciclable, que realizan tareas específicas y pueden recibir mantenimiento independientemente del programa principal.

Empaquetar tu código en funciones presenta cuatro ventajas importantes:

- Reduce la duplicación dentro de un programa, al permitirte agrupar rutinas de uso común en un solo componente. Esta acción presenta el beneficio secundario de reducir el tamaño de los programas, hacerlos más eficientes y fáciles de leer.
- Una función se crea una vez, pero se utiliza en muchas ocasiones, a menudo por más de un programa. Si el código de la función cambia, los cambios se realizan en un solo lugar (la definición de la función) y las invocaciones a la misma por parte de los programas permanecen intactas. Este hecho puede simplificar de manera significativa el trabajo requerido para el mantenimiento y la actualización del código, en especial cuando se compara con la opción de realizar los cambios en forma manual tras cada aparición del código que habrá de modificarse dentro del programa, sin olvidar el riesgo de cometer errores en el proceso.
- Se facilitan la depuración y la prueba del programa cuando éste se encuentra subdividido en funciones, y también se facilita el rastreo de la fuente de errores y la corrección con un mínimo impacto sobre el resto del programa.
- Trabajar con funciones alienta el pensamiento abstracto, porque empaquetar el código de un programa en funciones es, ni más ni menos, que comprender cómo puede encapsularse una tarea *específica* dentro de un componente *genérico*. En este sentido, las funciones alientan la creación de programas más robustos y extensibles.

Crear e invocar funciones

Existen tres componentes para toda función:

- *Argumentos*, que funcionan como los datos de entrada para la función.
- *Valores de retorno*, que son los datos de salida presentados por la función.
- El *cuerpo de la función*, que contiene el código que procesa los datos de entrada para transformarlos en datos de salida.

Para comenzar, veamos un ejemplo sencillo, que no utilice argumentos ni valores de retorno. Examina el siguiente script PHP, que contiene una función definida por el usuario para mostrar el día de la semana actual.

```
<?php
// definición de función
// muestra el nombre del día de la semana
function diaDeHoy(){
echo "Hoy es " . date('l', mktime());
}
// invocación de la función
diaDeHoy();
?>
```

Las definiciones de función principian con la palabra clave `function`, seguida por el nombre de la función y una lista de argumentos (opcionales) entre paréntesis. Las llaves (`{ }`) encierran el cuerpo principal de la función, el cual puede contener cualquier código PHP legal, incluidas definiciones de variables, pruebas condicionales, bucles y declaraciones de datos de salida. Los nombres de las funciones deben iniciar con una letra o un guión bajo, seguidos opcionalmente de más caracteres como letras, números o guiones bajos; no se permiten signos de puntuación ni espacios en blanco dentro de los nombres de función. En el ejemplo anterior, el nombre de la función es `diaDeHoy()` y su cuerpo contiene una sola declaración, que utiliza las funciones `echo` y `date()` para obtener y presentar el día de la semana actual.

Por supuesto, definir una función es sólo la mitad de la batalla; la otra mitad es utilizarla. En PHP invocar funciones es tan sencillo como invocarlas por su nombre (y pasar los argumentos opcionales en caso necesario). El cuerpo principal del ejemplo anterior lo hace: invoca la función por su nombre y luego se atrasa a un lado para ver los resultados.

Pregunta al experto

P: ¿Puedo invocar una función antes de definirla?

R: Sí, PHP 4.0 (y posteriores) permite que los desarrolladores invoquen funciones aunque la correspondiente definición de función aparezca más adelante en el programa. Esta función es muy útil para los programadores que prefieren colocar sus definiciones de función al final del script PHP, en lugar de hacerlo al principio, para mejorar la legibilidad.

Utilizar argumentos y valores de retorno

Una función que siempre regresa los mismos datos de salida es como una estación de radio que toca la misma canción todo el día, ¡nada útil, ni interesante! Lo que realmente te interesa es la posibilidad de cambiar la música que transmite la radio como respuesta a la información que tú proporcionas como radioescucha, y crear así un programa de complacencias bajo pedido. En PHP eso equivaldría a crear funciones que puedan aceptar valores de entrada en tiempo de ejecución y utilizar esos valores para afectar los datos de salida devueltos por la función.

Ahí es donde los *argumentos* entran en juego. Los argumentos son variables que "apartan el lugar" dentro de una función, y son reemplazadas en tiempo de ejecución por los valores proporcionados a la función desde el programa principal. Luego, el código de procesamiento dentro de la función manipula esos valores y regresa el resultado que se desea. Como los datos de entrada destinados a la función cambiarán cada vez que ésta sea invocada, los datos de salida siempre serán diferentes.

Para comprenderlo, examina el siguiente ejemplo, que define una función que acepta dos argumentos y los utiliza para realizar cálculos geométricos:

```
<?php
// definición de funciones
// calcula el perímetro de un rectángulo
// p = 2 * (l+a)
function obtenPerimetro($largo, $ancho){
  $perimetro = 2 * ($largo + $ancho);
  echo "El perímetro de un rectángulo con $largo unidades de largo y
  $ancho unidades de ancho es igual a $perimetro unidades";
}

// invocación de la función
// con argumentos
obtenPerimetro(4,2);
?>
```

La función del ejemplo anterior realiza un cálculo y luego presenta el resultado directamente en la página de salida. Pero, ¿qué sucede si quieres que la función realice su trabajo y, en lugar de presentar el resultado en pantalla, haga algo más con él? Bueno, en PHP puedes hacer que una función regrese un valor explícitamente a la declaración que la invoca, un valor como el cálculo del ejemplo. Esto se realiza utilizando la declaración `return` dentro de la función, como en el siguiente ejemplo:

```
<?php
// definición de funciones
// calcula el perímetro de un rectángulo
// p = 2 * (l+a)
function obtenPerimetro($largo, $ancho){
  $perimetro = 2 * ($largo + $ancho);
  return $perimetro;
}

// invocación de la función
// con argumentos
echo 'El perímetro de un rectángulo con 4 unidades de largo y 2 unidades
de ancho es igual a: ' . obtenPerimetro(4,2) . ' unidades';
?>
```

Puedes regresar varios valores de una función al colocarlos todos en una matriz y regresar este último. El siguiente ejemplo lo ilustra, al aceptar una frase y regresar al invocador las palabras individuales, invertidas, a manera de matriz:

```
<?php
// define función
// divide una cadena de texto en palabras
```

```
// la invierte y hace el regreso a manera de matriz
function invierteMe($frase){
   $palabras = explode(' ', $frase);
   foreach ($palabras como $k => $v){
     $palabras[$k] = strrev($v);
   }
   return $palabras;
}

// invoca la función
// datos de salida: 'aíd neuB'
echo implode(' ', invierteMe('Buen día'));

// invoca la función
// datos de salida: 'eT saírasac ogimnoc'
echo implode(' ', invierteMe('Te casarías conmigo'));
?>
```

NOTA

Cuando PHP encuentra una declaración `return` dentro de una función, detiene el proceso de la función y "regresa" el control al cuerpo principal del programa.

Establecer valores de argumentos por defecto

Como ya lo viste, los argumentos que espera recibir la función del programa principal se especifican en la lista de argumentos perteneciente a la función. Por mera conveniencia, puedes asignar valores por defecto a cualquiera o a todos estos argumentos; estos valores se utilizan en caso de que la invocación de la función carezca de algún argumento. He aquí un ejemplo, que genera una dirección de correo electrónico a partir de los argumentos nombre de usuario y dominio proporcionados a la función; en caso de que no se proporcione el argumento del dominio, se le asigna uno por defecto:

```
<?php
// define una función
// genera una dirección de correo electrónico a partir de los valores
proporcionados
function construyeDireccion($nombreusuario, $dominio = 'midominio.info') {
   return $nombreusuario . '@' . $dominio;
}

// invoca la función
// sin argumentos opcionales
// datos de salida: ' Mi dirección de correo electrónico es
juan@midominio.info'
echo 'Mi dirección de correo electrónico es ' . construyeDireccion
('juan');
```

```
// invoca la función
// con argumentos opcionales
// datos de salida: ' Mi dirección de correo electrónico es
diana@dominiobueno.net'
echo 'Mi dirección de correo electrónico es ' . construyeDireccion
('diana', 'dominiobueno.net');
?>
```

Advierte que en el primer caso, la función se invocó con un solo argumento, aunque por definición requiere dos. Sin embargo, como está presente un valor por defecto para el segundo argumento, la omisión es reemplazada automáticamente por éste; así no se generan errores.

TIP

Si sólo asignas valores por defecto a algunos de tus argumentos de función, colócalos al final de la lista de argumentos. Esto permite que PHP diferencie correctamente entre un argumento omitido que no cuenta con valores por defecto, de los argumentos omitidos que sí lo tienen.

Utilizar listas dinámicas de argumentos

Una definición de función en PHP por lo general tiene una lista fija de argumentos, cuando se sabe de antemano la cantidad de argumentos que se requerirán. Sin embargo, PHP también te permite definir funciones con las llamadas *listas de argumentos de longitud variable*, donde la cantidad de argumentos puede variar de acuerdo con la invocación de la función. Un buen ejemplo es la función que acepta una cantidad arbitraria de argumentos, los suma y regresa el promedio, como se ilustra en el siguiente ejemplo:

```
<?php
// define función
// calcula el promedio de los valores proporcionados
function calPromedio(){
  $args = func_get_args();
  $cuenta = func_num_args();
  $suma = array_sum($args);
  $prom = $suma / $cuenta;
  return $prom;
}

// invoca la función
// con 3 argumentos
// datos de salida: 6
echo CalPromedio(3,6,9);

// invoca la función
// con 8 argumentos
// datos de salida: 150
echo CalPromedio(100,200,100,300,50,150,250,50);
?>
```

Advierte que la definición de la función `calPromedio()` del ejemplo anterior no incluye una lista predefinida de argumentos; en cambio, los argumentos que recibe son recuperados en tiempo de ejecución utilizando las funciones `func_num_args()` y `func_get_args()`. La primera indica la cantidad de argumentos que serán enviados a la función principal, mientras que la segunda se encarga de enviar los valores de tales argumentos, como una matriz. Como se muestra en el ejemplo anterior, ambas resultan de gran utilidad cuando se utilizan funciones que aceptan una cantidad arbitraria de argumentos.

Comprender el ámbito de las variables

Un concepto clave en las funciones definidas por el usuario de PHP es el *ámbito de la variable*: la extensión de la visibilidad de la variable dentro del espacio del programa PHP. Por defecto, las variables utilizadas dentro de una función son locales: su impacto está restringido al espacio de esa función, y no pueden ser vistas ni manipuladas fuera de la función donde existen. Para comprenderlo, examina el siguiente ejemplo:

```
<?php
// define función
// cambia el valor de $resultado
function cambiaResultado() {
  $resultado = 25;
}

// define una variable en el programa principal
// presenta su valor
$resultado = 11;
echo 'El resultado es ' . $resultado; // datos de salida: 11

// ejecuta la función cambiaResultado()
cambiaResultado();

// presenta otra vez el valor de $resultado
echo 'El resultado es ' . $resultado; // datos de salida: 11
?>
```

Aquí, la variable `$resultado` es definida en el programa principal y la función `cambiaResultado()` contiene instrucciones para cambiar su valor; sin embargo, después de ejecutar la función, el valor de `$resultado` permanece intacto, lo que se debe a que las instrucciones de `cambiaResultado()` son locales y no se reflejan en el programa principal.

Esta "atadura" de las variables al espacio interno de la función es deliberada: mantiene el grado de separación entre el programa principal y sus funciones, y reduce las posibilidades de

que las variables del programa principal afecten a las localizadas dentro de las funciones. Sin embargo, en algunas ocasiones será necesario "importar" una variable del programa principal a una función o viceversa. Para tales casos, PHP cuenta con la palabra clave `global`: cuando se aplica a una variable dentro de una función, esta palabra clave la convierte en una variable global, es decir, la hace visible tanto dentro como fuera de la función que la contiene.

El siguiente ejemplo muestra con mayor claridad la diferencia entre el ámbito local y global de una variable:

```
<?php
// define función
// cambia el valor de $resultado
function cambiaResultado() {
  global $resultado;
  $resultado = 25;
}

// define una variable en el programa principal
// presenta su valor
$resultado = 11;
echo 'El resultado es ' . $resultado; // datos de salida: 11

// ejecuta la función cambiaResultado()
cambiaResultado();

// presenta el valor de $resultado otra vez
echo 'El resultado es: ' . $resultado; // datos de salida: 25
?>
```

En esta revisión del código original, la palabra clave `global` aplicada a la variable `$resultado`, dentro de la función `cambiaResultado()`, cambia el ámbito de la variable, lo incrementa de manera que alcanza a todo el programa. Como resultado, los cambios que se hagan a la variable dentro de la función se reflejarán en el programa principal (y viceversa). Los datos de salida comentados en el programa ilustran claramente este hecho.

Utilizar funciones recursivas

Otra idea relacionada con las funciones, un poco más compleja, es la *recursión*. Una *función recursiva* es la que se invoca a sí misma varias veces hasta que se satisface una condición. Tal función suele utilizarse para resolver cálculos complejos repetitivos, o para procesar estructuras profundamente anidadas.

Para mostrar la manera como trabaja una función recursiva, examina el siguiente ejemplo, que ocupa una de ellas para sumergirse en una matriz anidada y presentar todo su contenido:

```
<?php
// define una función recursiva
// función para imprimir todos los valores
// en una matriz anidada
function presentaValores($arr) {
  global $cuenta;
  global $salida;

  // verifica que los datos de entrada sean una matriz
  if (!is_array ($arr)){
    die('ERROR: Los datos de entrada no son una matriz');
  }

  // reitera en matriz
  // incrementa el contador 1 unidad por cada valor encontrado
  // si el valor encontrado es en sí una matriz:
  // invoca recursivamente la función para contar
  // el número de elementos en hijota matriz secundaria
  // en caso contrario:
  // añade el valor encontrado la matriz de salida
  foreach ($arr as $a){
    if (is_array($a)) {
      presentaValores($a);
    } else {
      $salida[] = $a;
      $cuenta++;
    }
  }

  // envía retorno al invocador del total contado y los valores
encontrados
  // como una matriz
  return array('total' => $count, 'valores' => $salida);
}

// define una matriz anidada
$data = array(
  'o' => array(
    'otro',
    'oso',
    'ocio'),
  't' => array(
```

```
    'té',
    'tomate',
    'tina',
    'treintaycinco' => array(
      array('treinta', 'cinco'),
      array('vingt', 'trois', array(
        'rojo' => 'barón',
        'azul' => 'sangre'
      ))
    )
  )
);

// cuenta y presenta valores de la matriz anidada
$ret = presentaValores($data);
echo $ret['total'] . ' valores encontrados: ';
echo implode(', ', $ret['valores']);
?>
```

Los datos de salida de este ejemplo serían:

```
12 valores encontrados: otro, oso, ocio, té, tomate,
tina, treinta, cinco, vingt, trois, barón, sangre
```

Lo importante de este ejemplo es la función `presentaValores()`, que acepta una matriz como argumento. Después, hace reiteraciones sobre dicha matriz utilizando el bucle `foreach`, y añade cada valor que encuentra a una matriz llamada `$salida`. Por cada valor encontrado suma 1 al contador llamado `$cuenta`. En caso de que un valor encontrado sea a su vez una matriz, la función `presentaValores()` se invoca a sí misma recursivamente para procesar hijota matriz secundaria (y cualquier otra que encuentre en niveles inferiores). Este proceso continúa hasta que ya no existen valores para procesar.

Dado que tanto `$salida` como `$cuenta` son variables globales, conservan sus valores en todo el proceso. Una vez que la función ha completado su tarea, ambas variables son empaquetadas en una sola matriz asociativa, que regresa al invocador y presenta la página con los resultados.

TIP

Aunque las funciones recursivas suelen ser la manera más rápida de resolver cálculos complejos, no son la única: casi siempre se puede realizar la misma tarea, pero de manera menos elegante, utilizando uno o más bucles.

Prueba esto 5-1 Calcular MCD y mcm

Ahora que has adquirido una cantidad considerable de conocimiento sobre las funciones definidas por el usuario en PHP, pongamos ese conocimiento a prueba con una sencilla aplicación práctica. El siguiente ejemplo invita al usuario a escribir dos números en un formulario Web para calcular el máximo común divisor (MCD) y el mínimo común múltiplo (mcm) de ambos.

NOTA

En caso de que te hayas saltado esa clase de matemáticas, el MCD es el mayor entero positivo entre el que se pueden dividir dos o más enteros sin dejar residuo; el mínimo común múltiplo de dos o más enteros positivos es el menor número entero positivo, distinto de cero, que es múltiplo de todos ellos. Por ejemplo, el MCD de 6 y 10 es 2 y el mcm de los mismos es 30. Para conocer más sobre el MCD y el mcm, incluida una descripción detallada del algoritmo euclidiano utilizado en este ejemplo, puedes consultar la dirección http:// en.wikipedia.org/wiki/Greatest_common_factor y http:// en.wikipedia.org/wiki/Least_common_multiple.

He aquí el código (*mcd_mcm.php*):

```
<?php
// define función
// obtiene el MCD de dos números
function mcd($a, $b){
  if ($b == 0){
    return $a;
  }
  else {
    return mcd($b, $a % $b);
  }
}

// define una función
// obtiene el mcm de dos números utilizando el MCD
function mcm($a, $b){
  return ($a * $b) / mcd($a, $b);
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 5-1: MCD y mcm</title>
  </head>
  <body>
    <h2>Proyecto 5-1: MCD y mcm</h2>
<?php
    // si el formulario no ha sido enviado
    // presenta formulario
```

```
    if(!isset($_POST['submit'])){
?>
    <form method="post" action="mcd_mcm.php">
      Escriba dos números enteros: <br />
      <input type="text" name="num_1" size="3" />
      <p>
      <input type="text" name="num_2" size="3" />
      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
<?php
    // si el formulario ha sido enviado
    // procesa los datos de entrada
    } else {
      $num1 = (int)$_POST['num_1'];
      $num2 = (int)$_POST['num_2'];

      // calcula y presenta el MCD y el mcm
      echo "Usted escribió los números: $num1, $num2";
      echo "<br />";
      echo "El MCD de ($num1, $num2) es " . mcd($num1, $num2);
      echo "<br />";
      echo "El mcm de ($num1, $num2) es " . mcm($num1, $num2);
    }
?>
  </body>
</html>
```

Este script genera un formulario para que el usuario escriba dos números enteros (figura 5-1).

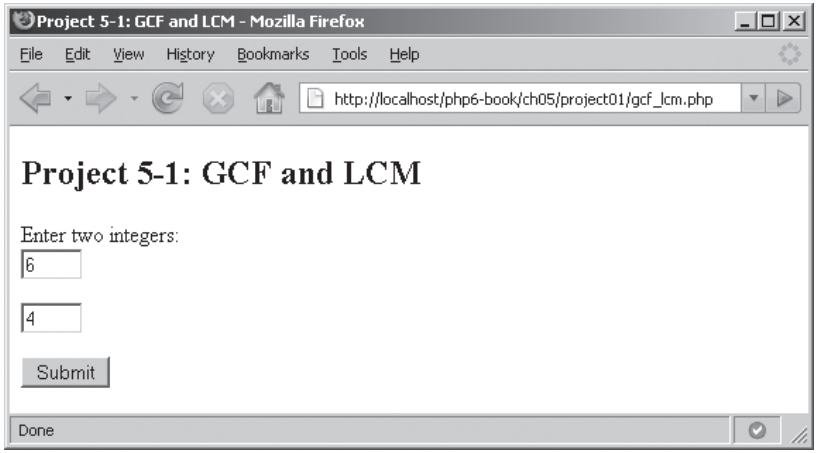


Figura 5-1 Formulario Web para ingresar dos números

(*continúa*)

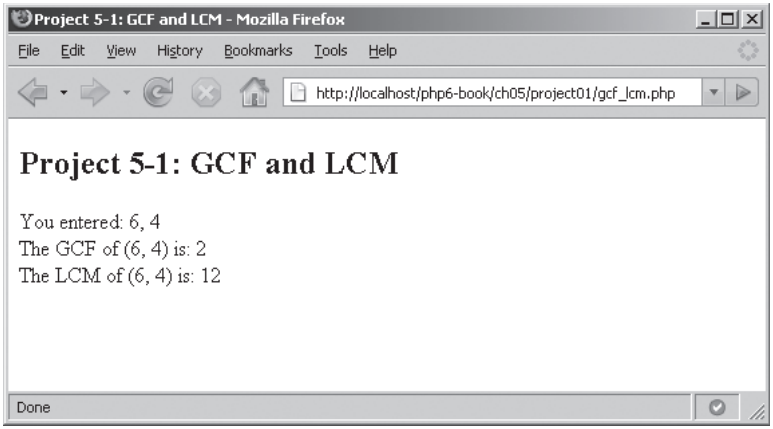


Figura 5-2 El resultado del formulario enviado, que muestra el MCD y el mcm calculados

Una vez que el formulario es enviado, los números escritos por el usuario pasan como argumentos a las funciones `mcd()` y `mcm()`, que calculan y regresan al invocador el máximo común divisor y el mínimo común múltiplo, respectivamente, de los números enviados. La figura 5-2 muestra el resultado.

Dado que este capítulo está dedicado por entero a las funciones, vale la pena ocupar unos minutos para analizar las funciones `mcd()` y `mcm()`.

1. La función `mcd()` acepta dos argumentos y calcula el MCD utilizando el algoritmo euclidiano. La descripción de este algoritmo en Wikipedia, http://en.wikipedia.org/wiki/Euclidean_algorithm, establece: "Dados dos números naturales, *a* y *b*, al menos uno de ellos diferente a cero: verificar si *b* es igual a cero; de ser así, *a* es el MCD. De lo contrario, repetir el proceso utilizando, respectivamente, *b* y el residuo de la división entre *a* y *b*."

   A partir de esta descripción, debe quedar claro que el algoritmo euclidiano puede expresarse como una función recursiva que se invoca a sí misma repetidamente, alimentando el residuo de la división previa a la siguiente, hasta que el residuo sea igual a cero. Si analizas la función `mcd()`, verás que eso es lo que hace.

2. Una vez calculado el MCD, es fácil obtener el mcm utilizando la relación `mcm (A,B) = (A*B)/MCD(A,B)`. Esta relación, y su derivación, también pueden ser consultadas en Wikipedia, http://en.wikipedia.org/wiki/Least_common_multiple, y si analizas la función `mcm()`, verás que se trata del mismo cálculo en acción. También notarás que `mcm()` invoca internamente a `mcd()`, un fino ejemplo de la cooperación entre funciones y un ejemplo de cómo las funciones pueden utilizarse para dividir largas y complejas tareas en operaciones más pequeñas y enfocadas.

Crear clases

Además de permitirte la creación de tus propias funciones, PHP también te deja agrupar funciones relacionadas utilizando una *clase*. Las clases son los constructores fundamentales detrás de la programación orientada a objetos, un paradigma de programación que requiere modelar el comportamiento del programa con "objetos", para luego utilizarlos como cimiento de tus aplicaciones.

Hasta hace poco, el soporte de PHP para la programación orientada a objetos era limitado; sin embargo, PHP 5.0 introdujo un nuevo modelo de objetos que ofrece a los programadores gran flexibilidad y fácil manejo para trabajar con clases y objetos. Aunque programación orientada a objetos suele considerarse "demasiado compleja" para los principiantes, la amplia difusión actual de la aplicación de objetos en PHP 5.1 y 5.2 significa que los aspirantes a programadores PHP no pueden darse el lujo de ignorarla.

Con esto en mente, las siguientes secciones presentarán una sencilla introducción sobre la manera en que funcionan las clases y los objetos en PHP. Invito a los lectores interesados a consultar la bibliografía sugerida al final de este capítulo para aprender más sobre el tema.

Introducción a clases y objetos

Imagina una *clase* como un ecosistema en miniatura: una colección autosuficiente e independiente de variables y funciones que trabajan juntas para realizar una o más tareas específicas (por lo general, relacionadas entre sí). Las variables dentro de las clases reciben el nombre de *propiedades* y a las funciones se les llama *métodos*.

Las clases sirven como modelos para crear *objetos*, que son instancias específicas de la clase. Cada objeto tiene propiedades y métodos que se corresponden con los de su clase principal. Cada objeto es una instancia completamente independiente, con sus propiedades y métodos específicos, y por lo mismo puede manejársele de manera independiente de otros objetos de la misma clase.

Para ponerlo en términos más concretos, considera el siguiente ejemplo: una clase Automóvil que contiene propiedades para el color y el modelo, además de métodos para aceleración, frenado y dirección. Es posible derivar dos objetos diferentes de la clase Automóvil, uno que represente un auto Ford y otro un Honda. Cada uno de estos objetos tendrá métodos para la aceleración, el frenado y la dirección, lo mismo que valores específicos para el color y el modelo. De la misma manera, es posible manipular cada uno de estos objetos de manera independiente; por ejemplo, podrías cambiar el color del Honda sin afectar al Ford, o invocar el método de aceleración del Ford sin tener impacto en el Honda.

La figura 5-3 muestra la relación entre las clases y sus objetos visualmente.

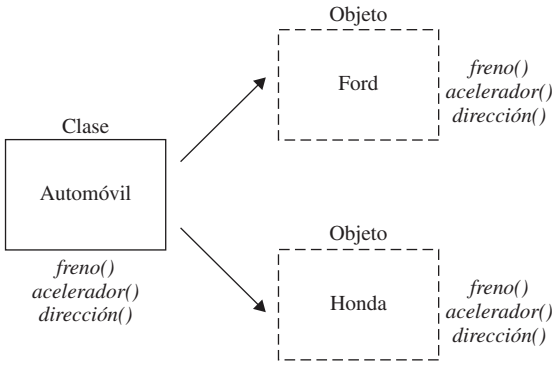


Figura 5-3 La relación entre clases y objetos

Definir y utilizar clases

En PHP las clases se definen de manera muy semejante a las funciones: una definición de clase comienza con la palabra clave `class`, seguida por el nombre de la clase y un par de llaves ({ }). La definición de clase completa debe estar encerrada dentro de las llaves; casi siempre, consta de definiciones de propiedades (variables) seguidas por métodos (funciones).

Para conocer el aspecto de una definición de clase, revisa el siguiente ejemplo; es la definición de la clase Automóvil con dos propiedades llamadas `$color` y `$modelo` y tres métodos llamados `acelerador()`, `freno()` y `direccion()`:

```
<?php
// define clase
class Automovil {

  // propiedades
  public $color;
  public $modelo;

  // métodos
  public function acelerador(){
    echo 'Aumenta la velocidad...';
  }

  public function freno(){
    echo 'Disminuye la velocidad...';
  }

  public function direccion(){
    echo 'Da una vuelta...';
  }
}
?>
```

Una vez que la clase ha sido definida, es posible crear objetos a partir de esa clase con la palabra clave `new`. Los objetos pueden acceder de manera directa a los métodos y las propiedades de la clase. He aquí un ejemplo, que crea un objeto de la clase Automóvil y lo asigna a `$carro`, después establece las propiedades del objeto e invoca sus métodos (advierte el símbolo `->` utilizado para conectar objetos a sus propiedades o métodos):

```
<?php
// crea objeto
$carro = new Automovil;

// establece las propiedades del objeto
$carro->color = 'rojo';
$carro->modelo = 'Ford Taurus';

// invoca los métodos del objeto
$carro->acelerador();
$carro->direccion();
?>
```

Para acceder a un método de clase desde la clase misma, o para cambiarlo, es necesario añadir un prefijo al método o la propiedad correspondiente; dicho prefijo es `$this`, que hace referencia a "esta" clase. Para ver su funcionamiento, examina la revisión del ejemplo pasado, que establece una propiedad de clase llamada `$velocidad` y luego la modifica desde las funciones `acelerador()` y `freno()`:

```
<?php
// define una clase
class Automovil {

  // propiedades
  public $color;
  public $modelo;
  public $velocidad = 55;

  // métodos
  public function acelerador(){
    $this->velocidad += 10;
    echo 'Aumenta la velocidad a ' . $this->velocidad . '...';
  }

  public function freno(){
    $this->velocidad -= 10;
    echo 'Disminuye la velocidad a ' . $this->velocidad . '...';
  }
```

```
  public function direccion(){
    $this->freno();
    echo 'Da una vuelta...';
    $this->acelerador();
  }
}
?>
```

Y ahora, cuando invoques estas funciones, verás los efectos de los cambios en `$velocidad`:

```
<?php
// crea objeto
$carro = new Automovil;

// invoca métodos
// datos de salida: 'Acelerador a 65...'
//                  'Freno a 55...'
//                  'Da una vuelta...'
//                  'Acelerador a 65...'
$carro->acelerador();
$carro->direccion();
?>
```

Pregunta al experto

P: ¿Puedo inspeccionar el interior de un objeto o una clase para ver su estructura?

R: Desde la versión 5.0, PHP ha incluido una muy completa API de *reflejo*, que te permite inspeccionar cualquier clase u objeto para obtener información detallada sobre sus propiedades, métodos, interfaces y constantes. Para utilizarlo, crea un objeto ReflectionClass o ReflectionObject y aplícalo a la clase u objeto al que le quieras pasar rayos X. He aquí un ejemplo:

```
<?php
// reflejo para clase
Reflection::export(new ReflectionClass('Gato'));

// reflejo para objeto
Reflection::export(new ReflectionObject('$miGato'));
?>
```

Para leer más sobre los reflejos en PHP puedes consultar www.php.net/manual/en/language.opp5.reflection.php

Prueba esto 5-2 Cifrar y descifrar texto

Ahora que conoces los aspectos básicos de clases y objetos, veamos un breve ejemplo práctico. El siguiente código define una clase llamada Jumbler, que permite a los usuarios cifrar (y descifrar) texto utilizando un sencillo algoritmo y una clave numérica proporcionada por el usuario. Mira la definición de clase (*jumbler.php*):

```
<?php
// define una clase
class Jumbler {

  // propiedades
  public $clave;

  // métodos
  // establece clave de cifrado
  public function creaClave($clave) {
    $this->clave = $clave;
  }

  // trae la clave de cifrado
  public function traeClave() {
    return $this->clave;
  }

  // cifra
  public function cifra($texto) {
    for ($x=0; $x<strlen($texto); $x++) {
      $cifra[] = ord($texto[$x]) + $this->traeClave() + ($x * $this
->traeClave());
    }
    return implode('/', $cifra);
  }

  // descifra
  public function descifra($cifra) {
    $datos = explode('/', $cifra);
    $texto = '';
    for ($x=0; $x=count($datos); $x++) {
      $texto .= chr($datos[$x] - $this->traeClave() - ($x * $this
->traeClave()));
    }
    return $texto;
  }
}
?>
```

(*continúa*)

Esta clase tiene una sola propiedad y cuatro métodos:

- La propiedad `$clave` guarda la clave numérica escrita por el usuario. Esta clave es la que se utiliza para realizar el cifrado.
- El método `creaClave()` acepta un argumento y configura la propiedad `$clave` para ese valor.
- El método `traeClave()` regresa el valor de la propiedad `$clave`.
- La función `cifra()` acepta una cadena de texto plano y la cifra utilizando la clave.
- La función `descifra()` acepta una cadena cifrada y la restaura a su estado original de texto plano utilizando la misma clave.

Unas breves palabras sobre el código de los métodos `cifra()` y `descifra()`, antes de proceder a la ejecución del ejemplo. Cuando se invoca `cifra()` con una cadena de texto plano, recorre esta cadena y calcula el valor numérico de cada carácter. El valor numérico es la suma de los siguientes elementos:

- El código ASCII del carácter, tal y como es regresado por la función `ord()`.
- La clave numérica establecida por el usuario a través del método `creaClave()`.
- El producto de la clave numérica y la posición del carácter dentro de la cadena.

Cada número regresado después del cálculo se añade a una matriz y una vez que toda la cadena ha sido procesada, se unen los elementos de la matriz en una sola cadena, separada por diagonales, lo cual se realiza con la función `implode()`.

La tabla 5-1 muestra brevemente la manera en que la palabra `'Ant'` (hormiga) se transforma en la cadena cifrada `'410/800/1151'`, utilizando el método descrito.

La rutina para descifrar invierte el proceso: primero divide la cadena de texto cifrado en números individuales utilizando las diagonales como delimitadores y los añade a una matriz. Después obtiene el carácter correspondiente a cada número restando los siguientes elementos:

- La clave numérica proporcionada por el usuario a través del método `creaClave()`; y

Carácter $c	Posición $p	ord($c)	Clave $clave	$clave * $p	Total
'A'	0	65	345	0	410
'n'	1	110	345	345	800
't'	2	116	345	690	1151

Tabla 5-1 Ejemplo de la ejecución de cifrado

- El producto de la clave numérica y la posición del carácter dentro de la cadena a partir del número, para después utilizar la función `chr()` para recuperar el carácter ASCII correspondiente al residuo. Los caracteres regresados con este proceso se unen en una sola cadena y se envían de regreso al invocador.

PRECAUCIÓN

La rutina de cifrado utilizada por la clase Jumbler es únicamente ilustrativa y muy insegura; ni se te ocurra utilizarla para cifrar datos en el mundo real.

Ahora que has entendido cómo funciona una clase, veamos cómo puede ser utilizada. El siguiente código genera un formulario para que el usuario escriba texto simple o cifrado, y una clave numérica. Después utiliza un objeto Jumbler para cifrar o descifrar. He aquí el código (*jumbler.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 5-2: Encripta Texto</title>
  </head>
  <body>
    <h2>Proyecto 5-2: Encripta Texto</h2>
<?php
    // si el formulario no ha sido enviado
    // presenta formulario
    if(!isset($_POST['submit'])){
?>
    <form method="post" action="jumbler.php">
      Escribir:
      <input type="radio" name="tipo" value="P" checked>Texto para
cifrar</input>
      <input type="radio" name="tipo" value="C">Texto cifrado </input>
      <br />
      <textarea name="texto" rows="6" cols="40" wrap="soft"></textarea>
      <p>
      Escriba la clave numérica: <br />
      <input type="text" name="clave" size="6" />
      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
<?php
    // si el formulario ha sido enviado
    // procesa los datos de entrada
    } else {
      $tipo = $_POST['tipo'];
      $texto = $_POST['texto'];
      $clave = (int)$_POST['clave'];
```

(*continúa*)

```
      // realiza cifrado o descifrado
      // presenta los datos de salida
      $j = new Jumbler;
      $j->creaClave($clave);
      if($tipo == 'C'){
        echo $j->descifra($texto);
      } else {
        echo $j->cifra($texto);
      }
    }
?>
  </body>
</html>
```

El script es muy sencillo: genera un formulario Web que contiene campos de texto para que el usuario escriba el mensaje y la clave numérica; también incluye dos botones de opción para indicar si se trata de texto simple o cifrado. Al enviar el formulario se crea un nuevo objeto de la clase Jumbler, y la clave proporcionada por el usuario queda registrada junto con el objeto mediante el método `creaClave()`. Dependiendo del tipo de mensaje escrito, se invoca uno de dos métodos del objeto: `cifra()` si es texto simple o `descifra()` si es un mensaje cifrado, y el resultado se presenta en la misma página.

La figura 5-4 muestra el formulario Web y la figura 5-5 es un ejemplo del texto cifrado que se genera después del envío.

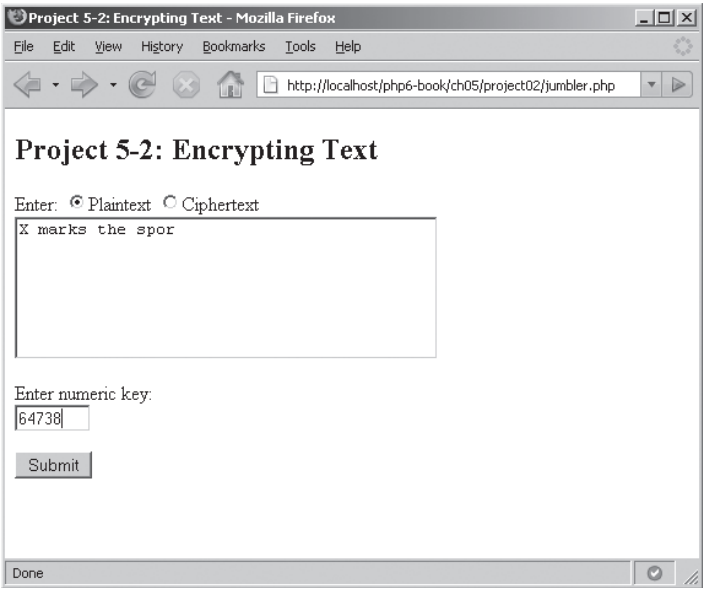


Figura 5-4 Formulario Web para escribir texto plano

Figura 5-5 El resultado del cifrado

Utilizar conceptos avanzados de programación orientada a objetos

El modelo de objetos PHP también da soporte a muchas características avanzadas adicionales, con lo que ofrece a los programadores un enorme poder y gran flexibilidad para construir aplicaciones basadas en objetos. Esta sección aborda tres de esas funciones: constructores y destructores, extensibilidad y visibilidad.

Utilizar constructores y destructores

PHP hace posible ejecutar código automáticamente cuando se crea un nuevo objeto de una clase. Para ello, utiliza un método de clase especial llamado *constructor*. También puedes ejecutar código cuando el objeto termina su acción, utilizando el método llamado *destructor*. Constructores y destructores se implementan definiendo las funciones llamadas `_construct()` y `_destruct()` dentro de la clase y colocando el código de inicio o término, según el caso, dentro de su respectivo método.

He aquí un ejemplo que muestra cómo funcionan:

```
<?php
// define clase
class Maquina {

    // constructor
    function _construct(){
        echo "Encendido...\n";
    }
```

```
    // destructor
    function _destruct(){
        echo "Apagado...\n";
    }
}

// crea un objeto
// datos de salida: "Encendido..."
$m = new Maquina();

// luego lo destruye
// datos de salida: "Apagado..."
unset ($m);
?>
```

Extender clases

Para la mayoría de los programadores, la extensibilidad es la razón más poderosa para utilizar la programación orientada a objetos. En términos sencillos, *extensibilidad* significa que una nueva clase puede derivarse de una clase existente, heredando todas las propiedades y métodos de la clase principal y añadiendo nuevas propiedades y métodos conforme se vayan necesitando. Así, por ejemplo, la clase Humano puede ser extensión de la clase Mamíferos, que a su vez es extensión de la clase Vertebrados; cada nueva extensión añade sus propias características y también hereda las de su principal.

En PHP, extender una clase es tan sencillo como añadir la palabra clave `extends` y el nombre de la clase que será extendida a la definición de la nueva clase, como en el siguiente ejemplo:

```
<?php
class Mamiferos {
  // define clase
}

class Humano extends Mamiferos {
  // define clase
}
?>
```

Con esta capacidad de extensión, todas las propiedades y métodos de la clase principal se ponen a la disposición de la clase secundaria y pueden utilizarse dentro de la lógica de programación de la clase. Para comprenderlo, examina el siguiente código:

```
<?php
// definición de la clase principal
class Mamiferos {
```

```
  public $edad;

  function _construct(){
    echo 'Crear un nuevo ' . get_class($this) . '...';
  }

  function creaEdad($edad){
    $this->edad = $edad;
  }

  function traeEdad(){
    return $this->edad;
  }

  function crece(){
    $this->edad += 4;
  }
}

// definición de la clase secundaria
class Humano extends Mamiferos {

  public $nombre;

  function _construct(){
    parent::_construct();
  }

  function creaNombre($nombre){
    $this->nombre = $nombre;
  }

  function traeNombre(){
    return $this->nombre;
  }

  function crece(){
    $this->edad += 1;
    echo 'Creciendo...';
  }
}
?>
```

Este ejemplo contiene dos definiciones de clase:

1. Mamíferos, la clase principal, que contiene la propiedad `$edad` y los métodos `creaEdad()`, `traeEdad()` y `crece()`.

2. Humano, que extiende la clase Mamíferos y hereda todas sus propiedades y métodos. Humano contiene la propiedad adicional `$nombre` y los métodos también adicionales `creaNombre()` y `traeNombre()`; además, su método `crece()` sobreescribe al que lleva el mismo nombre pero que pertenece al principal. También observa que el método `_construct()` de Humano invoca internamente al método del mismo nombre perteneciente a Mamíferos; la clave especial `parent` proporciona un fácil atajo para hacer referencia al principal de esta clase en particular.

   He aquí un ejemplo de la manera de utilizar la clase extendida:

```
<?php
$nene = new Humano;
$nene->creaEdad(1);
$nene->creaNombre('Tonka');
echo $nene->traeNombre() . ' tiene ahora ' . $nene->traeEdad() . ' años
de edad...';
$nene->crece();
$nene->crece();
echo $nene->traeNombre() . ' ahora tiene ' . $nene->traeEdad() . ' años
de edad...';
?>
```

La figura 5-6 muestra los datos de salida de este código.

A partir del resultado, debe quedar claro que a pesar de no contar explícitamente con los métodos `traeEdad()`, `creaEdad`, ni con la propiedad `$edad`, los objetos

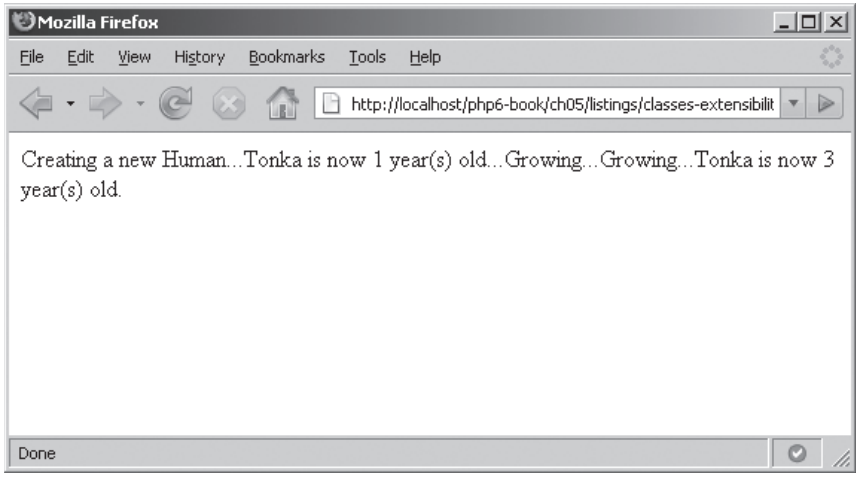


Figura 5-6 Uso de métodos de clase heredados

de la clase Humano pueden utilizarlos porque fueron heredados de la clase principal Mamíferos.

Una vez que comienzas a trabajar con herencia y extensibilidad, es fácil enredarse con largos y complejos árboles de clases. Por eso, PHP proporciona un operador `instanceof`, capaz de revisar si un objeto es instancia de una clase determinada. El operador regresa un valor *true* (verdadero) si el objeto tiene el nombre de clase en cualquier lugar de su árbol padre. He aquí un ejemplo de lo dicho en acción:

```
<?php
// árbol de clase
class Vertebrados {
}
class Mamiferos extends Vertebrados {
}
class Humano extends Mamiferos {
}

$nene = new Humano;
// datos de salida: true
echo ($nene instanceof Vertebrados) ? 'true' : 'false';
?>
```

TIP

También existe la función `get_class()`, que regresa el nombre de la clase a partir de la cual se creó un objeto, y la función `get_parent_class()`, que regresa el nombre de la clase principal.

Ajustar la configuración de visibilidad

En la primera parte de este capítulo conociste la diferencia entre el ámbito local y el global, y aprendiste que las variables utilizadas dentro de una función son invisibles al programa principal. Al trabajar con clases, PHP te permite ejercer aún más control sobre la *visibilidad* de las propiedades y métodos del objeto. Existen tres niveles de visibilidad representados por las palabras clave `public`, `protected` y `private`, que va de la mayor a la menor visibilidad, respectivamente.

Habrás notado que todas las propiedades y los métodos de los anteriores ejemplos han sido definidos con la palabra clave `public`; esto configura los métodos y propiedades de la clase como "públicos", y permite que una función invocadora los manipule directamente desde el cuerpo principal del programa. Esta condición "pública" es el nivel de visibilidad por defecto para cualquier miembro de la clase (método o propiedad) en PHP.

Si no te agrada este nivel de intromisión, puedes marcar una propiedad o método en particular como `private` (privado) o `protected` (protegido), dependiendo de la cantidad de control que quieras ceder sobre el contenido del objeto. Las propiedades y los métodos "privados" sólo son visibles dentro de la clase que los define, mientras que las propiedades y los

métodos "protegidos" son visibles para la clase que los define y para todas las clases secundarias (heredadas). Si se trata de acceder a estas propiedades y métodos fuera de su área visible por lo regular se provoca un error fatal que detiene la ejecución del script.

Para ver cómo funcionan en la práctica, examina el siguiente ejemplo:

```
<?php
// árbol de clases
class Mamiferos {
  public $nombre;
  protected $edad;
  private $especie;
}
class Humano extends Mamiferos {
}

$mamifero = new Mamifero;
$mamifero->nombre = 'William';   // ok
$mamifero->edad = 3;             // error fatal
$mamifero->especie = 'Ballena';  // error fatal

$humano = new Humano;
$humano->nombre = 'Beto';       // ok
$humano->edad = '1';            // error fatal
$humano->especie = 'Niño';      // sin definir
?>
```

Prueba esto 5-3 Generar formularios para listas de selección

Veamos ahora otro ejemplo para ilustrar algunos de los conceptos más avanzados aprendidos en la pasada sección. Aquí crearás objetos para imitar las etiquetas `<select>` y `<option>`, utilizadas en un formulario Web, escribiendo métodos que te permitan definir la lista de selección para tu programa. Hay tres objetos primarios que encontrarás en el siguiente ejemplo: un objeto base *Element* y dos objetos derivados llamados *Select* y *Option*.

Comencemos con el objeto base *Element*, que representa un elemento genérico del formulario:

```
<?php
// define clase
class Element {
  private $name;
  private $value;
  private $label;
// constructor
```

```
public function _construct(){
}

// método: crea el elemento 'name'
public function creaName($name){
  $this->name = $name;
}

// método: trae el elemento 'name'
public function traeName(){
  return $this->name;
}

// método: crea el elemento 'value'
public function creaValue($value){
  $this->value = $value;
}

// método: trae el elemento 'value'
public function traeValue(){
  return $this->value;
}

// método: crea elemento de texto 'label'
public function creaLabel($label){
  $this->label = $label;
}

// método: trae elemento 'label'
public function traeLabel(){
  return $this->label;
  }
}
?>
```

Aquí damos por hecho que cada elemento del formulario tiene tres propiedades: nombre (*name*), valor (*value*) y una etiqueta de texto (*label*). La clase define justamente estas tres propiedades y también proporciona métodos "crear" y "traer", para modificar y recuperar esas propiedades.

El objeto *Element* sirve como base para otros elementos, más específicos, como el objeto *Option*, que representa la etiqueta `<option>`. He aquí la definición:

```
<?php
// definición de la clase secundaria
class Option extends Element {

  // constructor
  public function _construct($value='', $label=''){
    parent::_construct();
```

(*continúa*)

```
    $this->creaValue($value);
    $this->creaLabel($label);
  }

  // método: datos de salida HTML para los elementos <option>
  public function render(){
    echo "<option value=\"" . $this->traeValue() . "\">" .
$this->traeLabel() . "</option>\n";
  }
}
?>
```

El constructor de la clase *Option* acepta dos argumentos: un valor y una etiqueta. Estos dos argumentos son utilizados después para establecer las propiedades `$value` y `$label` de *Option*, a través de los métodos `creaValue()` y `creaLabel()`. Advierte que estos métodos no están definidos específicamente en la clase *Option*, sino que pertenecen en realidad a la clase *Element*; sin embargo, debido a la extensibilidad, los métodos de la clase padre quedan automáticamente disponibles para sus hijos.

La clase *Option* añade además un nuevo método, `render()`, que envía el código HTML necesario para crear una etiqueta `<option>`. Advierte también que este método invoca internamente a los métodos `traeValue()` y `traeLabel()`, que también pertenecen a la clase principal, con el fin de generar el código HTML necesario.

Ahora todo lo que resta es definir la clase *Select*:

```
<?php
// define clase secundaria
class Select extends Element {

  protected $options;

  // constructor
  public function _construct(){
    parent::_construct();
    $this->options = array();
  }

  // método: añade una opción a la lista
  public function creaOption($option){
    $this->options[] = $option;
  }

  // método: regresa todas las opciones para la lista como una matriz
  private function traeOptions(){
    return (array)$this->options;
  }

  // método: envía el código HTML para la etiqueta <select>
  public function render(){
```

```
    echo $this->traeLabel() . ": <br />\n";
    echo "<select name=\"" . $this->traeName() . "\">\n";
    foreach ($this->traeOptions() as $opt){
      echo $opt->render();
    }
    echo "</select>";
  }
}
?>
```

Esta clase define una sola propiedad protegida, `$options`, como una matriz. La clase también ofrece el método `creaOption()`, que acepta un objeto *Option* y lo añade a la matriz `$options`; y el método `render()`, que itera sobre la matriz `$options` y envía el código HTML necesario para el elemento `<select>`. Los dos métodos anteriores son públicos; sin embargo, también existe el método `creaOptions()`, que es privado y sólo se utiliza internamente para regresar la matriz `$options` al invocador.

¿Cómo podrías ver estos objetos? He aquí un ejemplo:

```
<?php
      // genera una lista de selección
      $frutas = new Select();
      $frutas->creaLabel('Frutas');
      $frutas->creaName('frut_sel');
      $frutas->creaOption(new Option('Naranjas', 'Naranjas'));
      $frutas->creaOption(new Option('Fresas', 'Fresas'));
      $frutas->creaOption(new Option('Piñas', 'Piñas'));
      $frutas->creaOption(new Option('Plátanos', 'Plátanos'));
      $frutas->render();
?>
```

El primer paso es inicializar un objeto de la clase *Select* y asignarle una etiqueta y un nombre a través de los métodos `creaLabel()` y `creaName()`; recuerda que estos métodos han sido heredados de la clase principal *Element*. Después, el método `creaOption()`, perteneciente al objeto *Select*, se utiliza para asignar opciones a la lista de selección; cada invocación `creaOption()` transmite un objeto *Option* que contiene la etiqueta y el valor de la opción. Una vez que todas las opciones han sido asignadas, el método `render()`, del objeto *Select*, se encarga de escribir el código HTML de la página:

He aquí el código HTML generado por `render()`:

```
Frutas: <br />
<select name="frut_sel">
<option value="Naranjas">Naranjas</option>
<option value="Fresas">Fresas</option>
<option value="Piñas">Piñas</option>
<option value="Plátanos">Plátanos</option>
</select>
```

(*continúa*)

¿Por qué detenerse con una sola lista de selección? Lo mejor de la programación orientada a objetos, después de todo, es su capacidad para reciclar el código y, por lo mismo, es fácil generar varias listas de selección, reciclando las tres clases descritas. He aquí un ejemplo que lo ilustra (*select.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 5-3: Generar listas de selección en un formulario
</title>
  </head>
  <body>
    <h2>Proyecto 5-3: Generar listas de selección en un formulario</h2>
<?php
    // si el formulario no ha sido enviado
    // presenta formulario
    if (!isset($_POST['submit'])){
?>
    <form method="post" action="select.php">
      <?php
      // genera la lista de selección #1
      $frutas = new Select();
      $frutas->creaLabel('Frutas');
      $frutas->creaName('frut_sel');
      $frutas->creaOption(new Option('Naranjas', 'Naranjas'));
      $frutas->creaOption(new Option('Fresas', 'Fresas'));
      $frutas->creaOption(new Option('Piñas', 'Piñas'));
      $frutas->creaOption(new Option('Plátanos', 'Plátanos'));
      $frutas->render();
      ?>

      <p />

      <?php
      // genera la lista de selección #2
      $metales = new Select();
      $metales->creaLabel('Metales');
      $metales->creaName('metal_sel');
      $metales->creaOption(new Option('Acero', 'Acero'));
      $metales->creaOption(new Option('Plata', 'Plata'));
      $metales->creaOption(new Option('Oro', 'Oro'));
      $metales->creaOption(new Option('Platino', 'Platino'));
      $metales->render();
      ?>

      <p />

      <?php
      // genera la lista de selección #3
```

```
      $animales = new Select();
      $animales->creaLabel('Animales');
      $animales->creaName('animal_sel');
      $animales->creaOption(new Option('León', 'León'));
      $animales->creaOption(new Option('Hiena', 'Hiena'));
      $animales->creaOption(new Option('Zorro', 'Zorro'));
      $animales->render();
      ?>
      <p />
      <input type="submit" name="submit" value="Enviar" />
    </form>
<?php
    // si el formulario se ha enviado
    // procesa los datos de entrada
    } else {
      var_dump($_POST);
    }
?>
  </body>
</html>
```

La figura 5-7 ilustra la página generada.

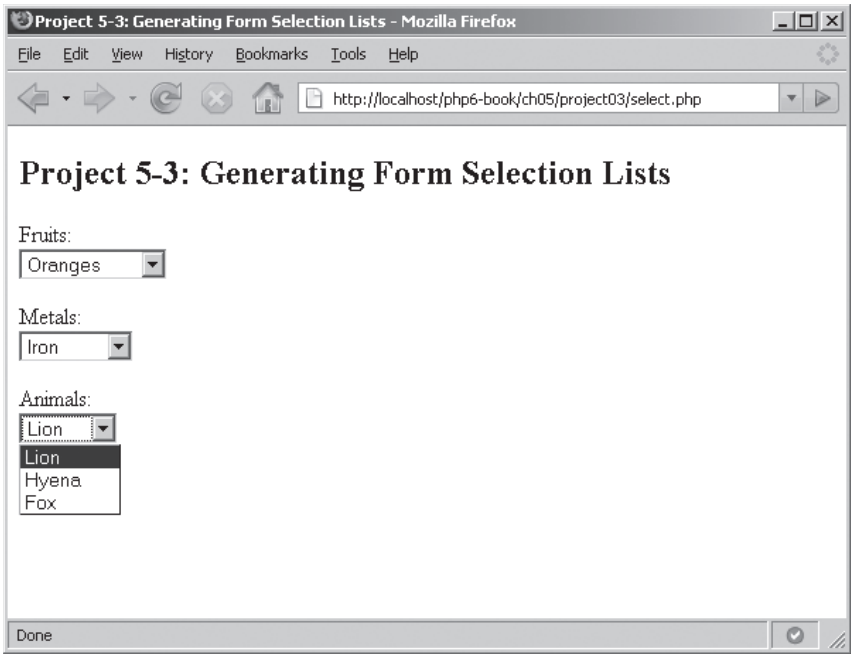


Figura 5-7 Formulario que contiene tres listas de selección generadas dinámicamente

Resumen

Este capítulo te presentó dos constructores avanzados: funciones definidas por el usuario, que te permiten empaquetar tu código PHP en bloques reciclables; y clases, que te permiten agrupar funciones relacionadas en unidades independientes y extensibles. También te ofreció un curso relámpago sobre importantes conceptos del diseño de software, como recursión, encapsulado, herencia, visibilidad y reflejo. Todos estos conceptos y técnicas fueron reforzados con el desarrollo de tres ejemplos prácticos; tal vez hayas encontrado estos ejemplos mucho más desafiantes en comparación con los de capítulos anteriores, pero con seguridad al trabajar con ellos has aumentado considerablemente tus habilidades en PHP.

Al final de este capítulo (y esta sección), ya conoces suficiente sobre la sintaxis y gramática de la programación orientada a objetos para reconocer y entender su uso en el código de terceros, y también para comenzar a escribir tus propios scripts modulares orientados a objetos. Estas habilidades te serán de gran utilidad para pasar a la siguiente sección de este libro, en que se integrarán tus aplicaciones PHP con datos provenientes de fuentes externas. Hasta entonces, dedica un tiempo a revisar los vínculos que se presentan a continuación, que ofrecen más información sobre los temas tratados en este capítulo.

- Funciones definidas por el usuario, en **www.php.net/manual/en/language.functions.php**
- Clases, en **www.php.net/manual/en/language.oop5.php**
- Visibilidad de clase, en **www.php.net/manual/en/language.oop5.visibility.php**

Autoexamen Capítulo 5

1. Escribe una ventaja de utilizar funciones.
2. ¿Cuál es la diferencia entre un argumento y un valor regresado?
3. Utilizando la relación `DISTANCIA = VELOCIDAD * TIEMPO`, escribe una función que calcule la distancia sobre una velocidad y un tiempo dados. Utiliza esta función para encontrar la distancia recorrida por una aeronave que sale de Londres, Inglaterra, a las 9:30 p.m. y llega a Bombay, India, a las 11:00 a.m. del siguiente día, suponiendo que la nave viaja a 910 km/h y que la diferencia de horarios entre Londres y Bombay es de 4.5 horas.
4. Utilizando la propiedad mcm `(P,Q,R,S) = MCD(MCD(MCD(P,Q),R),S)`, extiende el ejemplo MCD/mcm de este capítulo, de tal manera que acepte un número arbitrario de valores y regrese el MCD de la serie.
5. ¿Qué es un constructor?

6. ¿Qué sucede si intentas acceder a un método privado de una clase principal desde la clase secundaria?

7. ¿Cómo serían los datos de salida del siguiente código?

```
<?php
class Papi {
  public function habla(){
    echo get_class($this);
  }
}
class Nene extends Papi {
  public function juega(){
    parent::habla();
  }
}

$a = new Nene;
$a->juega();
?>
```

8. Utilizando la clase Select definida en este capítulo, crea una clase SeleccionarFecha que contenga tres listas de selección, una para el día, otra para el mes y otra para el año. Muestra el uso de esta clase en un formulario Web.

9. ¿Cuáles serían los datos de salida del siguiente código PHP?

```
<?php
echo date("l d F Y H:i a", mktime(12,15,22,11,17,2008));
?>
```

Parte II

Trabajar con datos de otras fuentes

Capítulo 6

Trabajar con archivos y directorios

Habilidades y conceptos clave

- Aprender a leer y escribir archivos en el disco
- Procesar directorios de manera recursiva
- Copiar, mover, cambiar nombre y borrar archivos y directorios
- Trabajar con rutas de acceso y atributos de archivos

El famoso John Donne dijo "Ningún hombre es una isla", y esta máxima se aplica también a los programas PHP. Hasta ahora, todos los ejemplos vistos han sido autosuficientes; los datos originales provienen del usuario, mediante formularios Web, o de variables escritas en el mismo código del programa. En la realidad, sin embargo, tus scripts PHP necesitarán trabajar con datos provenientes de archivos de disco, resultados de instrucciones SQL, documentos XML y muchas otras fuentes de datos.

PHP tiene muchas funciones integradas para acceder a esas fuentes de datos y este capítulo te pondrá en camino para descubrirlas, con especial énfasis en las funciones PHP para el sistema de archivos. Con más de 70 funciones disponibles, las opciones abundan; este capítulo te dará un curso relámpago sobre las más importantes, con ejemplos y proyectos prácticos para leer, escribir y manipular archivos y directorios de disco.

Leer archivos

La interfaz de programación de aplicaciones de PHP para la manipulación de archivos es muy flexible: te permite leer archivos en una cadena de texto o en una matriz, desde un sistema de archivos local o un URL remoto, por líneas, bytes o caracteres. Las siguientes secciones explican todas estas variantes con gran detalle.

Leer archivos locales

La manera más sencilla de leer el contenido de un archivo de disco en PHP es con la función `file_get_contents()`. Acepta el nombre y la ruta de acceso de un archivo de disco, y lee el archivo completo de un solo golpe en una variable de cadena. He aquí un ejemplo:

```
<?php
// lee el archivo en una cadena
$cadena = file_get_contents('ejemplo.txt') or die ('ERROR: No se
encuentra el archivo');
echo $cadena;
?>
```

Un método opcional para leer datos de un archivo es la función `file()` de PHP, que acepta el nombre y la ruta de acceso de un archivo y lo lee en una matriz, donde cada elemento de ésta corresponde a una línea del texto en el archivo. Para procesar el archivo, todo lo que necesitas hacer es iterar sobre el mismo utilizando un bucle `foreach`. He aquí un ejemplo, que lee el archivo en una matriz y luego lo despliega utilizando el bucle mencionado:

```
<?php
// lee archivo en una matriz
$matriz = file('ejemplo.txt') or die('ERROR: No se encuentra el
archivo');
foreach ($matriz as $lineaa){
  echo $lineaa;
}
?>
```

Leer archivos remotos

Tanto `file_get_contents()` como `file()` también pueden leer datos de un URL, utilizando el protocolo HTTP o FTP. He aquí un ejemplo que lee un archivo HTML de Web en una matriz:

```
<?php
// lee archivo en una matriz
$matriz = file('http://www.google.com') or die('ERROR: No se encuentra el
archivo');
foreach ($matriz as $lineaa){
  echo $lineaa;
}
?>
```

En el caso de vínculos en redes lentas, resulta más eficiente leer un archivo remoto en "fragmentos", para maximizar la eficiencia del ancho de banda disponible en la red. Para ello, utiliza la función `fgets()` con el fin de leer una cantidad específica de bytes del archivo, como en el siguiente ejemplo:

```
<?php
// lee archivo en una matriz (por fragmentos)
$cadena = '';$fp = fopen('http://www.google.com', 'r') or die('ERROR: No
se encuentra el archivo');
while (!feof($fp)) {
  $cadena .= fgets($fp,512);
}
fclose($fp);
echo $cadena;
?>
```

Este listado presenta cuatro nuevas funciones; vamos a verlas de cerca. Primero la función `fopen()`: acepta el nombre de la fuente del archivo y un argumento que indica si será abierto en modo de lectura (`'r'`), escritura (`'w'`) o apéndice (`'a'`) y luego crea un apuntador al archivo. Después, un bucle `while` invoca la función `fgets()`, que lee de manera continua e iterativa una cantidad específica de bytes del archivo y los añade a una variable de cadena; este bucle continúa hasta que la función `feof()` adquiere un valor *true* (verdadero), indicando que se ha alcanzado el final del archivo. Una vez que el bucle ha concluido, la función `fclose()` destruye el puntero del archivo.

NOTA

Para leer un URL remoto, la configuración de la variable PHP `'allow_url_fopen'` debe ser declarada con el valor *true* en el archivo de configuración *php.ini*. Si esta variable está declarada con *false*, todos los intentos por leer archivos remotos fallarán.

Leer segmentos específicos de un archivo

Una variante final se relaciona con la lectura sólo de un bloque específico de líneas a partir de cierto punto, lo que puede realizarse con una combinación de las funciones `fseek()` y `fgets()` de PHP. Examina el siguiente ejemplo, que establece una variable definida por el usuario llamada `leeBloque()` y acepta tres argumentos: el nombre del archivo, el número de línea inicial y la cantidad de líneas que serán leídas desde el punto de partida:

```
<?php
// define función
// lee un bloque de líneas de un archivo
function leeBloque($archivo, $inicio=1, $lineaas=null){
  // abre archivo
  $fp = fopen($archivo, 'r') or die('ERROR: No se encuentra el archivo');

  // inicializa los contadores
  $lineaasRevisadas = 1;
  $lineaasLeidas = 0;
  $salida = '';

  // hace bucle hasta el final del archivo
  while (!feof($fp)){
    // obtiene cada línea
    $lineaa = fgets($fp);
    // si se alcanza la posición de arranque
    // añade la línea a la variable de salida
    if ($lineaasRevisadas >= $inicio){
      $salida .= $lineaa;
      $lineaasLeidas++;
      // si el máximo de líneas está definido y es alcanzado
```

```
      // detiene el bucle
      if (!is_null($lineaasLeidas) && $lineaasLeidas == ($lineaas)){
        break;
      }
    }
    $lineaasRevisadas++;
  }
  return $salida;
}

echo leeBloque('ejemplo.txt', 3, 4);
?>
```

Dentro de `leeBloque()`, un bucle itera en el archivo línea por línea hasta alcanzar el punto de inicio (un contador de líneas llamado `$lineaasRevisadas` registra el número de línea actual, incrementando 1 en cada iteración del bucle). Una vez que se alcanza el número de la línea inicial, se leen ésta y las líneas siguientes en una variable de cadena, hasta que son procesadas todas las líneas especificadas como máximo o hasta que se alcanza el final del archivo.

Escribir archivos

La contraparte de leer datos de un archivo es escribirlos. PHP cuenta con un par de maneras para hacerlo. La primera es la función `file_put_contents()`, prima cercana de `file_get_contents()`, que conociste en la sección anterior: esta nueva función acepta el nombre del archivo y su ruta de acceso, junto con los datos que se escribirán en el archivo y luego escribe estos últimos en el lugar indicado. He aquí un ejemplo:

```
<?php
// escribe una línea en el archivo
$datos = "Un pez \n fuera del \n agua\n";
file_put_contents(salida.txt, $datos) or die('ERROR: No es posible
escribir en el archivo');
echo 'Datos escritos en el archivo';
?>
```

Si el archivo especificado en la invocación de `file_put_contents()` ya existe en el disco, la función lo sobrescribirá, como opción por defecto. Si, en cambio, lo que deseas es conservar el archivo existente y simplemente escribir nuevas líneas en él, debes añadir el marcador especial `FILE_APPEND` a la función `file_put_contents()` como tercer argumento. He aquí un ejemplo:

```
<?php
// escribe una línea en el archivo
$datos = "Un pez \n fuera del \n agua\n";
file_put_contents(salida.txt, $datos, FILE_APPEND)
```

```
 or die('ERROR: No es posible escribir en el archivo');
echo 'Datos escritos en el archivo';
?>
```

Una manera opcional para escribir datos en un archivo es crear un apuntador con `fopen()`, y luego escribir los datos en el punto especificado por el apuntador utilizando la función `fwrite()` de PHP. He aquí un ejemplo de esta técnica:

```
<?php
// abre y bloquea archivo
// escribe una cadena de texto en el archivo
// desbloquea y cierra archivo
$datos = "Un pez \n fuera del \n agua\n";
$fp = fopen('salida.txt', 'w') or die('ERROR: No es posible abrir el
archivo');
flock($fp, LOCK_EX) or die ('ERROR: No es posible bloquear el archivo');
fwrite($fp, $datos) or die ('ERROR: No es posible escribir en el
archivo');
flock($fp, LOCK_UN) or die ('ERROR: No es posible desbloquear el
archivo');
fclose($fp);
echo 'Datos escritos en el archivo';
?>
```

Pon atención a la función `flock()` del ejemplo anterior: "bloquea" un archivo antes de leerlo o escribir en él, para que ningún otro proceso tenga acceso a él. Con esta medida se reduce la posibilidad de corromper los datos, lo que puede ocurrir si dos procesos intentan escribir diferentes datos en el mismo archivo al mismo instante. El segundo parámetro para `flock()` especifica el tipo de bloqueo: `LOCK_EX` crea un bloqueo exclusivo de escritura, `LOCK_SH` crea un bloqueo no exclusivo para lectura y `LOCK_UN` destruye el bloqueo.

Pregunta al experto

P: ¿Puedo leer y escribir archivos binarios con PHP?

R: Sí, las funciones `file_get_contents()`, `file_put_contents()` y `file()`, leen y escriben datos en formato binario por defecto; esto te permite utilizar PHP con archivos binarios sin necesidad de preocuparte de que tus datos se puedan corromper.

NOTA:

Debe existir ya el directorio donde quieres guardar el archivo; de otra manera PHP generará un error fatal. Puedes verificar si un directorio existe con la función `file_exists()`, abordada más adelante en este mismo capítulo.

Prueba esto 6-1 Leer y escribir archivos de configuración

Ahora que sabes leer y escribir archivos, intentemos hacer una aplicación que utilice las funciones descritas en la sección anterior. Supongamos por un momento que estás desarrollando una aplicación de weblog, y quieres que tus usuarios puedan configurar ciertos aspectos del comportamiento de la aplicación; por ejemplo, cuántos mensajes aparecerán en la página principal o a qué correo electrónico enviarán los visitantes sus comentarios. En este caso, tal vez necesitarás construir un formulario Web que permita a los usuarios ingresar esos valores y guardarlos en un archivo de configuración que tu aplicación pueda leer cuando sea necesario.

El siguiente código genera un formulario Web con esas características, permite que los usuarios ingresen valores para aplicar diferentes configuraciones y luego guarda esta información en un archivo de disco. Cuando los usuarios vuelven a visitar el formulario, se leen los datos guardados en el archivo y se utilizan para llenar los campos del formulario.

He aquí el código (*configura.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 6-1: Leer y escribir archivos de configuración</title>
  </head>
  <body>
    <h2>Proyecto 6-1: Leer y escribir archivos de configuración</h2>
<?php
    // define el nombre del archivo de configuración y la ruta de acceso
    $configFile = 'config.ini';

    // si el formulario no ha sido enviado
    // envía formulario
    if (!isset($_POST['submit'])){

      // establece una matriz con parámetros por defecto
      $datos = array();
      // correo electrónico del Administrador
      $datos['AdminEmailAddress'] = null;
      // autor de la página
      $datos['DefAuthor'] = null;

      // número de mensajes en la página principal
      $datos['NumPosts'] = null;
      // número de comentarios
      $datos['NumComments'] = null;
```

(*continúa*)

```
      // dirección Web para notificar la colocación de nuevos mensajes
      $datos['NotifyURL'] = null;

      // lee los actuales datos de configuración
      // los utiliza para llenar los campos del formulario
      if(file_exists($configFile)) {
        $lineas = file($configFile);
        foreach ($lineas as $linea){
          $matriz = explode('=', $linea);
          $i = count($matriz) - 1;
          $datos[$matriz[0]] = $matriz[$i];
        }
      }
?>
    <form method="post" action="configura.php">
      Dirección de correo electrónico del administrador: <br />
      <input type="text" size="50" name="data[AdminEmailAdress]"
value="<?php echo $datos['AdminEmailAddress']; ?>"/>
      <p>
      Nombre del autor por defecto: <br />
      <input type="text" name="data[DefAuthor]" value="<?php echo
$datos['DefAuthor']; ?>"/>
      <p>
      Número de mensajes en la página de inicio: <br />
      <input type="text" size="4" name="data[NumPosts]" value="<?php echo
$datos['NumPosts']; ?>"/>
      <p>
      Número de comentarios anónimos: <br />
      <input type="text" size="4" name="data[NumComments]" value="<?php
echo $datos['NumComments']; ?>"/>
      <p>
      URL para la notificación automática de nuevos mensajes: <br />
      <input type="text" size="50" name="data[NotifyURL]" value="<?php
echo $datos['NotifyURL']; ?>"/>
      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
<?php
    // si el formulario ha sido enviado
    // procesa los datos de entrada
    } else {
      // lee los datos enviados
      $config = $_POST['data'];

      // valida los datos enviados conforme sea necesario
     if((trim($config['NumPosts']) != '' && (int)$config['NumPosts'] <=0)
||  (trim($config['NumComments'])  !=  ''  &&  (int)$config['NumComments']
<= 0)){
        die('ERROR: Por favor escriba un número válido');
      }
```

```
      // abre y bloquea el archivo de configuración para escritura
      $fp = fopen($configFile, 'w+') or die ('ERROR: No es posible abrir
el archivo de configuración para escritura');
      flock($fp, LOCK_EX) or die('ERROR: No es posible bloquear el
archivo de configuración para escritura');

      // escribe cada uno de los valores de configuración en el archivo
      foreach ($config as $clave => $valor) {
        if(trim($valor) != ''){
          fwrite($fp, "$clave=$valor\n") or die('ERROR: No es posible
escribir [$clave] en el archivo de configuración');
        }
      }
      // cierra y guarda el archivo
      flock($fp, LOCK_UN) or die ('ERROR: No es posible desbloquear el
archivo');
      fclose($fp);
      echo 'Los datos de configuración han sido escritos con éxito en el
archivo.';
    }
?>
  </body>
</html>
```

Este ejemplo muestra un uso común y práctico de las funciones de archivo de PHP: leer y escribir archivos de configuración en el contexto de una aplicación Web. La figura 6-1 muestra el formulario creado por el script.

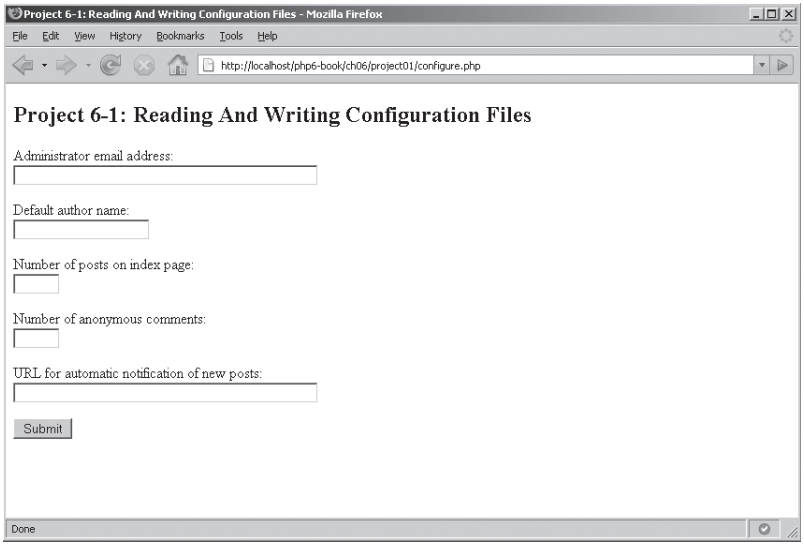


Figura 6-1 Formulario Web para insertar los datos de configuración

(*continúa*)

Una vez que el formulario ha sido enviado, los datos insertados llegan en forma de una matriz asociativa, cuyas claves corresponden a la configuración de variables en uso. Luego, estos datos son validados y se abre un apuntador en el archivo de configuración *config.ini*. A continuación, un bucle `foreach` itera sobre la matriz, escribiendo claves y valores en la posición marcada por el puntero de archivo en formato `clave=valor`, donde cada par clave-valor ocupa una línea independiente. El apuntador al archivo se cierra después, guardando los datos en el disco.

Así se vería el archivo *config.ini* después de enviar los datos que aparecen en el formulario de la figura 6-1:

```
AdminEmailAddress=admin@abc.com
DefAuthor=Charles W
numPosts=8
numComments=4
```

Si el usuario vuelve a visitar el formulario, el script verifica primero si existe un archivo llamado *config.ini* en el directorio en uso. En caso positivo, lee las líneas del archivo en una matriz con la función PHP `file()`; después un bucle `foreach` procesa la matriz, divide cada línea donde aparece el signo de igual (=) y convierte el resultado de los pares clave-valor en una matriz asociativa. Esta matriz se utiliza luego para rellenar por anticipado los campos del formulario, asignando el valor correspondiente al atributo '`value`' de cada campo.

La figura 6-2 presenta uno de estos formularios, llenado con anticipación con los datos que lee del archivo de configuración.

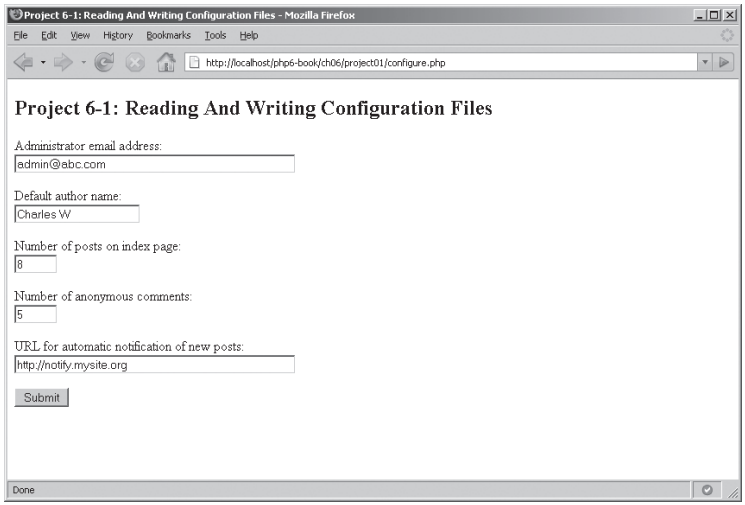


Figura 6-2 Formulario llenado con anticipación con datos provenientes del archivo de configuración

Por supuesto, los usuarios tienen toda la libertad de volver a enviar el formulario con nuevos datos; como ya se explicó, al volver a enviarlo, los datos son reescritos y el archivo de configuración queda actualizado con la información más reciente.

Procesar directorios

PHP también permite que los programadores trabajen con directorios en el sistema de archivos, iterando su contenido o recorriendo el árbol de los mismos. Iterar en un directorio es tan sencillo como invocar el objeto PHP DirectoryIterator, como en el siguiente ejemplo, que utiliza este objeto para leer un directorio y presentar una lista de los archivos que contiene:

```
<?php
// inicializa directorio con el nombre
// del directorio que procesa
$dit = new DirectoryIterator('.');

// recorre el directorio en bucle
// presenta los nombres de los archivos encontrados
while ($dit->valid()){
  if (!$dit->isDot()){
    echo $dit->getFilename() . "\n";
  }
  $dit->next ();
}

unset($dit);
?>
```

Aquí, se inicializa un objeto DirectoryIterator con un nombre de directorio, y se utiliza el método `rewind()`, perteneciente al objeto, para enviar el apuntador interno a la primera entidad en el directorio. Un bucle `while`, que se ejecuta mientras exista una entidad `valid()`, se utiliza para iterar sobre el directorio. Con el método `getFilename()` se recuperan los nombres de cada archivo individual, mientras que el método `isDot()` se utiliza para filtrar las entidades del directorio actual (.) y el directorio principal (..). El método `next()` mueve el apuntador interno hacia la siguiente entidad.

Puedes realizar la misma tarea con el bucle `while` y algunas funciones PHP para la manipulación de directorios, como se muestra en el siguiente ejemplo:

```
<?php
// crea apuntador de directorios
$dp = opendir('.') or die ('ERROR: No es posible abrir el directorio');

// lee el contenido del directorio
// presenta nombres de archivo encontrados
while ($archivo = readdir($dp)) {
```

```
  if ($archivo != '.' && $archivo != '..') {
    echo "$archivo \n";
  }
}

// destruye el apuntador de directorios
closedir($dp);
?>
```

Aquí, la función `opendir()` regresa el apuntador al directorio mencionado en la invocación de la misma función. La función `readdir()` utiliza después este apuntador para iterar sobre el directorio, regresando una sola entidad cada vez que es invocado. A partir de ahí, es fácil filtrar los directorios `.` y `..`, e imprimir los nombres de las entidades restantes. Una vez realizada esta tarea, la función `closedir()` cierra el apuntador.

TIP

También ten en mente la función `scandir()` de PHP, que acepta un nombre de directorio y regresa una matriz que contiene una lista de los archivos que incluye, además de sus respectivos tamaños. A partir de ahí es fácil procesar la matriz con un bucle `foreach`.

En algunos casos será necesario que proceses no sólo los directorios del primer nivel, sino también los subdirectorios y los directorios dentro de éstos. Por lo general, la mejor herramienta para alcanzar este objetivo suele ser una función recursiva, como las que aprendiste en el capítulo 5. Examina el siguiente ejemplo, que ilustra una de estas funciones en acción:

```
<?php
// define una función
// imprime nombres de archivos de un directorio
// y sus directorios hijo
function printDir($dir, $depthStr='+'){
  if (file_exists($dir)){
    // crea apuntador de directorio
    $dp = opendir($dir) or die ('ERROR: No es posible abrir el
directorio');

    // lee el contenido del directorio
    // presenta los nombres de los archivos encontrados
    // se invoca a sí mismo reiteradamente si encuentra subdirectorios
    while ($archivo = readdir($dp)){
      if ($archivo != '.' && $archivo != '..'){
        echo "$depthStr $dir/$archivo \n";
        if(is_dir("$dir/$archivo")){
          printDir("$dir/$archivo", $depthStr.'+');
        }
      }
     }
    // cierra apuntador de directorio
```

```
    closedir($dp);
  }
}

// presenta contenido del directorio
// y sus hijos
if (file_exists('.')) {
  echo '<pre>';
  printDir('.');
  echo '</pre>';
}
?>
```

La función `printDir()` en el código puede parecer compleja, pero es muy sencilla. Acepta dos argumentos: el nombre del directorio de nivel superior que se utilizará, y una "cadena de profundidad" (`$depthStr`), la cual indica, por sangrado, la posición de un archivo o directorio en particular dentro de la jerarquía. Con estos datos de entrada, la función abre un apuntador hacia el directorio mencionado y comienza a procesarlo con `readdir()`, presentando el nombre de cada directorio o archivo que encuentra. En caso de que localice un directorio, la cadena de profundidad incrementa un carácter y la función `printDir()` se llama a sí misma para procesar ese subdirectorio. El proceso continúa hasta que no queda ningún subdirectorio o archivo por procesar.

La figura 6-3 muestra un ejemplo de los datos de salida generados por el programa.

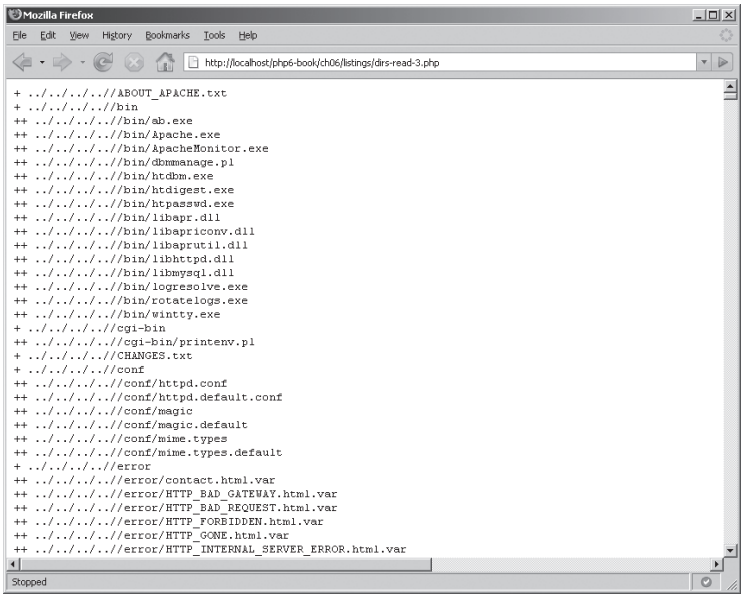


Figura 6-3 Lista de directorio con sangrías

Realizar otras operaciones de archivos y directorios

Además de las herramientas que ya estudiaste en secciones anteriores, PHP cuenta con una amplia gama de funciones para la manipulación de archivos, que te permiten verificar los atributos de archivo; copiar, mover y borrar archivos; además de trabajar con rutas de acceso y extensiones. La tabla 6-1 muestra una lista con las funciones más importantes de esta categoría.

Verificar si existe un archivo o un directorio

Si intentas añadir algo a un archivo inexistente, por lo regular PHP generará un error fatal. Lo mismo sucede cuando intentas acceder, leer o escribir, desde o hacia un directorio inexistente. Para evitar estos mensajes de error, siempre debes verificar si el archivo o directorio al que estás tratando de acceder en realidad existe; para ello se utiliza la función PHP `file_exists()`. El siguiente ejemplo la presenta en acción:

Función	Lo que hace
`file_exists()`	Comprueba la existencia de un directorio o un archivo
`filesize()`	Muestra el tamaño de un archivo en bytes
`realpath()`	Muestra la ruta de acceso absoluta de un archivo
`pathinfo()`	Muestra una matriz de información sobre el archivo y su ruta de acceso
`stat()`	Proporciona información sobre los atributos y permisos de un archivo
`is_readable()`	Prueba si el archivo tiene permisos de lectura
`is_writable()`	Prueba si el archivo tiene permisos de escritura
`is_executable()`	Prueba si el archivo es ejecutable
`is_dir`	Prueba si la entidad actual es un directorio
`is_file`	Prueba si la entidad actual es un archivo
`copy()`	Copia un archivo
`rename()`	Cambia el nombre de un archivo
`unlink()`	Borra un archivo
`mkdir()`	Crea un nuevo directorio
`rmdir()`	Elimina un directorio
`include() / require()`	Lee un archivo externo dentro del actual script PHP

Tabla 6-1 Funciones PHP comunes para directorios y archivos

```
<?php
// verifica archivo
if (file_exists('ciertoarchivo.txt')){
  $cadena = file_get_contents('ciertoarchivo.txt');
} else {
  echo 'El archivo no existe. ';
}

// verifica el directorio
if (file_exists('ciertodirectorio')) {
  $archivos = scandir('ciertodirectorio');
} else {
  echo 'El directorio no existe.';
}
?>
```

Calcular el tamaño del archivo

Para calcular el tamaño de un archivo en bytes, utiliza la función `filesize()` con el nombre del archivo como argumento:

```
<?php
// calcula el tamaño del archivo
// datos de salida: 'El tamaño del archivo es 1327 bytes.'
if (file_exists('ejemplo.txt')) {
  echo 'El tamaño del archivo es de ' . filesize('ejemplo.txt') . '
bytes.';
} else {
  echo 'El archivo no existe. ';
}
?>
```

Encontrar la ruta de acceso absoluta de un archivo

Para obtener la ruta de acceso absoluta de un archivo, utiliza la función `realpath()`, como se muestra a continuación:

```
<?php
// obtiene la ruta de acceso de un archivo
// datos de salida: 'Ruta de acceso: /usr/local/apache/htdocs/
//                   /php-book/ch06/listings/ejemplo.txt'
if (file_exists('ejemplo.txt')) {
  echo 'Ruta de acceso: ' . realpath('ejemplo.txt');
} else {
  echo 'El archivo no existe. ';
}
?>
```

Pregunta al experto

P: ¿Las funciones de archivo de PHP son sensibles a las mayúsculas respecto a los nombres de archivo?

R: Depende del sistema de archivos que los contenga. Windows no es sensible a las mayúsculas en nombres de archivo, por lo que las funciones de archivo de PHP tampoco lo son cuando se ejecutan en ese sistema operativo. Sin embargo, los nombres de archivo en los sistemas *NIX sí lo son, por lo que las funciones PHP ejecutadas en ellos también son sensibles a las mayúsculas. En este sentido, con un archivo llamado *eJEMplo.txt*, la declaración `<?php file_exists('ejemplo.txt'); ?>` tal vez regrese un valor *false* (falso) si el script se ejecuta en un sistema *NIX, pero *true* (verdadero) sí se ejecuta en Windows.

También puedes ejecutar la función `pathinfo()`, que regresa una matriz que contiene ruta de acceso, nombre y extensión del archivo. He aquí un ejemplo:

```
<?php
// obtiene la información de la ruta de acceso del archivo
if(file_exists('ejemplo.txt')) {
  print_r(pathinfo('ejemplo.txt'));
} else {
  echo 'El archivo no existe. ';
}
?>
```

Recuperar atributos de archivo

Puedes obtener información detallada sobre un archivo específico, como a quién pertenece, permisos, modificaciones y horas de acceso, con la función PHP `stat()`, que regresa esta información como una matriz asociativa. He aquí un ejemplo:

```
<?php
// obtiene información del archivo
if(file_exists('ejemplo.txt')) {
  print_r(stat('ejemplo.txt'));
} else {
  echo 'El archivo no existe. ';
}
?>
```

Puedes comprobar si un archivo tiene permisos de lectura, de escritura o si es ejecutable con las funciones `is_readable()`, `is_writable()` e `is_executable()`, respectivamente. El siguiente ejemplo muestra su uso:

```
<?php
// obtiene información del archivo
```

```
// datos de salida: 'El archivo tiene: permisos de lectura  permisos de
escritura'
if (file_exists('ejemplo.txt')) {
  echo ' El archivo tiene: ';
  // verificar el bit de lectura
  if (is_readable('ejemplo.txt')){
    echo ' permisos de lectura ';
  }
  // verificar el bit de escritura
  if (is_writable('ejemplo.txt')){
    echo ' permisos de escritura ';
  }
  // verificar el bit de ejecución
  if (is_executable('ejemplo.txt')){
    echo ' es ejecutable ';
  }
} else {
  echo 'El archivo no existe. ';
}
?>
```

La función `is_dir()` regresa un valor *true* (verdadero) si el argumento que se le pasa es un directorio, mientras que la función `is_file()` regresa un valor *true* (verdadero) si el argumento que se le transmite es un archivo. He aquí un ejemplo:

```
<?php
// comprueba si el archivo es un directorio
if (file_exists('ejemplo.txt')) {
  if (is_file('ejemplo.txt')){
    echo 'Es un archivo.';
  }
  if (is_dir('ejemplo.txt')) {
    echo 'Es un directorio.';
  }
} else {
  echo 'ERROR: el archivo no existe.';
}
?>
```

Crear directorios

Para crear un directorio nuevo y vacío, invoca la función `mkdir()` con la ruta de acceso y el nombre del directorio que será creado:

```
<?php
if (!file_exists('mydir')) {
  if (mkdir('mydir')) {
    echo 'El directorio fue creado con éxito.';
```

```
  } else {
    echo 'ERROR: El directorio no pudo ser creado.';
  }
} else {
  echo 'ERROR: El directorio ya existe.';
}
?>
```

Copiar archivos

Puedes copiar un archivo de un lugar a otro invocando la función PHP `copy()`, con el nombre del archivo de origen y la ruta de acceso destino como argumentos. He aquí un ejemplo:

```
<?php
// copia un archivo
if (file_exists('ejemplo.txt')) {
  if (copy('ejemplo.txt', 'ejemplo.new.txt')) {
    echo 'El archivo fue copiado con éxito.';
  } else {
    echo 'ERROR: El archivo no pudo ser copiado.';
  }
} else {
  echo 'ERROR: El archivo no existe.';
}
?>
```

Es importante hacer notar que si el archivo ya existe en el destino, la función `copy()` lo sobreescribirá.

Cambiar nombre a archivos o directorios

Para cambiar nombre a un archivo (o un directorio) o moverlo, invoca la función `rename()` con la ruta de acceso antigua y nueva como argumentos. He aquí un ejemplo que cambia el nombre de un archivo y un directorio, además de mover el archivo a un lugar diferente en el proceso:

```
<?php
// cambia el nombre del archivo y lo mueve
if (file_exists('ejemplo.txt')) {
  if (rename('ejemplo.txt', '../ejemplo.nuevo.txt')) {
    echo 'Se cambió el nombre del archivo correctamente.';
  } else {
    echo 'ERROR: No pudo cambiarse el nombre del archivo.';
  }
} else{
  echo 'ERROR: El archivo no existe.';
}
```

```
// cambia el nombre del directorio
if (file_exists('midir')) {
  if (rename('midir', 'miotrodir')) {
    echo 'Se cambió el nombre del directorio correctamente.';
  } else {
    echo 'ERROR: No pudo cambiarse el nombre del directorio.';
  }
} else {
  echo 'ERROR: El directorio no existe.';
}
?>
```

Lo mismo que `copy()`, si el archivo de destino ya existe, la función `rename()` lo reescribirá.

PRECAUCIÓN

PHP te permitirá copiar, borrar, cambiar nombre, crear y manipular archivos y directorios sólo si el usuario "propietario" del script tiene los privilegios necesarios para realizar esas tareas.

Eliminar archivos o directorios

Para eliminar un archivo, envía el nombre del mismo y su ruta de acceso a la función `unlink()` de PHP, como en el siguiente ejemplo:

```
<?php
// elimina archivo
if (file_exists('dummy.txt')) {
  if (unlink('dummy.txt')) {
    echo 'El archivo fue eliminado con éxito.';
  } else {
    echo 'ERROR: El archivo no pudo ser eliminado';
  }
} else {
  echo 'ERROR: El archivo no existe.';
}
?>
```

Para eliminar un directorio vacío, PHP ofrece la función `rmdir()`, que realiza la función inversa de `mkdir()`. Pero si el directorio no está vacío, es necesario eliminar primero todo su contenido (incluidos los subdirectorios) y sólo después de eso puede invocarse la función `rmdir()` para eliminar el directorio. Puedes realizar esta operación manualmente, pero una función recursiva suele ser más eficiente. He aquí un ejemplo que demuestra la manera de eliminar un directorio y todos sus directorios anidados:

```
<?php
// define función
// elimina todos los archivos de un directorio
function removeDir($dir) {
  if (file_exists($dir)) {
    // crea apuntador de directorio
    $dp = opendir($dir) or die ('ERROR: No es posible abrir el
directorio');

    // lee el contenido del directorio
    // borra los archivos encontrados
    // se invoca a sí mismo reiteradamente si encuentra directorios
    while ($archivo = readdir($dp)) {
      if ($archivo != '.' && $archivo != '..') {
        if (is_file("$dir/$archivo")) {
          unlink("$dir/$archivo");
        } else if (is_dir("$dir/$archivo")){
          removeDir("$dir/$archivo");
        }
      }
    }

    // cierra apuntador de directorio
    // elimina los directorios ahora vacíos
    closedir($dp);
    if (rmdir($dir)) {
      return true;
    } else {
      return false;
    }
  }
}

// elimina directorio y todos sus directorios anidados
if (file_exists('midir')) {
  if (removeDir('midir')) {
    echo 'El directorio fue eliminado correctamente.';
  } else {
    echo 'ERROR: El directorio no pudo ser eliminado.';
  }
} else {
  echo 'ERROR: El directorio no existe.';
}
?>
```

En este caso, la función `removeDir()` es recursiva y acepta un argumento: el nombre del directorio de nivel superior que será eliminado. La función comienza por crear un apuntador hacia el directorio con `opendir()` para luego iterar sobre su contenido con un bucle

`while`; ya estudiaste esta misma técnica en una sección anterior de este capítulo. Por cada directorio encontrado, se utilizan los métodos `is_file()` e `is_dir()` para determinar si la entidad encontrada es un archivo o un subdirectorio; en caso de ser un archivo, se utiliza la función `unlink()` para eliminarlo; en caso de ser un directorio, se invoca de manera recursiva la función `removeDir()` para realizar el mismo proceso con el contenido de los subdirectorios. El proceso continúa hasta que no queda ningún archivo ni directorio; en este punto, el directorio de nivel superior está vacío y puede ser eliminado rápidamente con `rmdir()`.

Leer y evaluar archivos externos

Para leer y evaluar archivos externos desde tu script PHP, utiliza las funciones `include()` o `require()`. Una aplicación muy común de estas funciones es incluir un encabezado estándar (*header*), un pie de página (*footer*) o una nota de derechos de autor (copyright) en todas las páginas Web de un sitio. He aquí un ejemplo:

```
<!DOCTYPE html PUBLIC "-// W3C// DTD XHTML 1.0 Transitional // EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http:// www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title></title>
  </head>
  <body>
    <?php require('encabezado.php'); ?>
    <p/>
    Éste es el cuerpo de la página.
    <p/>
    <?php include('pie.php'); ?>
  </body>
</html>
```

En este caso el script lee dos archivos externos, *encabezado.php* y *pie.php*; coloca el contenido de ambos en el lugar donde las funciones `include()` o `require()` los invocan. Es importante resaltar que cualquier código PHP que vaya a ser evaluado dentro de los archivos incluidos de esta manera debe ir entre las etiquetas `<?php ... ?>`.

Pregunta al experto

P: ¿Cuál es la diferencia entre `include()` y `require()`?

R: Muy sencillo: una función `include()` perdida generará una alerta, pero la ejecución del script continuará; por otra parte, una función `require()` perdida generará un error fatal que detendrá la ejecución del script.

Prueba esto 6-2 Crear una galería de fotografías

Ahora presentaremos otro ejemplo práctico; en él utilizarás lo que aprendiste sobre la manipulación de directorios, además de diferentes funciones. El siguiente programa examina un directorio específico en busca de fotografías y genera dinámicamente una página Web donde muestra esas imágenes, lo que sería una galería digital de imágenes. He aquí el código (*galeria.php*):

```
<!DOCTYPE html PUBLIC "-// W3C// DTD XHTML 1.0 Transitional // EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http:// www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 6-2: Crea una galería de imágenes</title>
  </head>
  <style type="text/css">
  ul {
        list-style-type: none;
  }

  li {
      float: left;
      padding: 10px;
      margin: 10px;
      font: bold 10px Verdana, sans-serif;
  }

  img {
      display: block;
      border: 1px solid #333300;
      margin-bottom: 5px;
      }
  </style>
  <body>
    <h2>Proyecto 6-2: Crea una galería de imágenes</h2>
    <ul>
<?php
    // define la ubicación de las imágenes
    // debe ser una ubicación accesible para el dueño del script
    $dirFotos = './fotos';

    // define cuáles extensiones de archivo son imágenes
    $extFotos = array('gif', 'jpg', 'jpeg', 'tif', 'tiff', 'bmp', 'png');

    // inicializa la matriz para conservar los nombres de archivo de las
imágenes encontradas
    $listaFotos = array();
```

```
    // lee el contenido del directorio
    // construye una lista de fotos
    if (file_exists($dirFotos)) {
      $dp = opendir($dirFotos) or die ('ERROR: No es posible abrir el
directorio');
      while ($archivo = readdir($dp)){
        if($archivo != '.' && $archivo != '..') {
          $datosArchivo = pathinfo($archivo);
          if (in_array($datosArchivo['extension'], $extFotos)) {
            $listaFotos[] = "$dirFotos/$archivo";
          }
        }
      }
      closedir($dp);
    } else {
      die ('ERROR: El directorio no existe.');
    }

    // itera sobre la lista de fotos
    // muestra cada foto y el nombre de archivo
    if (count($listaFotos) > 0) {
      for ($x=0; $x<count($listaFotos); $x++) {
?>
        <li>
          <img height="150" width="200"
            src="<?php echo $listaFotos[$x]; ?>" />
          <?php echo basename($listaFotos[$x]); ?><br/>
          <?php echo round(filesize($listaFotos[$x])/1024) . 'KB'; ?>
        </li>
<?php
      }
    } else {
      die('ERROR: No se encontraron imágenes en el directorio.');
    }
?>
    </ul>
  </body>
<html>
```

Este programa comienza definiendo la ruta de acceso al directorio que contiene las imágenes; necesitarás modificarlo para que apunte hacia un directorio de tu equipo local, de modo que el programa funcione. También establece una matriz llamado `$extFotos` que contiene las extensiones de imágenes más comunes; esta información resulta útil cuando se buscan archivos de imágenes en el directorio seleccionado.

Una vez establecidas ambas variables, el programa utiliza las funciones `opendir()` y `readdir()` para realizar reiteraciones sobre el directorio especificado y procesar cada enti-

(*continúa*)

dad dentro de él. Si la extensión de una de estas entidades coincide con un elemento de `$extFotos`, se añaden el archivo y su ruta de acceso a otra matriz, esta última llamada `$listaFotos`. El proceso continúa hasta que se procesan todas las entidades que se encuentran en el directorio señalado.

Suponiendo que se ha encontrado por lo menos una imagen, el siguiente (y último) paso es generar el código HTML para mostrar las imágenes en la página Web. Esto se realiza reiterando sobre `$listaFotos` con un bucle `for` y generando una etiqueta `<img />` por cada entidad encontrada, junto con el nombre de la imagen y su tamaño en kilobytes. De esta manera, la imagen a la que se hace referencia se forma como un elemento dentro de una lista horizontal HTML, con el fin de que aparezcan bien acomodadas en la página final.

La figura 6-4 muestra la salida de este programa. Como se percibe, es muy sencillo utilizar las funciones PHP de archivos y directorios para hacer reiteraciones en un directorio y procesar los archivos encontrados en él.

Figura 6-4 Galería de fotos Web generada dinámicamente

Resumen

Leer y escribir archivos son tareas básicas que debe conocer todo programador PHP; este capítulo abordó ambas tareas y mucho más. Comenzó con una demostración de cómo leer archivos locales y remotos, lo mismo que crear archivos nuevos y añadirles datos a los existentes. Después pasó a la manipulación de directorios con PHP y explicó muchas funciones del lenguaje aplicables a directorios y archivos, ilustrando cada una de ellas con ejemplos breves. Los conocimientos teóricos se aplicaron en dos proyectos prácticos: leer y escribir en un archivo de configuración y crear una galería de imágenes en una página Web.

Para conocer más sobre los temas abordados en este capítulo, visita los siguientes vínculos:

- Trabajar con archivos y directorios, en **www.melonfire.com/community/columns/trog/article.php?id=208** y **www.melonfire.com/community/columns/trog/article.php?id=211**
- Funciones del sistema de archivos, en **www.php.net/manual/en/ref.filesystem.php**
- Funciones de directorios, en **www.php.net/manual/en/ref.dir.php**

✓ *Autoexamen Capítulo 6*

1. Menciona las funciones que debes utilizar para

A Obtener la ruta de acceso absoluta de un archivo.

B Borrar un archivo.

C Recuperar el tamaño de un archivo.

D Leer el contenido de un archivo en una matriz.

E Verificar si el archivo tiene permisos de lectura.

F Verificar si el archivo existe.

G Leer el contenido de un directorio en una matriz.

H Escribir una cadena de texto en un archivo.

I Crear un nuevo directorio.

J Eliminar un directorio.

K Crear un apuntador a un directorio.

L Verificar si una entidad dentro de un directorio es un archivo.

2. ¿Por qué utilizarías una función `flock()`?

3. Escribe un script PHP para contar el número de líneas de un archivo.

4. Escribe un script PHP para leer un archivo, invertir su contenido y escribir el resultado en un nuevo archivo.

5. Escribe un script PHP que entre a un directorio y renombre todos los archivos con extensión *.txt* por la extensión *.xtx*.
6. Utilizando como modelo la función recursiva `removeDir()` que aparece en este capítulo, escribe una función recursiva para copiar el contenido de un directorio a otro.
7. Escribe un script PHP que lea el directorio actual y regrese una lista de sus archivos ordenada cronológicamente, de acuerdo con las últimas modificaciones hechas. (Pista: La función PHP para obtener las últimas modificaciones en el tiempo es `filemtime()`).

Capítulo 7

Trabajar con bases de datos y SQL

Habilidades y conceptos clave

- Aprender conceptos sobre bases de datos y el lenguaje estructurado de consultas (SQL, Structured Query Language)
- Añadir, editar, borrar y ver registros utilizando las bases de datos MySQL y SQLite
- Recuperar registros de bases de datos con PHP
- Validar y guardar datos de entrada del usuario en una base de datos con PHP
- Escribir programas portátiles sustentados por bases de datos

Una de las razones de la popularidad de PHP como lenguaje de creación de scripts para Web es su amplio soporte a diferentes bases de datos. Este soporte facilita que los desarrolladores Web creen sitios sustentados en bases de datos y que se hagan nuevos prototipos de aplicaciones Web de manera rápida y eficiente, sin demasiada complejidad.

PHP soporta más de quince diferentes motores de bases de datos, incluidos Microsoft SQL Server, IBM DB2, PostgreSQL, MySQL y Oracle. Hasta PHP 5, este soporte se proporcionaba mediante extensiones nativas de las bases de datos, cada una con sus propias características y funciones; sin embargo, esto dificultaba a los programadores el cambio de una base a otra. PHP 5 rectificó esta situación introduciendo una API común para el acceso a base de datos: las extensiones de objetos de datos de PHP (PDO, *PHP Data Objects*), que proporcionan una interfaz unificada para trabajar con bases de datos y ayudan a que los desarrolladores manipulen diferentes bases de datos de manera consistente.

En la versión 5.3 de PHP, las extensiones PDO han sido mejoradas, con soporte para más motores de bases de datos y mejoras considerables en la seguridad y el desempeño. Para fines de compatibilidad con versiones anteriores, se sigue dando soporte a las extensiones de bases de datos nativas. Dado que en muchas ocasiones tendrás que escoger entre una extensión nativa (que puede ser más veloz u ofrecer más características) y una PDO (que ofrece portabilidad y consistencia para diferentes motores de bases de datos), en este capítulo se abordan las dos opciones: te presenta una introducción sobre PDO y también explica el funcionamiento de las dos extensiones nativas más populares de PHP: las extensiones mejoradas de MySQL y las extensiones de SQLite.

Introducción a bases de datos y SQL

En la era de Internet, la información no se acumula ya en gabinetes; en cambio, se almacena como 1 y 0 digitales en bases de datos electrónicas, que son los “contenedores” de datos que

imponen cierta estructura en la información. Estas bases de datos electrónicas no sólo ocupan menos espacio físico que sus contrapartes de madera y metal; también contienen herramientas para ayudar a los usuarios a filtrar y recuperar información rápidamente utilizando diversos criterios. En particular, la mayoría de las bases de datos en la actualidad son *relacionales*, las cuales le permiten al usuario definir relaciones entre las tablas que conforman la base para realizar búsquedas y análisis más efectivos.

Actualmente hay disponibles muchos sistemas de administración de bases de datos; algunos son comerciales, otros gratuitos. Tal vez hayas oído mencionar algunos: Oracle, Microsoft Access, MySQL y PostgreSQL. Estos sistemas de bases de datos son aplicaciones poderosas, con muchas características, capaces de organizar y realizar búsquedas entre millones de registros a grandes velocidades; por ello son muy utilizados por empresas privadas y oficinas gubernamentales, en muchas ocasiones para llevar a cabo tareas de misión crítica.

Antes de comenzar con los vaivenes de la manipulación de registros con PHP, es esencial tener un claro entendimiento de los conceptos primordiales de las bases de datos. Si eres nuevo en el tema, las siguientes secciones te proporcionarán una base y también te permitirán comenzar a trabajar con ejercicios prácticos de SQL. Esta información será de utilidad para comprender el material más avanzado en secciones posteriores.

Comprender las bases de datos, registros y llaves primarias

Toda base de datos está compuesta por una o más *tablas*. Estas tablas, que estructuran los datos en filas y columnas, imponen una organización de datos. La figura 7-1 muestra una tabla típica.

El ejemplo contiene cifras por ventas de varias localidades, donde cada fila, también llamada *registro*, tiene información sobre diferentes lugares y años. Cada registro, a su vez, está dividido en columnas, también llamadas *campos*, y cada campo contiene un fragmento diferente de información. Esta estructura tabular facilita la búsqueda de registros, dentro de la tabla, que coincidan con ciertos *criterios*, por ejemplo: todos los lugares cuyas ventas sean mayores a $10 000, o las ventas de todos los lugares realizadas en el año 2008. Los registros

ID	*Año*	*Ubicación*	*Ventas* ($)
1	2007	Dallas	9 495
2	2007	Chicago	8 574
3	2007	Washington	12 929
4	2007	Nueva York	13 636
5	2007	Los Ángeles	8 748
6	2007	Boston	3 478
7	2008	Dallas	15 249
8	2008	Chicago	19 433
9	2008	Washington	3 738
10	2008	Nueva York	12 373
11	2008	Los Ángeles	16 162
12	2008	Boston	4 745

Figura 7-1 Tabla de ejemplo

en la tabla no tienen ningún orden en particular; pueden organizarse alfabéticamente, por año, por total de ventas, por lugar o cualquier otro criterio que selecciones para especificar su orden. Así las cosas, para facilitar la identificación de un registro en particular, se hace necesario añadir un atributo identificador único para cada registro, como un número en serie o un código de secuencia. En el ejemplo anterior, cada registro es identificado por un campo "ID registro" único; este campo es conocido como la *llave primaria* de la tabla.

TIP

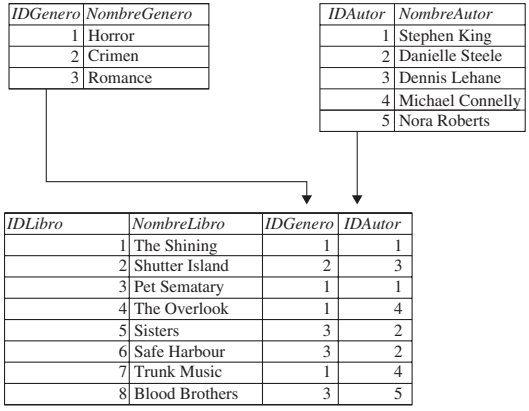
Una manera fácil de comprender estos conceptos es con una analogía. Imagina que la base de datos es una biblioteca, y que cada tabla es un anaquel dentro de la biblioteca. Así, un registro sería el equivalente electrónico de un libro en el anaquel y el título del libro sería su llave primaria. Sin el título sería imposible distinguir fácilmente un libro de otro. (La única manera de hacerlo así sería abrir cada libro y revisar su contenido, proceso que consumiría mucho tiempo, ¡mismo que la llave primaria nos ahorra!)

Una vez que tienes información en una tabla, por lo general querrás utilizarla para responder ciertas preguntas, por ejemplo: ¿cuántos lugares vendieron más de $5 000 en 2008? A estas preguntas se les conoce como *consultas*, y a las preguntas respondidas por la base de datos a esas consultas se les conoce como *colecciones de resultados*. Las consultas se formulan utilizando el lenguaje estructurado de consultas (*SQL*).

Comprender relaciones y llaves externas

Ya sabes que una sola base de datos puede contener varias tablas. En las bases de datos relacionales, las tablas pueden vincularse entre sí por uno o más campos que compartan en común, llamados *llaves externas*. Éstas hacen posible la creación de relaciones uno-a-uno o uno-a-varios entre diferentes tablas, además de combinar datos de diversas tablas para crear colecciones de resultados más complejos.

Para comprenderlo mejor, examina la figura 7-2, que muestra tres tablas vinculadas.



IDGenero	*NombreGenero*
1	Horror
2	Crimen
3	Romance

IDAutor	*NombreAutor*
1	Stephen King
2	Danielle Steele
3	Dennis Lehane
4	Michael Connelly
5	Nora Roberts

IDLibro	*NombreLibro*	*IDGenero*	*IDAutor*
1	The Shining	1	1
2	Shutter Island	2	3
3	Pet Sematary	1	1
4	The Overlook	1	4
5	Sisters	3	2
6	Safe Harbour	3	2
7	Trunk Music	1	4
8	Blood Brothers	3	5

Figura 7-2 Relaciones entre tablas

La figura 7-2 muestra tres tablas, que contienen información sobre autores (tabla A), géneros (tabla G) y libros (tabla B), respectivamente. Las tablas G y B son claras: contienen una lista de géneros y nombres de autores, respectivamente, con cada registro identificado con una llave primaria única. La tabla B es un poco más compleja: cada libro de la tabla está vinculado con un género específico mediante la llave primaria perteneciente al género (tabla G) y a un autor mediante la llave primaria de autor (tabla A).

Siguiendo estas llaves hasta sus respectivas tablas fuente, es fácil identificar el autor y género de un libro específico. Por ejemplo, se puede ver que el libro "The Shining" fue escrito por "Stephen King" y pertenece al género "Horror". De manera similar, comenzando por el punto opuesto, se puede ver que el autor "Michael Connelly" ha escrito dos libros: "The Overlook" y "Trunk Music".

Relaciones como las que aparecen en la figura 7-2 son el fundamento de las bases de datos relacionales. Vincular tablas utilizando llaves externas es también más eficiente que su alternativa: crear una sola tabla que incluya hasta el nombre del perico, para almacenar toda la información, puede parecer conveniente a primera vista; pero actualizar una tabla así exige siempre una tarea manual (y propensa a errores) para localizar cada elemento de un valor específico y reemplazarlo con un nuevo valor. Dividir la información en tablas independientes y vincularlas con llaves externas asegura que una pieza de información aparece una, y sólo una vez, en la base de datos. Con esto se eliminan las redundancias, se simplifican las búsquedas (al estar localizables en un solo lugar), y se hace que la base sea más compacta y manejable.

Al proceso de afinar la base de datos definiendo y aplicando relaciones uno-a-uno y uno-a-varios entre las tablas que la integran se le conoce como *normalización de la base de datos*, y es una tarea clave que enfrentan los ingenieros informáticos cuando crean una nueva base de datos. En el proceso de normalización, el ingeniero también identifica relaciones cruzadas y dependencias incorrectas entre tablas; también optimiza la organización de los datos con el fin de que las consultas SQL se ejecuten a su máxima eficiencia. Existen *formas de normalización* disponibles que te ayudan a probar hasta qué punto está normalizada tu base de datos; estas normas proporcionan una guía útil para ayudarte a estar seguro de que el diseño de tu base tiene una estructura consistente y eficiente por igual.

Comprender las declaraciones SQL

El lenguaje estructurado de consultas, SQL, es el lenguaje estándar utilizado para comunicarse con la base de datos, añadir o cambiar registros y privilegios de usuario, y para realizar consultas. Casi todos los RDBMS comerciales utilizan en la actualidad este lenguaje, que se convirtió en estándar ANSI en 1989.

Las declaraciones SQL caen en alguna de las siguientes categorías:

- **Lenguaje de definición de datos (DLL, Data Definition Language)** Consta de declaraciones que definen la estructura y las relaciones de la base de datos y sus tablas. Por lo

general, estas declaraciones se utilizan para crear, borrar y modificar bases de datos y tablas; especificar nombres de campos y tipos; así como establecer índices.

- **Lenguaje de manipulación de datos (DML, Data Manipulation Language)** Estas declaraciones alteran o extraen datos de una base; son utilizadas para añadir y borrar registros. También se ocupan para realizar consultas, recuperar registros de una tabla que coincidan con uno o más criterios especificados por el usuario, y unir tablas utilizando los campos que comparten.
- **Lenguaje para el control de datos (DCL, Data Control Language)** Estas declaraciones se utilizan para definir acceso a diferentes niveles y privilegios de seguridad en la base de datos. Utilizarás estas declaraciones para otorgar o negar privilegios a los usuarios, asignar funciones, cambiar claves de acceso, ver permisos y crear colecciones de resultados para proteger el acceso a los datos.

Los comandos SQL imitan la lengua inglesa, lo que facilita su aprendizaje. La sintaxis también es muy intuitiva: cada declaración SQL inicia con una "palabra de acción", como DELETE (borrar), INSERT (insertar), ALTER (alterar) o DESCRIBE (describir) y termina con un punto y coma (;). Los espacios en blanco, tabuladores y retornos de carro son ignorados. A continuación aparecen algunos ejemplos de declaraciones válidas SQL:

```
CREATE DATABASE biblioteca;
SELECT película FROM películas WHERE calificación > 4;
DELETE FROM autos WHERE año_de_manufactura < 1980;
```

La tabla 7-1 muestra la sintaxis de algunas declaraciones SQL comunes, con su respectiva explicación.

Declaración SQL	Lo que hace
`CREATE DATABASE` *`nombre de la base`*	Crea una nueva base de datos
`CREATE TABLE` *`nombre de la tabla`* `(`*`campo1, campo2, ...`*`)`	Crea una nueva tabla
`INSERT INTO` *`nombre de la tabla`* `(campo1, campo2, ...) VALUES (`*`valor1, valor2, ...`*`)`	Inserta un nuevo registro en una tabla con valores específicos
`UPDATE` *`nombre de la tabla`* `SET` *`campo1=valor1, campo2=valor2, ...`* `[WHERE condición]`	Actualiza registros en una tabla con nuevos valores
`DELETE FROM` *`nombre de la tabla`* `[WHERE condición]`	Borra registros de una tabla
`SELECT` *`campo1, campo2, ...`* `FROM` *`nombre de la tabla`* `[WHERE condición]`	Recupera los registros de una tabla que coinciden con los criterios de búsqueda
`RENAME` *`nombre de la tabla`* `TO` *`nuevo nombre de la tabla`*	Cambia el nombre de una tabla
`DROP TABLE` *`nombre de la tabla`*	Borra una tabla
`DROP DATABASE` *`nombre de la base`*	Borra una base de datos

Tabla 7-1 Declaraciones SQL comunes

Pregunta al experto

P: ¿Qué tanto soporte tiene SQL?

R: SQL es un estándar tanto ANSI como ISO, y está ampliamente respaldado por todas las bases de datos de compilación estándar SQL. Es decir, muchos proveedores de bases de datos cuentan también con "estándares" SQL extendidos con sus propias extensiones, para ofrecer a sus clientes un conjunto mejorado de características o un mejor rendimiento. Tales extensiones varían de proveedor a proveedor, y es posible que las declaraciones SQL que las utilizan no funcionen de la misma manera en todas las bases de datos. Por lo tanto, siempre es conveniente revisar la documentación de la base de datos con la que trabajes, para saber si ofrece extensiones particulares y cómo afectan las declaraciones SQL que escribas.

Prueba esto 7-1 Crear y alimentar una base de datos

Ahora que conoces los fundamentos, hagamos un ejercicio práctico que te debe familiarizar con el uso de las bases de datos y SQL. En esta sección utilizarás la línea de comando interactiva de cliente de MySQL para crear una base de datos y tablas, añadir y editar registros y generar colecciones de resultados que coincidan con varios criterios.

NOTA

A lo largo del siguiente ejercicio las palabras en negritas indican las instrucciones que debes escribir en la línea de comandos de MySQL. Las instrucciones, también llamadas "comandos", pueden escribirse en mayúsculas o minúsculas. Antes de comenzar con el ejercicio, asegúrate de haber instalado, configurado y probado la base de datos MySQL, de acuerdo con las instrucciones que aparecen en el apéndice A de este libro.

Comienza por iniciar la consola cliente de MySQL y conectarla a la base de datos con tu nombre de usuario y contraseña:

```
shell> mysql -u user -p
Contraword: ******
```

Si todo resulta bien, verás un mensaje de bienvenida y el indicador interactivo SQL, como el siguiente:

```
mysql>
```

Ahora puedes ingresar declaraciones SQL en este indicador. Las declaraciones serán transmitidas al servidor MySQL y ejecutadas en éste, y el resultado será mostrado en las líneas posteriores al indicador. Recuerda terminar cada declaración con un punto y coma.

(*continúa*)

Crear la base de datos

Como todas las tablas se almacenan en una base de datos, el primer paso consiste en crear una de éstas, utilizando la declaración `CREATE DATABASE`:

```
mysql> CREATE DATABASE musica;
Query OK, 1 row affected (0.05 sec)
```

A continuación, selecciona esta base de datos recién creada como la que se usará por omisión para todos los futuros comandos; para ello utiliza la declaración `USE`:

```
mysql> USE musica;
Database changed
```

Añadir tablas

Una vez que has inicializado tu base de datos, es hora de añadirle algunas tablas. El comando SQL para realizar esta tarea es la declaración `CREATE TABLE`, que requiere un nombre de tabla y una descripción detallada de sus campos. He aquí un ejemplo:

```
mysql> CREATE TABLE artistas (
    -> artista_id INT(4) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    -> artista_nombre VARCHAR (50) NOT NULL,
    -> artista_pais CHAR (2) NOT NULL
    -> );
Query OK, 0 rows affected (0.07 sec)
```

Esta declaración crea una tabla llamada *artistas* con tres campos: *artista_id*, *artista_nombre* y *artista_pais*. Advierte que cada nombre de campo va seguido por una *declaración de tipo*; esta declaración identifica el tipo de dato que almacenará el campo, ya sea una cadena de caracteres, numérico, temporal o booleano. MySQL soporta diferentes tipos de datos y los más importantes están resumidos en la tabla 7-2.

En el anterior ejemplo, hay otras pocas indicaciones (*modificadores*) adicionales que se establecen en la tabla:

- El modificador `NOT NULL` asegura que el campo no pueda recibir valores NULL después de cada definición de campo.
- El modificador `PRIMARY KEY` marca el campo correspondiente a la llave primaria de la tabla.
- El modificador `AUTO_INCREMENT`, que sólo es aplicable a campos numéricos, le indica a MySQL que genere automáticamente valores para este campo cada vez que se inserta un nuevo registro en la tabla, incrementando en 1 el valor previo.

Tipo de campo	Descripción
INT	Tipo numérico que puede aceptar un rango de valores de –2147483648 a 2147483647
DECIMAL	Tipo numérico que soporta valores de punto flotante y decimales
DATE	Campo de fecha con el formato AAAA-MM-DD
TIME	Campo de tiempo en formato HH:MM:SS
DATETIME	Tipo que combina fecha y hora en formato AAAA-MM-DD HH:MM:SS
YEAR	Campo específico para años que acepta un rango de 1901 a 2155; en formatos AAAA o AA
TIMESTAMP	Tipo sello cronológico en formato AAAAMMDDHHMMSS
CHAR	Tipo cadena de caracteres con un tamaño máximo de 255 caracteres y longitud fija
VARCHAR	Tipo cadena de caracteres con un tamaño máximo de 255 caracteres y longitud variable
TEXT	Tipo cadena de caracteres con un tamaño máximo de 65535 caracteres
BLOB	Tipo binario para datos variables
ENUM	Tipo cadena de caracteres que acepta un valor de una lista de posibles valores definida con anterioridad
SET	Tipo cadena de caracteres que acepta cero o más valores de un conjunto de posibles valores definido con anterioridad

Tabla 7-2 Tipos de datos MySQL

Ahora sigamos adelante para crear otras dos tablas utilizando estas declaraciones SQL:

```
mysql> CREATE TABLE ratings (
    -> rating_id INT(2) NOT NULL PRIMARY KEY,
    -> rating_nombre VARCHAR (50) NOT NULL
    -> );
Query OK, 0 rows affected (0.13 sec)

mysql> CREATE TABLE canciones (
    -> cancion_id INT(4) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    -> cancion_titulo VARCHAR(100) NOT NULL,
    -> ex_cancion_artista INT(4) NOT NULL,
    -> ex_cancion_rating INT(2) NOT NULL
    -> );
Query OK, 0 rows affected (0.05 sec)
```

(*continúa*)

Añadir registros

Añadir registros a la tabla es tan sencillo como invocar la declaración INSERT con los valores apropiados. He aquí un ejemplo, que añade un registro a la tabla *artistas* especificando valores para los campos *artista_id* y *artista_nombre*:

```
mysql> INSERT INTO artistas (artista_id, artista_nombre, artista_pais)
    -> VALUES ('1', 'Aerosmith', 'US');
Query OK, 1 row affected (0.00 sec)
```

Recordarás, de la sección anterior, que el campo *artista_id* fue marcado con el modificador `AUTO_INCREMENT`. Ésta es una extensión MySQL sobre el SQL estándar, que indica a MySQL que asigne un valor automáticamente a este campo en caso de quedar sin especificación. Para verlo en acción trata de añadir otro registro utilizando la siguiente declaración:

```
mysql> INSERT INTO artistas (artista_nombre, artista_pais)
    -> VALUES ('Abba', 'SE');
Query OK, 1 row affected (0.00 sec)
```

De manera similar, añade algunos registros a la tabla *ratings*:

```
mysql> INSERT INTO ratings (rating_id, rating_nombre) VALUES (4, 'Bueno');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO ratings (rating_id, rating_nombre) VALUES (5, 'Excelente');
Query OK, 1 row affected (0.00 sec)
```

Y algunos a la tabla *canciones*:

```
mysql> INSERT INTO canciones (cancion_titulo, ex_cancion_artista,
     ex_cancion_rating) -> VALUES ('Janie\'s Got a Gun', 1, 4);
Query OK, 1 row affected (0.04 sec)
mysql> INSERT INTO canciones (cancion_titulo, ex_cancion_artista,
     ex_cancion_rating) -> VALUES ('Crazy', 1, 5);
Query OK, 1 row affected (0.00 sec)
```

Advierte que los registros en la tabla *canciones* están vinculados a los registros de la tabla *artistas* y *ratings* por llaves externas. En la siguiente sección verás en acción este tipo de relaciones por llave externa.

NOTA

El archivo de código de este libro tiene una lista completa de declaraciones SQL `INSERT` para llenar las tres tablas utilizadas en este ejercicio. Ejecuta estas declaraciones y termina de construir las tablas antes de pasar a la siguiente sección.

Ejecutar consultas

Una vez que los datos están en la base, es hora de hacer algo con ellos. SQL te permite buscar registros que coinciden con ciertos criterios específicos utilizando la declaración `SELECT`. He aquí un ejemplo que regresa todos los registros de la tabla *artistas*:

```
mysql> SELECT artista_id, artista_nombre FROM artistas;
+------------+----------------+
| artista_id | artista_nombre |
+------------+----------------+
|          1 | Aerosmith      |
|          2 | Abba           |
|          3 | Timbaland      |
|          4 | Take That      |
|          5 | Girls Aloud    |
|          6 | Cubanismo      |
+------------+----------------+
6 rows in set (0.00 sec)
```

En casi todos los casos querrás añadir filtros a tu consulta, para reducir el tamaño de la colección de resultados y asegurar que contiene sólo los registros que coinciden con ciertos criterios. Esto se hace añadiendo la cláusula `WHERE` a la declaración `SELECT` junto con una o más expresiones condicionales. He aquí un ejemplo que muestra sólo los artistas de Estados Unidos:

```
mysql> SELECT artista_id, artista_nombre FROM artistas
    -> WHERE artista_pais = 'US';
+------------+----------------+
| artista_id | artista_nombre |
+------------+----------------+
|          1 | Aerosmith      |
|          3 | Timbaland      |
+------------+----------------+
2 rows in set (0.00 sec)
```

Todos los operadores de comparación estándar con los que ya estás familiarizado por PHP tienen soporte en SQL. El ejemplo anterior utiliza el operador de igualdad (=); el siguiente ejemplo muestra el funcionamiento del operador "mayor que o igual a" (>=), que regresa una lista con las canciones con rating 4 o superior:

```
mysql> SELECT cancion_titulo, ex_cancion_rating FROM canciones
    -> WHERE ex_cancion_rating >= 4;
+-------------------+-------------------+
| canción_título    | ex_canción_rating |
+-------------------+-------------------+
| Janie's Got A Gun |                 4 |
| Crazy             |                 5 |
| En Las Delicious  |                 5 |
```

(*continúa*)

```
| Pray              |                 4 |
| Apologize         |                 4 |
| SOS               |                 4 |
| Dancing Queen     |                 4 |
+-------------------+-------------------+
7 rows in set (0.00 sec)
```

Puedes combinar expresiones condicionales utilizando los operadores lógicos `AND`, `OR` y `NOT` (tal y como se hace en una declaración condicional regular PHP). He aquí un ejemplo, que presenta una lista con los artistas de Estados Unidos y el Reino Unido:

```
mysql> SELECT artista_nombre, artista_pais FROM artistas
    -> WHERE artista_pais = 'US'
    -> OR artista_pais = 'UK';
+----------------+--------------+
| artista_nombre | artista_país |
+----------------+--------------+
| Aerosmith      | US           |
| Timbaland      | US           |
| Take That      | UK           |
| Girls Aloud    | UK           |
+----------------+--------------+
4 rows in set (0.02 sec)
```

Ordenar y limitar colecciones de resultados

Si quieres ver los datos de una tabla ordenados por un campo específico, SQL cuenta con la cláusula `ORDER BY`. Te permite definir el nombre del campo sobre el que desees basar el orden del resultado y su dirección (ascendente o descendente).

Por ejemplo, para ver la lista de las canciones en orden alfabético, utiliza la siguiente declaración SQL:

```
mysql> SELECT cancion_titulo FROM canciones
    -> ORDER BY cancion_titulo;
+---------------------------+
| canción_título            |
+---------------------------+
| Another Crack In My Heart |
| Apologize                 |
| Babe                      |
| Crazy                     |
| Dancing Queen             |
| En Las Delicious          |
| Gimme Gimme Gimme         |
| Janie's Got A Gun         |
| Pray                      |
| SOS                       |
```

```
| Sure                        |
| Voulez Vous                 |
+----------------------------+
12 rows in set (0.04 sec)
```

Para invertir el orden de la lista, añade el modificador `DESC`, como se muestra a continuación:

```
mysql> SELECT cancion_titulo FROM canciones
    -> ORDER BY cancion_titulo DESC;
+----------------------------+
| canción_título             |
+----------------------------+
| Voulez Vous                |
| Sure                       |
| SOS                        |
| Pray                       |
| Janie's Got A Gun          |
| Gimme Gimme Gimme          |
| En Las Delicious           |
| Dancing Queen              |
| Crazy                      |
| Babe                       |
| Apologize                  |
| Another Crack In My Heart  |
+----------------------------+
12 rows in set (0.00 sec)
```

SQL también te permite limitar la cantidad de registros que aparecen en la colección de resultados con la palabra clave `LIMIT`, que acepta dos parámetros: la posición del registro para comenzar (a partir de 0) y la cantidad de registros que aparecerán. Por ejemplo, para mostrar las filas 4-9 (inclusivas) en una colección de resultados, utiliza la siguiente declaración:

```
mysql> SELECT cancion_titulo FROM canciones
    -> ORDER BY cancion_titulo
    -> LIMIT 3,6;
+--------------------+
| canción_título     |
+--------------------+
| Crazy              |
| Dancing Queen      |
| En Las Delicious   |
| Gimme Gimme Gimme  |
| Janie's Got A Gun  |
| Pray               |
+--------------------+
5 rows in set (0.00 sec)
```

(*continúa*)

Utilizar comodines

La declaración `SELECT` de SQL también da soporte a la cláusula `LIKE`, que puede utilizarse para realizar búsquedas dentro de campos de texto con el uso de comodines. Hay dos tipos de comodines aceptados en la cláusula `LIKE`: el signo de porcentaje (`%`), que es utilizado para representar cero o más apariciones de cierto carácter y el guión bajo (_), cuyo uso significa exactamente una aparición de cierto carácter.

El siguiente ejemplo muestra la cláusula `LIKE` en acción, que busca títulos de canciones que contengan el carácter 'g':

```
mysql> SELECT cancion_id, cancion_titulo FROM canciones
    -> WHERE cancion_titulo LIKE '%g%';

+------------+-------------------+
| canción_id | canción_título    |
+------------+-------------------+
|          1 | Janie's Got A Gun |
|          7 | Apologize         |
|          8 | Gimme Gimme Gimme |
|         10 | Dancing Queen     |
+------------+-------------------+
4 rows in set (0.00 sec)
```

Fusionar tablas

Hasta ahora, todas las consultas que has visto se concentran en una sola tabla. Pero SQL también te permite realizar búsquedas entre dos o más tablas al mismo tiempo y combinar el resultado en una sola colección de resultados. Esta tarea lleva el nombre técnico de *fusión*, pues "fusiona" diferentes tablas que comparten campos entre sí (las llaves externas) para crear nuevas vistas de los datos.

TIP

Cuando fusiones tablas, usa un prefijo para cada nombre de campo con el nombre de la tabla a la que pertenece, con el fin de evitar confusiones en caso de que varios campos en diferentes tablas tengan el mismo nombre.

He aquí un ejemplo de fusión entre las tablas *canciones* y *artistas* utilizando el campo común *artista_id* (la cláusula `WHERE` es utilizada para transformar los campos comunes entre sí):

```
mysql> SELECT cancion_id, cancion_titulo, artista_nombre FROM canciones,
artistas
    -> WHERE canciones.ex_cancion_artista = artistas.artista_id;
```

```
+------------+---------------------------+----------------+
| canción_id | canción_título            | artista_nombre |
+------------+---------------------------+----------------+
|          1 | Janie's Got A Gun         | Aerosmith      |
|          2 | Crazy                     | Aerosmith      |
|          8 | Gimme Gimme Gimme         | Abba           |
|          9 | SOS                       | Abba           |
|         10 | Dancing Queen             | Abba           |
|         11 | Voulez Vous               | Abba           |
|          7 | Apologize                 | Timbaland      |
|          4 | Sure                      | Take That      |
|          5 | Pray                      | Take That      |
|          6 | Another Crack In My Heart | Take That      |
|         12 | Babe                      | Take That      |
|          3 | En Las Delicious          | Cubanismo      |
+------------+---------------------------+----------------+
12 rows in set (0.00 sec)
```

Y a continuación tenemos un ejemplo que fusiona las tres tablas y luego filtra la colección de resultados aún más para incluir sólo las canciones con rating 4 o superior de artistas que no sean de Estados Unidos:

```
mysql> SELECT cancion_titulo, artista_nombre, rating_nombre
    -> FROM canciones, artistas, ratings
    -> WHERE canciones.ex_cancion_artista = artistas.artista_id
    -> AND canciones.ex_cancion_rating = ratings.rating_id
    -> AND ratings.rating_id >= 4
    -> AND artistas.artista_pais != 'US';
+------------------+----------------+---------------+
| canción_título   | artista_nombre | rating_nombre |
+------------------+----------------+---------------+
| En Las Delicious | Cubanismo      | Excelente     |
| Pray             | Take That      | Buena         |
| SOS              | Abba           | Buena         |
| Dancing Queen    | Abba           | Buena         |
+------------------+----------------+---------------+
4 rows in set (0.02 sec)
```

Modificar y eliminar registros

Así como insertas registros en una tabla (con el comando `INSERT`), también es posible borrarlos con la declaración `DELETE`. Por lo general, seleccionarás un subgrupo específico de filas para ser borradas añadiendo la cláusula `WHERE` a la declaración `DELETE`, como se muestra en el siguiente ejemplo, que elimina todas las canciones con un rating igual a o menor que 3:

```
mysql> DELETE FROM canciones
    -> WHERE ex_cancion_rating <= 3;
Query OK, 5 rows affected (0.02 sec)
```

(*continúa*)

También existe una declaración UPDATE, que puede utilizarse para cambiar el contenido de un registro; esta declaración también acepta la cláusula WHERE, de manera que puedes aplicar los cambios sólo en los registros que coinciden con cierto criterio. Examina el siguiente ejemplo, que cambia el rating 'Excelente' por 'Fantástico':

```
mysql> UPDATE ratings SET rating_nombre = 'Fantástico'
    -> WHERE rating_nombre = 'Excelente';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Puedes modificar varios campos separándolos con comas. He aquí un ejemplo que actualiza el registro de una canción en particular en la tabla *canciones*, modificando tanto el título como el rating:

```
mysql> UPDATE ratings SET cancion_titulo = 'Waterloo',
    -> ex_cancion_rating = 5
    -> WHERE cancion_id = 9;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Pregunta al experto

P: Estoy utilizando la base de datos ____ y no soporta el tipo de campo ____. ¿Qué hago ahora?

R: Diferentes bases de datos tienen distinta sintaxis para tipos de datos de los campos. Por ejemplo, MySQL llama a los campos enteros INT, mientras que SQLite llama a los mismos campos INTEGER. De cualquier manera, casi todas las bases de datos soportan (por lo menos) tipos de datos para valores enteros, de punto flotante, cadenas de caracteres, binarios y NULL. Lo único que necesitas es revisar la documentación de tu base de datos y encontrar la sintaxis para utilizar correctamente los tipos de datos en tus declaraciones SQL.

P: ¿Por qué necesito incluir todos los campos requeridos en la declaración SELECT? ¿No puedo utilizar el comodín * para recuperar todos los campos disponibles?

R: SQL sí soporta el asterisco (*) como carácter comodín para representar "todos los campos de la tabla", pero es preferible siempre designar explícitamente los campos que quieras ver en la colección de resultados. Esto permite que la aplicación sobreviva a los cambios estructurales en las tablas (o la tabla), además de que es más eficiente en el uso de memoria porque la colección de resultados no contendrá datos irrelevantes o no deseados.

Utilizar la extensión MySQLi de PHP

Como ya se explicó, PHP permite que los desarrolladores interactúen con la base de datos de dos maneras: utilizando extensiones personalizadas específicas de la base, o la extensión neutral (PDO). Mientras que estas últimas son más portátiles, muchos desarrolladores encuentran preferible utilizar las extensiones nativas de la base de datos con la que trabajan, sobre todo cuando ofrecen mejor rendimiento o más características que la versión PDO.

De los diferentes motores de base de datos soportados por PHP, el más popular es MySQL. No resulta difícil entender el porqué: tanto PHP como MySQL son proyectos de código libre, y al utilizarlos juntos, los desarrolladores obtienen beneficios de los grandes ahorros en costos de licencias en comparación con las opciones comerciales. Históricamente, además, PHP ha ofrecido soporte extraoficial para MySQL desde la versión 3, y tiene sentido aprovechar el tremendo esfuerzo que han realizado los desarrolladores de PHP y MySQL para asegurar que los dos paquetes trabajen juntos sin problemas.

Además de dar soporte a MySQL mediante las extensiones de objetos de datos de PHP (que se abordarán en la siguiente sección), PHP también incluye una extensión MySQL personalizada que lleva el nombre de MySQL Mejorado (MySQLi Improved). Esta extensión brinda beneficios de velocidad y de características superiores a la versión PDO y es una buena opción para desarrollar proyectos específicos con MySQL. Las siguientes secciones abordan esta extensión con gran detalle.

Recuperar datos

En la sección anterior creaste una base de datos y utilizaste una consulta `SELECT` para recuperar una colección de resultados de ella. Ahora harás lo mismo utilizando PHP, como en el siguiente script:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "user" "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar consulta
// itera sobre colección de resultados
// muestra cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $mysqli ->query($sql)) {
  if ($result->num_rows > 0) {
    while($row = $result->fetch_array()) {
      echo $row[0] . ":" . $row[1] . "\n";
    }
```

```
    $result->close();
  } else {
    echo "No se encontró ningún registro que coincida con su búsqueda.";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

La figura 7-3 muestra cómo se debe ver el resultado del script.

Como puedes ver, utilizar PHP para obtener datos de una base requiere varios pasos, que se describen a continuación:

1. Con el fin de establecer comunicación con el servidor que contiene la base de datos MySQL, primero necesitas abrir una conexión con el mismo. Toda la comunicación entre PHP y el servidor de base de datos se realiza a través de esta conexión.

   Para inicializar esta conexión, inicializa un objeto de la clase MySQLi y pasa cuatro argumentos al constructor del objeto: el nombre del servidor anfitrión de MySQL al que intentas conectarte, un nombre y una contraeña válidos para obtener el acceso necesario, y el nombre de la base de datos que quieres utilizar.

   Dando por hecho que la conexión se estableció con éxito, este objeto representa la conexión con la base de datos para todas las futuras operaciones y expone los métodos para realizar consultas, búsquedas y para procesar las colecciones de resultados. Si el intento de conexión fracasó, el objeto será falso; entonces es posible obtener un mensaje de error que explique las razones de la falla; esto se lleva a cabo invocando la función `mysqli_connect_error()`.

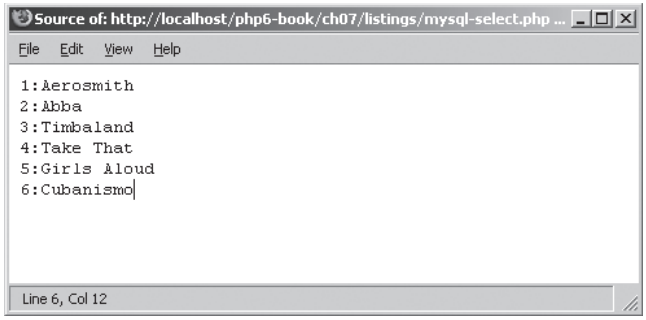


Figura 7-3 Registros recuperados de una base de datos MySQL con PHP

TIP

Si los dos servidores, el de base de datos y el Web, se ejecutan en la misma computadora, puedes utilizar `localhost` como nombre de servidor.

2. El siguiente paso consiste en crear y ejecutar la consulta SQL. Esto se realiza invocando el método `query()` para el objeto de MySQLi y transmitiéndole la consulta que se va a ejecutar. Si la consulta no tuvo éxito, el método regresa un valor booleano falso, y un mensaje de error que explica la causa de la falla es almacenado en la propiedad `'error'` del objeto MySQLi.
3. Si, por otra parte, la consulta se ejecuta con éxito y regresa uno o más registros, el valor de retorno del método `query()` se convierte en otro objeto, este último una instancia de la clase MySQLi_Result. Este objeto representa la colección de resultados regresada por la consulta, y expone varios métodos para procesar los registros individuales en la colección de resultados.

   Uno de esos métodos es `fetch_array()`. Cada vez que se invoca, regresa el siguiente registro de la colección de resultados como una matriz. Esto hace que el método `fetch_array()` se acople bien con el uso de los bucles `while` y `for`. El contador del bucle determina cuántas veces debe ejecutarse; este resultado se obtiene de la propiedad `'num_rows'` del objeto MySQLi_Result, que almacena el número de filas regresadas por la consulta. Los campos individuales del registro pueden accesarse como elementos de una matriz, utilizando ya sea el índice del campo o su nombre.

PRECAUCIÓN

La propiedad `'num_rows'` sólo tiene significado cuando se utiliza con consultas que regresan datos, como `SELECT`; no debe utilizarse con los consultas `INSERT`, `UPDATE` o `DELETE`.

4. Cada colección de resultados regresada después de una consulta ocupa cierta cantidad de memoria. Así, una vez que el resultado ha sido procesado, es buena idea destruir el objeto MySQLi_Result, para liberar la memoria que utiliza; esto se realiza con el método `close()`. Y una vez que has terminado de trabajar con la base de datos, también es buena idea destruir el objeto principal MySQLi de manera similar, invocando el método `close()`.

Regresar registros como matrices y objetos

El ejemplo anterior mostró un método para procesar la colección de resultados: el método `fetch_array()` del objeto MySQLi_Result. Este método regresa cada registro de la colección de resultados como una matriz que contiene llaves indexadas tanto numéricamente como por cadenas de caracteres; esto ofrece a los desarrolladores la conveniencia de hacer referencia a campos individuales de cada registro ya sea por índice o por nombre de campo.

El ejemplo anterior mostró cómo recuperar campos individuales utilizando el número de índice. El siguiente ejemplo, equivalente al anterior, realiza la misma tarea utilizando nombres de campo:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar consulta
// reitera sobre colección de resultados
// muestra cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $mysqli ->query($sql)) {
  if ($result->num_rows > 0) {
    while($row = $result->fetch_array()) {
      echo $row['artista_id'] . ":" . $row['artista_nombre'] . "\n";
    }
    $result->close();
  } else {
    echo "No se encontró ningún registro que coincida con su búsqueda.";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Sin embargo, existe un tercer método para recuperar registros: como objetos, utilizando el método `fetch_object()`. Aquí, cada registro se representa como un objeto y los campos de un registro se representan como propiedades del objeto. Después, los campos individuales pueden ser accesados utilizando la notación estándar `$objeto->propiedad`. El siguiente ejemplo ilustra este procedimiento:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar consulta
// reitera sobre colección de resultados
```

```
// muestra cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $mysqli->query($sql)) {
  if ($result->num_rows > 0) {
    while($row = $result->fetch_object()) {
      echo $row->artista_id . ":" . $row->artista_nombre . "\n";
    }
    $result->close();
  } else {
    echo "No se encontró ningún registro que coincida con su búsqueda.";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Añadir y modificar datos

Es igual de sencillo ejecutar un consulta que cambie los datos en la base, ya sea insertar (`INSERT`), actualizar (`UPDATE`) o borrar (`DELETE`). El siguiente ejemplo lo muestra, añadiendo un nuevo registro a la tabla *artistas*:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar consulta
// añade un nuevo registro
// datos de salida: " Nuevo artista con id: 7 ha sido añadido. "
$sql = "INSERT INTO artistas (artista_nombre, artista_pais) VALUES
('Kylie Minogue', 'AU')";
if ($mysqli->query($sql)=== true) {
  echo 'Nuevo artista con id: ' . $mysqli->insert_id . 'ha sido
  añadido.';
} else {
  echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Como verás, esto no es muy diferente de la necesidad de aplicar una consulta `SELECT` al programar. De hecho, es un poco más sencillo porque no hay una colección de datos para procesar; todo lo que se necesita es probar el valor regresado del método `query()` y verificar si la consulta se ejecutó correctamente o no. Advierte también el uso de la nueva propiedad '`insert_id`', que regresa el ID generado por la última consulta `INSERT` (sólo es útil si la tabla en la cual se aplicó `INSERT` contiene un campo que se incrementa automáticamente).

¿Qué hay de incrementar un registro existente? Lo único que necesitas es cambiar la línea de comando SQL:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar consulta
// añade un nuevo registro
// datos de salida: "1 fila(s) actualizadas."
$sql = "UPDATE artistas SET artista_nombre = 'Eminem', artista_pais =
'US' WHERE artista_id = 7;
if ($mysqli->query($sql)=== true) {
  echo $mysqli->affected_rows . ' fila(s) actualizadas.';
} else {
  echo "ERROR: No fue posible ejecutar: $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Cuando ejecutas `UPDATE` o `DELETE`, el número de filas afectadas por la declaración se almacenarán en la propiedad '`affected_rows`' del objeto MySQLi. El ejemplo siguiente muestra el uso de esta propiedad.

Utilizar declaraciones preparadas

En caso de que necesites ejecutar un consulta particular varias veces con diferentes valores (por ejemplo, una serie de declaraciones `INSERT`), el servidor MySQL da soporte a *declaraciones preparadas*, que ofrecen medios más eficientes para completar esta tarea en lugar de invocar repetidamente el método `$mysqli->query()`.

En esencia, una declaración preparada es una consulta SQL modelo que contiene variables prealmacenadas para los valores que serán insertados o modificados. Esta declaración es almacenada en el servidor de base de datos y la invoca todas las veces que sea necesario; las variables almacenadas previamente se reemplazan con los nuevos valores cada vez que se ejecutan. Dado que la declaración está almacenada en el servidor de base de datos, una decla-

ración preparada suele ser más rápida para realizar operaciones por lote que implican ejecutar la misma consulta SQL una y otra vez con diferentes valores.

Pregunta al experto

P: ¿Por qué las declaraciones preparadas ofrecen mejor rendimiento?

R: En condiciones normales, cada vez que se ejecuta una declaración SQL sobre el servidor de base de datos, éste debe segmentar el código SQL y verificar su sintaxis y estructura antes de permitir su ejecución. Con una declaración preparada, la declaración SQL está almacenada temporalmente en el servidor de base de datos y, por lo mismo, sólo necesita segmentarse y validarse una sola vez. Más aún, cada vez que se ejecuta la declaración, sólo los valores que cambian necesitan transmitirse al servidor, en lugar de enviar la declaración completa.

Para ver una declaración preparada en acción, examina el siguiente script, que inserta varias canciones en la base de datos utilizando precisamente una declaración preparada con la extensión MySQLi de PHP:

```
<?php
// define valores a ser insertados
$canciones = array(
  array('Patience', 4, 3),
  array('Beautiful World', 4, 4),
  array('Shine', 4, 4),
  array('Hold On', 4, 3),
);

// intenta establecer la conexión con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// prepara el consulta modelo
// lo ejecuta varias veces
$sql = "INSERT INTO canciones (cancion_titulo, ex_cancion_artista, ex_
cancion_rating) VALUES (?, ?, ?)";
if ($stmt = $mysqli->prepare($sql)) {
  foreach ($canciones as $s) {
    $stmt->bind_param('sii', $s[0], $s[1], $s[2]);
    if ($stmt->execute()) {
     echo "Nueva canción con id: " . mysqli->insert_id . "ha sido añadida.\n";
    } else {
     echo "ERROR: No fue posible ejecutar consulta: $sql. " . $mysqli->error;
     }
  }
```

```
} else {
  echo "ERROR: No fue posible preparar consulta: $sql. " . $mysqli
->error;
}

// cierra conexión
$mysqli->close();
?>
```

Como se aprecia en este ejemplo, el uso de una declaración preparada implica un proceso diferente al que has visto en ejemplos previos.

Para utilizar una declaración preparada, primero es necesario definir la cadena de la consulta que será ejecutada varias veces. Esta cadena de consulta por lo regular contendrá uno o varios contenedores, representados por los signos de interrogación (`?`). Estos contenedores serán sustituidos por los nuevos valores cada vez que se ejecute la declaración. Entonces se transmite la cadena de consulta con sus contenedores al método `prepare()` del objeto MySQLi, que verifica el comando SQL en busca de errores y regresa un objeto MySQLi_Stmt que representa la declaración preparada.

Ahora, ejecutar la declaración preparada es cuestión de realizar dos acciones, por lo general con un bucle:

1. *Unir los valores a la declaración preparada*. Los valores que van a ser interpolados en la declaración necesitan *unirse* a sus contenedores con el método `bind_param()` del objeto MySQLi_Stmt. El primer argumento para este método debe ser una cadena de secuencia ordenada que indique los tipos de datos de los valores que serán interpolados (*s* para una cadena, *i* para un entero, *d* para un número de doble precisión); este argumento es seguido por los nuevos valores. En el ejemplo anterior, la cadena '`sii`' indica que los valores que se interpolarán en la declaración preparada serán, en secuencia, un tipo cadena de caracteres (el título de la canción), un tipo entero (la llave externa del artista) y otro tipo entero (la llave externa para el rating).
2. *Ejecutar la declaración preparada*. Una vez que se han unido los valores a sus contenedores, se ejecuta la declaración preparada invocando el método `execute()` del objeto MySQLi_Stmt. Este método reemplaza los contenedores en la declaración preparada con los nuevos valores y se ejecuta en el servidor.

De esta manera, el uso de una declaración preparada ofrece beneficios de rendimiento cuando necesitas ejecutar la misma declaración varias veces, y sólo los valores cambian en cada ejecución del comando. Las bases de datos más populares, incluyendo MySQL, PostgreSQL, Oracle e InterBase, soportan declaraciones preparadas, y esta característica debe ser utilizada cuando esté disponible para realizar tus operaciones SQL por lote de manera más eficiente.

Manejo de errores

Si tu código de base de datos no funciona como esperabas, no te preocupes; la extensión MySQLi contiene una gran cantidad de funciones que te pueden decir la causa. Ya las has visto en acción en diferentes lugares de los ejemplos anteriores, pero a continuación aparece una lista completa:

- La propiedad `'error'` del objeto MySQLi guarda el último mensaje de error generado por el servidor de base de datos.
- La propiedad `'errno'` del objeto MySQLi guarda el último error de código regresado por el servidor de base de datos.
- La función `mysqli_connect_error()` regresa el mensaje de error generado por el último intento (fallido) de conexión.
- La función `mysqli_connect_errno()` regresa el error de código generado por el último intento (fallido) de conexión.

TIP

Para hacer tu aplicación más robusta contra errores, y para localizarlos y resolverlos con mayor facilidad, es una buena idea utilizar estas funciones para manejo de errores con toda libertad dentro de tu código.

Prueba esto 7-2 Añadir empleados a una base de datos

Ahora que ya conoces el uso básico de las extensiones MySQLi de PHP, practicarás lo que aprendiste en la sección anterior en una aplicación "real". La aplicación es un formulario Web que permite al usuario insertar nombres de empleados y sus puestos en una base de datos para empleados basada en MySQL. PHP valida y limpia los valores ingresados en el formulario; después se transforman en registros de la base con el uso de la extensión MySQLi de PHP.

Para empezar a construir esta aplicación, primero ejecuta el programa cliente de línea de comando MySQL y crea una base de datos vacía.

```
mysql> CREATE DATABASE empleados;
Query OK, 1 row affected (0.20 sec)
```

Después crea una tabla que contenga los registros de los empleados:

```
mysql> USE empleados;
Database changed
mysql> CREATE TABLE empleados (
```

(*continúa*)

```
    -> id INT (4) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> nombre VARCHAR(255) NOT NULL,
    -> puesto VARCHAR(255) NOT NULL
    -> );
Query OK, 0 row affected (0.20 sec)
```

¿Todo listo? Lo siguiente es construir un formulario Web que acepte los datos del empleado (en este caso, su nombre y puesto). Al momento de enviarlo, el procesador del formulario verificará los datos de entrada y, de ser válidos, generará una consulta `INSERT` para insertarlos en la tabla creada en el paso anterior.

He aquí el script (*empleados.php*):

```
<!DOCTYPE html PUBLIC "-// W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-2: Añadir empleados a una base de datos</title>
    <style type="text/css">
    div#message{
      text-align:center;
      margin-left:auto;
      margin-right:auto;
      width:40%;
      border: solid 2px green
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 7-2: Añadir empleados a una base de datos</h2>
    <h3>Añade un Nuevo Empleado</h3>
<?php
    // si el formulario ha sido enviado
    // procesa los datos del formulario
    if (isset($_POST['submit'])) {
      // intenta conectarse con la base de datos MySQL
      $mysqli = new mysqli("localhost", "usuario" "contra", "empleados");
      if ($mysqli === false) {
        die("ERROR: No fue posible conectarse con la base de datos. " .
mysqli_connect_error());
      }

      // abre bloque de mensaje
      echo '<div id="message">';

      // recupera y verifica los valores de entrada
      $inputError = false;
      if (empty($_POST['emp_nombre'])) {
        echo 'ERROR: Por favor ingrese un nombre de empleado válido';
```

```
        $inputError = true;
      } else {
        $nombre = $mysqli->escape_string($_POST['emp_nombre']);
      }

      if ($inputError != true && empty($_POST['emp_puesto'])) {
        echo 'ERROR: Por favor ingrese un puesto válido';
        $inputError = true;
      } else {
        $puesto = $mysqli->escape_string($_POST['emp_puesto']);
      }

      // añade valores a la base de datos utilizando el consulta INSERT
      if ($inputError != true) {
      $sql = "INSERT INTO empleados (nombre, puesto)
                VALUES ('$nombre', '$puesto')";
        if ($mysqli->query($sql) === true) {
          echo 'Nuevo registro de empleado añadido con ID: ' . $mysqli
          ->insert_id;
        } else {
          echo "ERROR: No fue posible ejecutar el consulta: $sql. " .
          $mysqli->error;
        }
      }

      // cierra bloque de mensaje
      echo '</div>';

      // cierra conexión
      $mysqli->close();
    }
?>
    </div>

    <form action="empleados.php" method="POST">
      Nombre del empleado: <br />
      <input type="text" name="emp_nombre" size="40" />
      <p />
      Puesto del empleado: <br />
      <input type="text" name="emp_puesto" size="40" />
      <p />
      <input type="sumbit" name="submit" value="Enviar" />
    </form>

  </body>
</html>
```

Cuando se invoca este script, genera un formulario Web con campos para el nombre y puesto de los empleados. La figura 7-4 muestra cómo luce el formulario inicial.

(*continúa*)

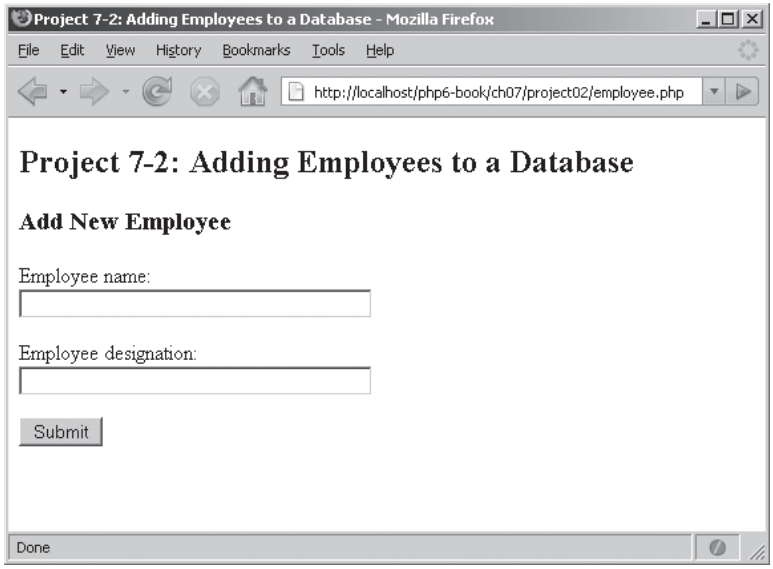


Figura 7-4 Formulario Web para insertar nuevos empleados

Cuando se envía el formulario, el script procede a inicializar un nuevo objeto MySQLi y abre la conexión para el servidor de bases de datos MySQL. Luego verifica los campos del formulario para asegurarse de que contienen valores de entrada válidos. De ser así, cada uno de estos valores es "limpiado" utilizando el método `escape_string()` del objeto MySQLi, que automáticamente limpia los caracteres especiales en los datos de entrada como preludio para insertarlos en la base de datos y luego interpola en la consulta INSERT, que se ejecuta con el método `query()` del objeto para guardar los valores en la base de datos. Los errores, si existen, se muestran utilizando la propiedad error del objeto MySQLi.

La figura 7-5 muestra los datos de salida cuando se agrega un registro correctamente y la figura 7-6 muestra los datos de salida cuando falla la validación en un campo de entrada.

Eso fue muy fácil. Ahora, ¿qué tal si rehacemos el script para que, además de permitir la inserción de nuevos empleados a la base de datos, también muestre los registros existentes en la misma? No es muy difícil; se añade otra invocación al método `query()`, esta vez con una consulta `SELECT`, y se procesa la colección de resultados utilizando un bucle.

He aquí el código modificado:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-2: Añadir empleados a una base de datos</title>
    <style type="text/css">
    div#message {
```

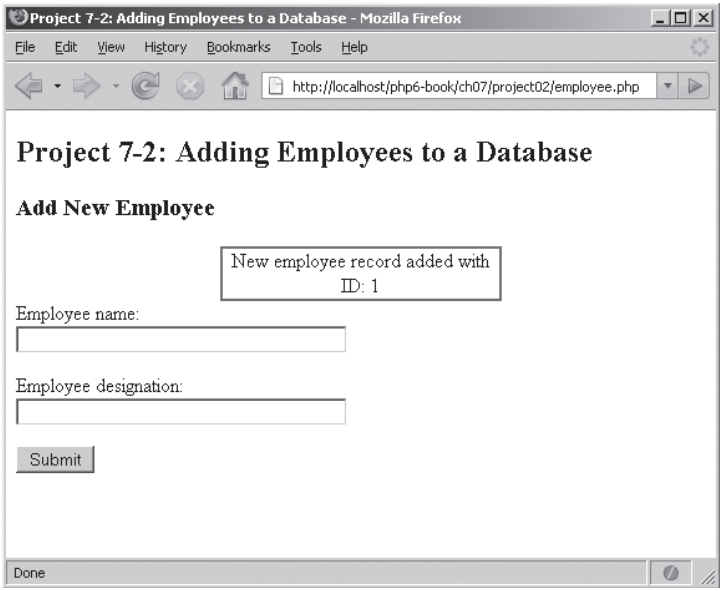


Figura 7-5 El resultado de una inserción correcta de un nuevo empleado en la base de datos

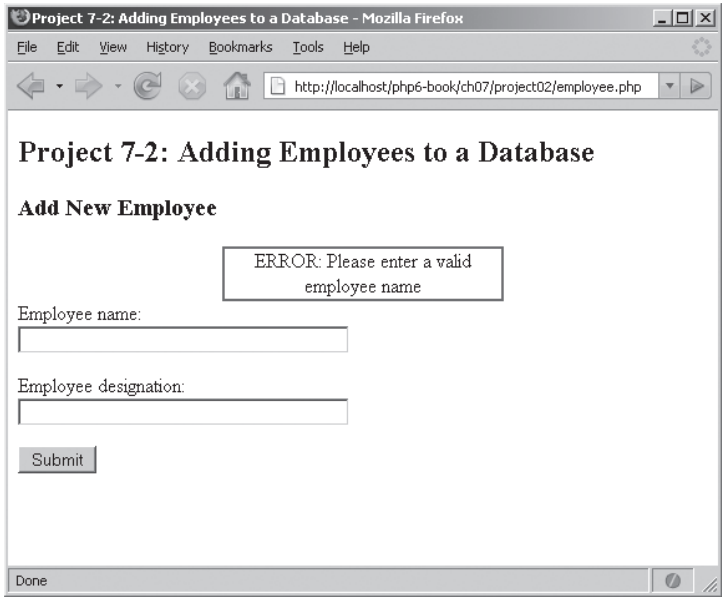


Figura 7-6 Los datos de salida después de enviar datos no válidos en el formulario Web

(*continúa*)

```
      text-align:center;
      margin-left:auto;
      margin-right:auto;
      width:40%;
      border: solid 2px green
    }
    table {
      border-collapse: collapse;
      width: 320px;
    }
    tr.heading {
      font-weight: bolder;
    }
    td {
      border: 1px solid black;
      padding: 0 0.5em;
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 7-2: Añadir empleados a una base de datos</h2>
    <h3>Añade un Nuevo Empleado</h3>
<?php
    // intenta conectarse con la base de datos MySQL
    $mysqli = new mysqli("localhost", "usuario", "contra", "empleados");
    if ($mysqli === false) {
      die("ERROR: No fue posible conectarse con la base de datos. " .
mysqli_connect_error());
    }

    // si el formulario ha sido enviado
    // procesa los datos del formulario
    if (isset($_POST['submit'])) {
      // abre bloque de mensaje
      echo '<div id="message">';

      // recupera y verifica los valores de entrada
      $inputError = false;
      if (empty($_POST['emp_nombre'])) {
        echo 'ERROR: Por favor ingrese un nombre de empleado válido';
        $inputError = true;
      } else {
        $nombre = $mysqli->escape_string($_POST['emp_nombre']);
      }

      if ($inputError != true && empty($_POST['emp_puesto'])) {
        echo 'ERROR: Por favor ingrese un puesto válido';
        $inputError = true;
      } else {
```

```
        $puesto = $mysqli->escape_string($_POST['emp_puesto']);
      }

      // añade valores a la base de datos utilizando el consulta INSERT
      if ($inputError != true) {
      $sql = "INSERT INTO empleados (nombre, puesto)
              VALUES ('$nombre', '$puesto')";
        if ($mysqli->query($sql) === true) {
          echo 'Nuevo registro de empleado añadido con ID: ' . $mysqli
          ->insert_id;
        } else {
          echo "ERROR: No fue posible ejecutar el consulta: $sql. " .
$mysqli->error;
        }
      }

      // cierra bloque de mensaje
      echo '</div>';
    }
?>
    </div>

    <form action="empleados.php" method="POST">
      Nombre del empleado: <br />
      <input type="text" name="emp_nombre" size="40" />
      <p />
      Puesto del empleado: <br />
      <input type="text" name="emp_puesto" size="40" />
      <p />
      <input type="submit" name="submit" value="Enviar" />
    </form>

    <h3>Lista de empleados</h3>
<?php
    // obtiene registros
    // da formato como una tabla HTML
    $sql = "SELECT id, nombre, puesto FROM empleados";
    if($result = $mysqli->query($sql)) {
      if ($result->num_rows > 0) {
        echo "<table>\n";
        echo "  <tr class=\"heading\">\n";
        echo "    <td>ID</td>\n";
        echo "    <td>Nombre</td>\n";
        echo "    <td>Puesto</td>\n";
        echo "    </tr>\n";
        while ($row = $result->fetch_object()) {
          echo "  <tr>\n";
          echo "    <td>" . $row->id . "</td>\n";
```

(*continúa*)

```
          echo "    <td>" . $row->nombre . "</td>\n";
          echo "    <td>" . $row->puesto . "</td>\n";
          echo "  </tr>\n";
        }
        echo "</table>";
        $result->close();
    } else {
        echo "No hay empleados en la base de datos.";
      }
    } else {
      echo "ERROR: No fue posible ejecutar el consulta: $sql. " .
$mysqli->error;
    }

    // cierra conexión
    $mysqli->close();
?>
  </body>
</html>
```

Esta versión del script ahora incluye la consulta `SELECT`, que recupera los registros de la tabla de empleados como un objeto y los procesa utilizando un bucle. Estos registros son formados como una tabla HTML y presentados en la parte inferior de la página Web. Un efecto secundario útil de estas modificaciones es que los nuevos registros de empleados guardados mediante el formulario se reflejarán de inmediato en la tabla HTML una vez que se envíe el formulario.

La figura 7-7 muestra los datos de salida de esta versión modificada del script.

Utilizar la extensión SQLite de PHP

En este punto ya sabes conectar un script PHP a una base de datos MySQL para recuperar, añadir, editar y borrar registros. Sin embargo, MySQL no es el único juego en la ciudad; PHP 5.x también incluye soporte integrado para SQLite, opción eficiente y más ligera que MySQL. Esta sección aborda la base de datos SQLite y la extensión SQLite de PHP con gran detalle.

Introducción a SQLite

SQLite es una base de datos rápida y eficiente que ofrece una opción viable a MySQL, sobre todo para aplicaciones pequeñas y de tamaño mediano. Sin embargo, a diferencia de MySQL, que contiene una gran cantidad de componentes interrelacionados, SQLite está integrada en una sola biblioteca. También es significativamente más pequeña que MySQL, su línea de comandos pesa menos de 200 KB, y soporta todos los comandos SQL estándar con los que estás familiarizado. MySQL y SQLite también difieren en sus políticas de licencia: a diferencia de

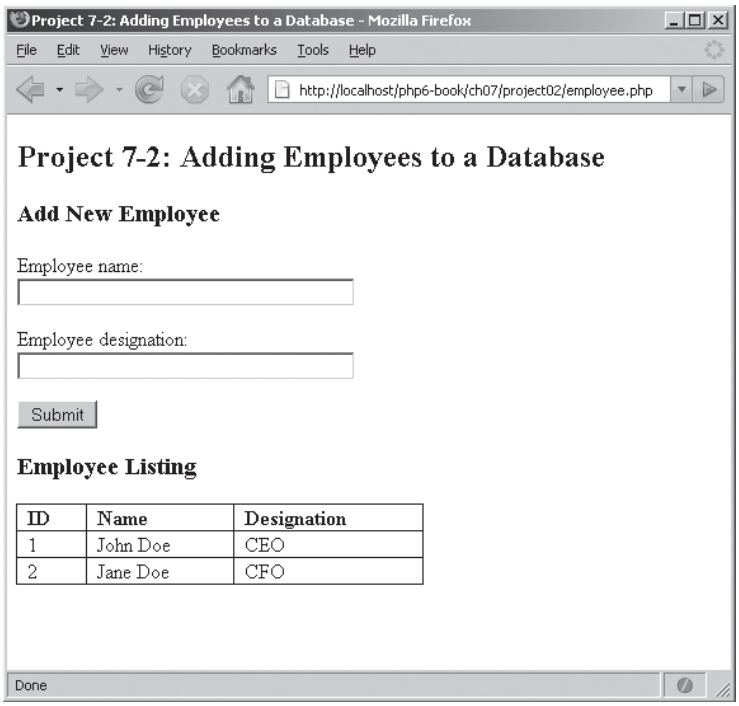


Figura 7-7 Página Web que muestra la lista de los empleados

MySQL, el código fuente de SQLite es completamente de dominio público, lo cual significa que los desarrolladores pueden utilizarlo y distribuirlo como les plazca, para productos tanto comerciales como no comerciales.

Sin embargo, el tamaño de SQLite disfraza su poder. La base de datos tiene una capacidad de soportar dos terabytes y puede tener un mejor rendimiento que MySQL en ciertas situaciones. En parte, esto se debe a razones estructurales: SQLite lee y escribe registros directamente del disco y por lo mismo ocupa menos recursos que MySQL, que opera sobre una arquitectura cliente-servidor que puede ser afectada por variables relacionadas con el nivel de la red.

NOTA

En el siguiente ejercicio, las palabras en negritas indican comandos que debes escribir en la línea de comando de SQLite. Los comandos pueden escribirse en mayúsculas o minúsculas. Antes de comenzar con el ejercicio asegúrate de haber instalado, configurado y probado la base de datos SQLite de acuerdo con las instrucciones que aparecen en el apéndice A de este libro.

SQLite da soporte a todas las declaraciones estándar de SQL que ya conoces y has llegado a amar en las dos secciones anteriores: `SELECT`, `INSERT`, `DELETE`, `UPDATE` y

`CREATE TABLE`. El siguiente ejemplo muestra el uso de SQLite replicando las tablas de la base de datos MySQL utilizadas en la sección anterior. Comienza iniciando el programa cliente de línea de comandos de SQLite y crea una nueva base de datos en el directorio en uso que lleve el nombre *musica.db*:

```
shell> sqlite musica.db
```

Si todo salió bien, verás un mensaje de bienvenida y un indicador SQL interactivo como el siguiente:

```
sqlite>
```

Ahora puedes comenzar a dictar declaraciones SQL desde este indicador. Comienza por crear una tabla que contenga la información de los artistas:

```
sqlite> CREATE TABLE artistas (
   ...> artista_id INTEGER NOT NULL PRIMARY KEY,
   ...> artista_nombre TEXT NOT NULL,
   ...> artista_pais TEXT NOT NULL
   ...>);
```

Contrariamente a MySQL, que ofrece un amplio rango de diferentes tipos de datos para sus campos, SQLite soporta sólo cuatro tipos, los cuales se muestran en la tabla 7-3.

Tipo de campo	Descripción
`INTEGER`	Campo numérico para firmar valores enteros
`REAL`	Campo numérico para valores de punto flotante
`TEXT`	Tipo cadena de caracteres
`BLOB`	Tipo binario

Tabla 7-3 Tipos de datos de SQLite

Lo que es más importante, SQLite es una base de datos "sin tipos". Esto significa que los campos de esta base de datos NO necesitan estar asociados a un tipo específico, e incluso cuando lo están, pueden almacenar valores de un tipo diferente al especificado. La única excepción a esta regla son los campos tipo `INTEGER PRIMARY KEY`: que son campos "autonuméricos" que generan identificadores numéricos únicos para cada registro de la tabla, de manera similar a los campos `AUTO_INCREMENT` de MySQL.

Con estos hechos en mente, continúa con el ejercicio y crea las restantes dos tablas utilizando estas declaraciones SQL:

```
sqlite> CREATE TABLE ratings (
   ...> rating_id INTEGER NOT NULL PRIMARY KEY,
```

```
   ...> rating_nombre TEXT NOT NULL
   ...> );

sqlite> CREATE TABLE canciones (
   ...> cancion_id INTEGER NOT NULL PRIMARY KEY,
   ...> cancion_titulo TEXT NOT NULL,
   ...> ex_cancion_artista INTEGER NOT NULL,
   ...> ex_cancion_rating INTEGER NOT NULL
   ...> );
```

Ahora alimenta las tablas con registros:

```
sqlite> INSERT INTO artistas (artista_id, artista_nombre, artista_pais)
   ...> VALUES ('1', 'Aerosmith', 'US');
sqlite> INSERT INTO artistas (artista_nombre, artista_pais)
   ...> VALUES ('Abba', 'SE');

sqlite> INSERT INTO ratings (rating_id, rating_nombre)
   ...> VALUES (4, 'Bueno');
sqlite> INSERT INTO ratings (rating_id, rating_nombre)
   ...> VALUES (5, 'Excelente');

sqlite> INSERT INTO canciones (cancion_titulo, ex_cancion_artista,
        ex_cancion_rating)
   ...> VALUES ('Janie''s Got a Gun', 1, 4);
sqlite> INSERT INTO canciones (cancion_titulo, ex_cancion_artista,
        ex_cancion_rating)
   ...> VALUES ('Crazy', 1, 5);
```

NOTA

El archivo de código para este libro tiene una lista completa con las declaraciones SQL `INSERT` necesarias para alimentar las tres tablas utilizadas en este ejercicio. Ejecuta esas declaraciones y termina de construir las tablas antes de proseguir.

Una vez que hayas terminado, prueba la declaración `SELECT`:

```
sqlite> SELECT artista_nombre, artista_pais FROM artistas
   ...> WHERE artista_pais = 'US'
   ...> OR artista_pais = 'UK';
Aerosmith|US
Timbaland|US
Take That|UK
Girls Aloud|UK
```

SQLite soporta por completo las fusiones SQL; he aquí un ejemplo:

```
sqlite> SELECT cancion_titulo, artista_nombre, rating_nombre
   ...> FROM canciones, artistas, ratings
   ...> WHERE canciones.ex_cancion_artista = artista.artista_id
   ...> AND canciones.ex_canciones_rating = ratings.rating_id
```

```
   ...> AND ratings.rating_id >= 4
   ...> AND artistas.artista_pais != 'US';
En Las Delicious|Cubanismo|Fantastic
Pray|Take That|Good
SOS|Abba|Fantastic
Dancing Queen|Abba|Good
```

Recuperar datos

Recuperar datos de una base SQLite con PHP no es muy diferente de recuperarlos de una base MySQL. El siguiente script PHP muestra el proceso utilizando la base de datos *musica.db* creada en la sección anterior:

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");

// intenta ejecutar el consulta
// reitera sobre una colección de resultados
// presenta cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $sqlite->query($sql)) {
  if($result->numRows() > 0) {
    while($row = $result->fetch()) {
      echo $row[0] . ":" . $row[1] . "\n";
    }
  } else {
    echo "No se encontraron registros que coincidieran con los criterios
del consulta.";
  }
} else{
  echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

El script realiza las siguientes acciones:

1. Abre el archivo de base de datos, para realizar las operaciones, al inicializar una instancia de la clase SQLiteDatabase y transmitiendo al constructor del objeto la ruta de acceso completa a la base de datos. Si este archivo de base de datos no es localizado, se creará uno vacío con el nombre proporcionado (siempre y cuando el script tenga privilegios de escritura sobre el directorio correspondiente).

2. El siguiente paso consiste en crear y ejecutar la consulta SQL. Esto se realiza invocando el método `query()` del objeto SQLiteDatabase y transmitiéndole el consulta que se va a ejecutar. Dependiendo de si el consulta fue exitoso o fracasó, la función regresa un valor verdadero o falso; en caso de fallo, el código de error correspondiente a la razón de la falla puede obtenerse invocando el método `lastError()` del objeto SQLiteDatabase. La función `sqlite_error_string()` convierte este código de error en un mensaje comprensible para los humanos.

TIP

Hay una semejanza cercana entre estos pasos y los que seguiste para utilizar la extensión MySQL en la sección anterior.

3. Por otra parte, si la consulta se ejecutó correctamente y regresa uno o más registros, el valor de retorno del método `query()` es otro objeto, este último será una instancia de la clase SQLiteResult. Este objeto representa la colección de resultados regresada por el consulta, y expone la consulta para procesar registros individuales en la colección de resultados.

   Este script utiliza el método `fetch()`, que regresa el siguiente resultado de la colección de resultados como una matriz cada vez que se invoca. Utilizado en un bucle, este método proporciona una manera conveniente de hacer reiteraciones sobre la colección de resultados, un registro a la vez. Los campos individuales del registro pueden ser accesados como elementos de la matriz, utilizando ya sea el índice o el nombre del campo. La cantidad de resultados en la colección de éstos puede recuperarse a través del método `numRows()` del objeto SQLiteResult.

4. Una vez que la colección de resultados entera ha sido procesada y no restan operaciones por ejecutarse en el archivo de base de datos, es buena idea cerrar el manejador de la base de datos para liberar la memoria que ocupa, lo cual se realiza destruyendo el objeto SQLiteDatabase.

Recuperar registros como matrices y objetos

El método `fetch()` del objeto SQLiteResult acepta un modificador adicional, que controla la manera en que se recuperan los elementos de la colección de resultados. Este modificador puede ser cualquiera de los valores `SQLITE_NUM` (para recuperar cada registro como una matriz numérica indexada), `SQLITE_ASSOC` (para regresar cada registro como una matriz asociativa) o `SQLITE_BOTH` (para regresar cada registro de ambas maneras, como una matriz numérica indexada y como una matriz asociativa, además de la opción por defecto). He aquí un ejemplo, que muestra estos modificadores en acción y produce una serie de datos de salida equivalente al ejemplo anterior:

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");
// intenta ejecutar el consulta
// presenta registros utilizando diferentes estilos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if($result = $sqlite->query($sql)) {

  // recupera registros como una matriz numérica
  $row = $result->fetch(SQLITE_NUM);
  echo $row[0] . ":" . $row[1] . "\n";

  // recupera registros como una matriz asociativa
  $row = $result->fetch(SQLITE_ASSOC);
  echo $row['artista_id'] . ":" . $row['artista_nombre'] . "\n";

  // recupera registros como un objeto
  $row = $result->fetchObject();
  echo $row->artista_id . ":" . $row->artista_nombre . "\n";

} else {
  echo "ERROR: no fue posible ejecutar $sql. " . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

También es posible regresar cada registro como un objeto, reemplazando `fetch()` con el método `fetchObject()`. He aquí un ejemplo equivalente al anterior, sólo que en lugar de recuperar valores de campo como elementos de una matriz, lo hace como propiedades de un objeto:

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");
// intenta ejecutar el consulta
// reitera sobre una colección de resultados
// presenta cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if($result = $sqlite->query($sql)) {
  if ($result->numRows() > 0) {
    while ($row = $result->fetchObject()) {
      echo $row->artista_id . ":" . $row->artista_nombre . "\n";
```

```
      }
  } else {
    echo "No se encontraron registros que coincidieran con los criterios
del consulta.";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

Una característica importante de la extensión SQLite es su capacidad de recuperar *todos* los registros de una colección de resultados al mismo tiempo como un conjunto de matrices anidadas, a través del método `fetchAll()` del objeto SQLiteResult. Un bucle `foreach` puede reiterar sobre esta colección anidada, recuperando un registro tras otro. El siguiente ejemplo muestra este procedimiento en acción:

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");

// intenta ejecutar el consulta
// reitera sobre una colección de resultados
// presenta cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $sqlite->query($sql)) {
  $data = $result->fetchAll();
  if (count($data) > 0) {
    foreach ($data as $row) {
      echo $row[0] . ":" . $row[1] . "\n";
    }
  } else {
    echo "No se encontraron registros que coincidieran con los criterios
del consulta.";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

PRECAUCIÓN

El método `fetchAll()` regresa la colección de resultados completa como un conjunto de matrices anidadas, y todos ellos se almacenan en la memoria de la computadora hasta que termina el proceso. Para no agotar la memoria, no utilices este método si es posible que tu consulta regrese un gran número de registros.

Añadir y modificar datos

Para consultas que no regresan una colección de resultados, como `INSERT`, `UPDATE` y `DELETE`, la extensión SQLite ofrece el método `queyExec()`. El siguiente ejemplo lo muestra en acción, añadiendo un nuevo registro a la tabla *artistas*.

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");

// intenta ejecutar el consulta
// añade un nuevo registro
// datos de salida: "Nuevo artista con el id:8 ha sido añadido."
$sql = "INSERT INTO artistas (artista_nombre, artista_pais) VALUES
('James Blunt', 'UK')";
if (sqlite->consultaExec($sql) == true) {
  echo 'Nuevo artista con el id:' . $sqlite->lastInsertRowid() . 'ha sido
añadido.';
} else {
  echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

Dependiendo de si el consulta se ejecutó con éxito o no, la función `consultaExec()` regresa un valor verdadero o falso; es fácil revisar este valor de respuesta y mostrar un mensaje de éxito o de falla. Si el registro fue insertado en una tabla con un campo `INTEGER PRIMARY KEY`, SQLite automáticamente asignará al registro un número de identificación único. Este número puede ser recuperado con el método `lastInsertRowid()` del objeto SQLiteDatabase, como se mostró en el ejemplo anterior.

PRECAUCIÓN

Para que funcionen los comandos `INSERT`, `UPDATE` y `DELETE`, recuerda que el script PHP *debe* tener privilegios de escritura para el archivo de base de datos SQLite.

Cuando realizas un consulta `UPDATE` o `DELETE`, la cantidad de filas afectadas por éste también puede ser recuperada a través del método `changes()` del objeto SQLiteDatabase. El siguiente ejemplo lo muestra, actualizando los ratings de las canciones en la base de datos y regresando la cuenta de los registros afectados:

```
<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('musica.db') or die ("No fue posible abrir
la base de datos");

// intenta ejecutar el consulta
// actualiza registro
// datos de salida: "3 fila(s) actualizadas."
$sql = "UPDATE canciones SET ex_cancion_rating = 4 WHERE ex_cancion_
rating = 3";
if (sqlite->consultaExec($sql) == true) {
  echo $sqlite->changes() . 'fila(s) actualizadas.';
} else {
  echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

Manejo de errores

Ambos métodos, `query()` y `consultaExec()`, regresan un valor falso si ocurre un error durante la preparación o ejecución del consulta. Es fácil verificar el valor de retorno de estos métodos y recuperar el código correspondiente al error invocando el método `lastError()` del objeto SQLiteDatabase. Desafortunadamente, este código de error no es de gran utilidad por sí mismo; así que, para obtener una descripción literal de lo que sucedió, se debe envolver la invocación `lastError()` en la función `sqlite_error_string()`, como se hizo en la mayoría de los ejemplos anteriores.

Prueba esto 7-3 Crear una lista personal de pendientes

Ahora que ya tienes una idea del manejo de SQLite, ¿qué tal si lo utilizas en una aplicación práctica?: una lista personal de pendientes que puedas consultar y actualizar a través del explorador Web. Esta lista de pendientes te permitirá ingresar tareas y fechas de vencimiento, asignar prioridades a las tareas, editar sus descripciones y marcarlas cuando hayan sido com-

pletadas. Es un poco más complicada que las aplicaciones que has hecho hasta ahora, porque incluye varias partes dinámicas; sin embargo, si has puesto atención hasta este punto, no será muy difícil que comprendas lo que está sucediendo.

Para comenzar, crea una nueva base de datos SQLite llamada *pendientes.db* y añade una tabla vacía para contener la descripción de las tareas y las fechas:

```
shell> sqlite pendientes.db
sqlite> CREATE TABLE tareas (
   ...> id INTEGER PRIMARY KEY,
   ...> nombre TEXT NOT NULL,
   ...> vencimiento TEXT NOT NULL,
   ...> prioridad TEXT NOT NULL
   ...> );
```

Esta tabla tiene cuatro campos: para el ID del registro, el nombre de la tarea, la fecha de vencimiento de ésta y la prioridad que tiene.

El siguiente paso consiste en crear un formulario Web para añadir nuevas tareas a la base de datos (*guarda.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-3: Crear una lista personal de pendientes</title>
    <style type="text/css">
    div#message{
      text-align:center;
      margin-left:auto;
      margin-right:auto;
      width:40%;
      border: solid 2px green
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 7-3: Crear una lista personal de pendientes</h2>
    <h3>Añade una Nueva Tarea</h3>

    <?php
    // si el formulario no ha sido enviado
    // genera un nuevo formulario
    if (!isset($_POST['submit'])) {
    ?>

    <form method="post" action="guarda.php">
      Descripción: <br />
      <input type="text" name="nombre" />
```

```
      <p>
      Fecha de vencimiento (dd/mm/aaaa): <br />
      <input type="text" name="dd" size="2" />
      <input type="text" name="mm" size="2" />
      <input type="text" name="aa" size="4" />
      <p>
      Prioridad: <br />
      <select name="prioridad">
        <option name="Alta">Alta</option>
        <option name="Media">Media</option>
        <option name="Baja">Baja</option>
      </select>
      <p>
      <input type="submit" name="sumbit" value="Guardar" />
    </form>

    <?php
    } else {
      // si el formulario ya fue enviado
      // intenta establecer conexión con la base de datos
      $sqlite = new SQLiteDataBase('pendientes.db') or die ("No fue
posible abrir la base de datos");

      // verifica y limpia los datos de entrada
      $nombre = !empty($_POST['nombre']) ? sqlite_escape_string
($_POST['nombre']) : die('ERROR: Se requiere el nombre de la tarea');
      $dd = !empty($_POST['dd']) ? sqlite_escape_string (int)
($_POST['dd']) : die('ERROR: Se requiere fecha de vencimiento');
      $mm = !empty($_POST['mm']) ? sqlite_escape_string (int)
($_POST['mm']) : die('ERROR: Se requiere fecha de vencimiento');
      $aa = !empty($_POST['aa']) ? sqlite_escape_string (int)
($_POST['aa']) : die('ERROR: Se requiere fecha de vencimiento');
      $fecha = checkdate($mm, $dd, $aa) ? mktime(0, 0, 0, $mm, $dd, $aa)
: die('ERROR: La fecha de vencimiento es inválida');
      $prioridad = !empty($_POST['prioridad']) ? sqlite_escape_string
($_POST['prioridad']) : die('ERROR: Se requiere la prioridad de la
tarea');

      // intenta ejecutar el consulta
      // añade nuevo registro
      $sql = "INSERT INTO tareas (nombre, vencimiento, prioridad) VALUES
('$nombre', '$vencimiento', '$prioridad')";
      if ($sqlite->consultaExec($sql) == true){
        echo '<div id="message"> El registro ha sido añadido con éxito a
la base de datos.</div>';
      } else {
        echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
```

(*continúa*)

```
      }

      // cierra conexión
      unset($sqlite);
    }
    ?>

  </body>
</html>
```

La figura 7-8 muestra el formulario Web.

Cuando el usuario envía este formulario, primero se validan los datos para asegurar que estén presentes todos los campos requeridos. Los datos ingresados en los tres campos correspondientes a la fecha también son verificados con la función PHP `checkdate()` para asegurar que los tres juntos formen una fecha válida. Después, se limpian los datos de entrada al pasarlos por la función `sqlite_escape_string()`, que elimina los caracteres especiales de los datos de entrada automáticamente y los guarda en la base de datos utilizando el consulta `INSERT`.

La figura 7-9 muestra el resultado de añadir con éxito una nueva tarea a la base de datos.

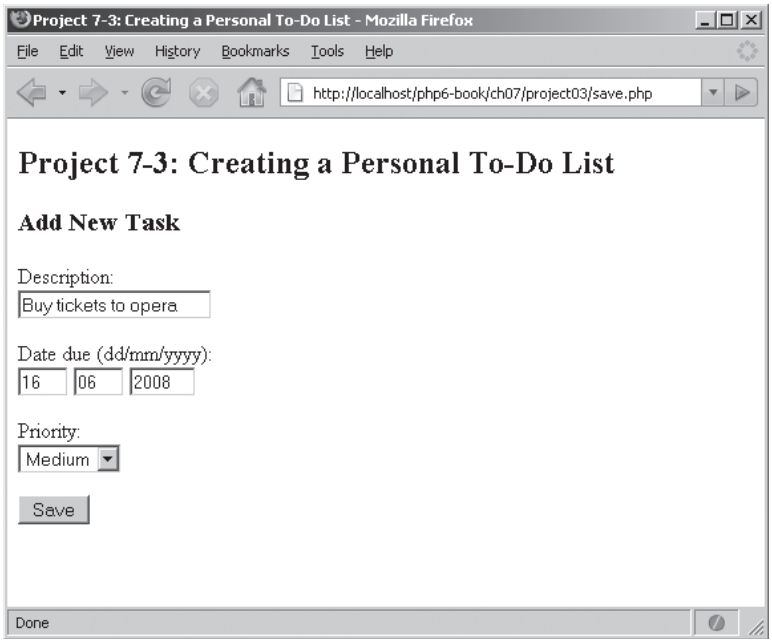


Figura 7-8 Formulario Web para añadir tareas a la base de datos

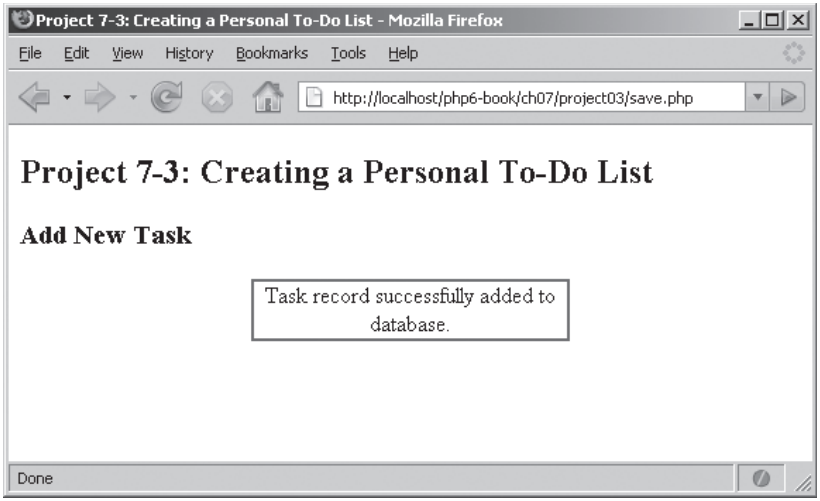


Figura 7-9 El resultado de añadir un nuevo pendiente a la base de datos

Eso es suficiente para ingresar tareas a la lista de pendientes. ¿Qué tal si ahora los vemos? Como probablemente lo habrás adivinado, es cuestión de utilizar una consulta `SELECT` para recuperar todos los registros de la base de datos y luego dar formato a la información resultante de manera que sea apropiada para desplegarse en una página Web. He aquí el código (*lista.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-3: Crear una lista de pendientes personales</title>
    <style type="text/css">
    div#message{
      text-align:center;
      margin-left:auto;
      margin-right:auto;
      width:60%;
      border: solid 2px green
    }
    table{
    border-collapse: collapse;
    width: 500px;
    }
    tr.heading{
    font-weight: bolder;
     }
```

(*continúa*)

```
     td{
    border: 1px solid black;
    padding: 1em;
     }
     tr.high {
       background: #cc1111;
     }
     tr.medium {
       background: #00aaaa;
     }
     tr.low {
       background: #66dd33;
     }
     a {
        color: black;
        border: outset 2px black;
        text-decoration: none;
        padding: 3px;
     }
     </style>
  </head>
  <body>
    <h2>Proyecto 7-3: Crear una lista de pendientes personales</h2>
    <h3>Lista de Tareas</h3>

<?php
// intenta establecer conexión con la base de datos
$sqlite = new SQLiteDataBase('pendientes.db') or die ("No fue posible
abrir la base de datos");

// obtiene registros
// como una tabla HTML
$sql = "SELECT id, nombre, vencimiento, prioridad FROM tareas ORDER BY
vencimiento";
if ($result = $sqlite->query($sql)) {
  if ($result->numRows() > 0) {
    echo "<table>\n";
    echo "  <tr class=\"heading\">\n";
    echo "    <td>Descripción</td>\n";
    echo "    <td>Fecha de vencimiento</td>\n";
    echo "    <td></td>\n";
    echo "  </tr>\n";
    while($row = $result->fetchObject()) {
      echo "  <tr class=\"" . strtolower($row->priority) . "\">\n";
      echo "    <td>" . $row->nombre . "</td>\n";
      echo "    <td>" . date('d M Y', $row->vencimiento) . "</td>\n";
      echo "    <td><a href=\"marca.php?id=" . $row->id . "\"> Marcar
como finalizada</a></td>\n";
      echo "  </tr>\n";
```

```
    }
    echo "</table>";
  } else {
    echo '<div id="message"> No hay tareas en la base de datos.</div>';
  }
} else {
  echo 'ERROR: No fue posible ejecutar consulta: $sql.' . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>

  <p/>
  <a href="guarda.php">Añadir una nueva tarea</a>

  </body>
</html>
```

No hay nada extraño aquí: el script ejecuta la consulta `SELECT` con el método `exec()`, y luego itera sobre la colección de resultados, presentando cada registro encontrado como una fila de una tabla HTML. Advierte que dependiendo del campo *prioridad* de cada registro, la fila de la tabla HTML correspondiente tiene un color diferente: rojo, verde o azul.

La figura 7-10 muestra un ejemplo de los datos de salida.

Verás algo más en la figura: cada elemento de la lista de pendientes viene con una opción `'Marcar como Finalizada'`. Esta opción apunta hacia otro script PHP, *marca.php*, que es responsable de eliminar el registro correspondiente de la base de datos. Mira con atención el URL que apunta a *marca.php*, y verás que el ID del registro también es transmitido, como la variable `$id`. Dentro de *marca.php*, este ID será accesado a través de la matriz `$_GET`, como `$_GET['id']`.

¿Qué hace *marca.php*? Nada muy complejo, simplemente utiliza el ID del registro transmitido por `$_GET` para formular y ejecutar un consulta `DELETE`, que elimina el registro de la base de datos. He aquí el código (*marca.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-3: Crear una lista de pendientes personales</title>
    <style type="text/css">
    div#message {
      text-align:center;
      margin-left:auto;
      margin-right:auto;
      width:40%;
      border: solid 2px green
```

(*continúa*)

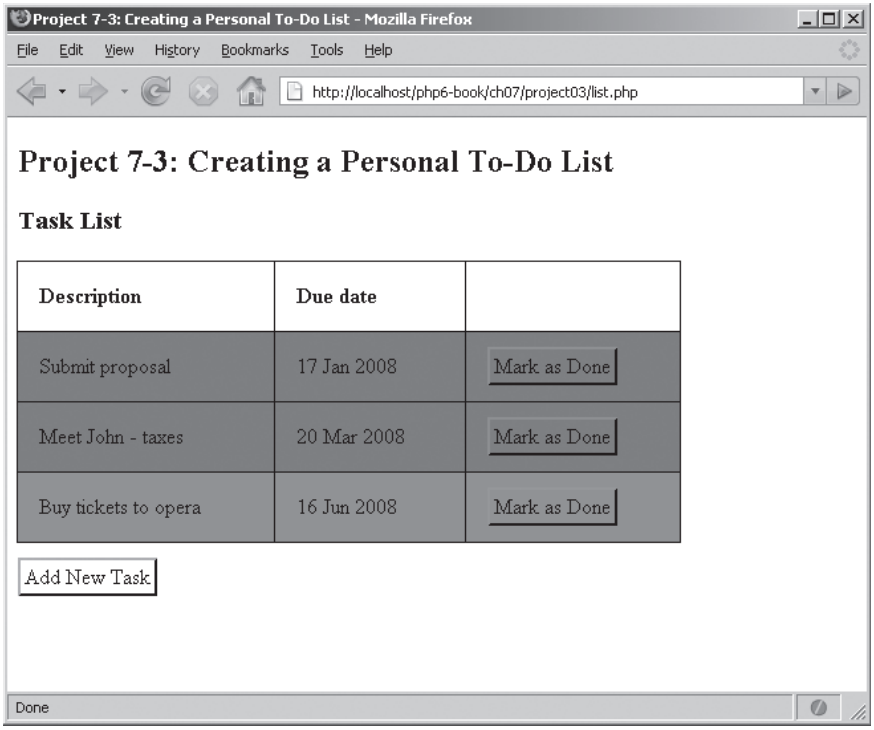


Figura 7-10 Página Web que muestra los elementos de la lista de pendientes, ordenados por fecha de vencimiento

```
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 7-3: Crear una lista de pendientes personales</h2>
    <h3>Elimina la Tarea Terminada</h3>

    <?php
    if (isset($_GET['id'])) {
      // intenta establecer conexión con la base de datos
      $sqlite = new SQLiteDataBase('pendientes.db') or die ("No fue
posible abrir la base de datos");

      // verifica y limpia los datos de entrada
      $id = !empty($_GET['id']) ? sqlite_escape_string((int)$_GET['id'])
: die('ERROR: Se requiere el ID de la tarea');

      // elimina registro
      $sql = "DELETE FROM tareas WHERE id = '$id'";
```

```
      if ($sqlite->consultaExec($sql) == true) {
        echo '<div id="message">Registro de la tarea eliminado
exitosamente de la base de datos.</div>';
      } else {
        echo "ERROR: No fue posible ejecutar $sql." . sqlite_error_
string($sqlite->lastError());
      }

      // cierra conexión
      unset($sqlite);
    } else {
      die ('ERROR: Se requiere el ID de la tarea');
    }
    ?>

  </body>
</html>
```

La figura 7-11 muestra los datos de salida al eliminar de manera correcta una tarea de la base de datos.

Y ahora, cuando revisas *lista.php*, tu lista de pendientes ya no mostrará el elemento que has marcado como terminado. Sencillo, ¿no es cierto?

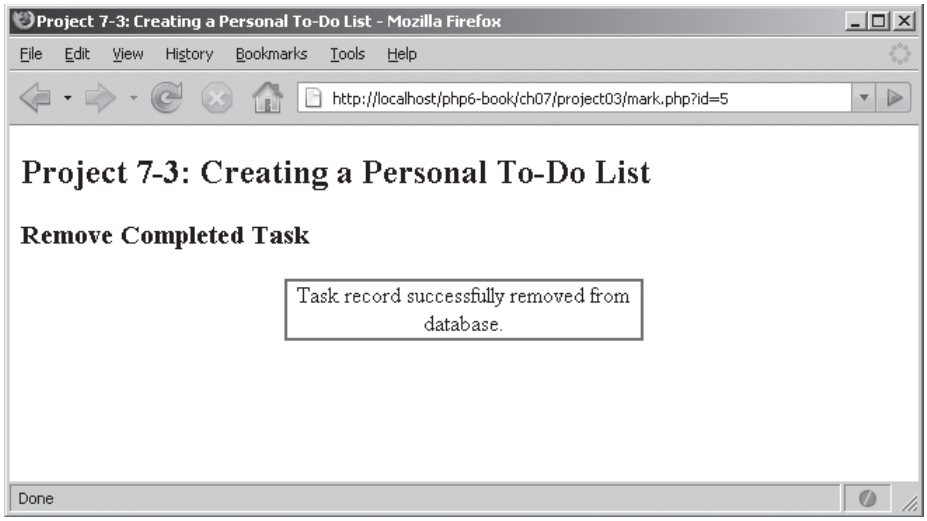


Figura 7-11 El resultado de marcar una tarea como realizada

Utilizar las extensiones PDO de PHP

En las secciones anteriores aprendiste a integrar tus aplicaciones PHP con las bases de datos MySQL y SQLite. Como ya habrás notado, las extensiones MySQLi y SQLite utilizan diferentes nombres de métodos para realizar su trabajo; como resultado, conmutar de una base de datos a otra implica, en esencia, volver a escribir todo el código de base de datos para utilizar los nuevos métodos. Por ello PHP ofrece una extensión neutral para bases de datos: objetos de datos de PHP (PDO), que brinda gran portabilidad y puede reducir de manera significativa el esfuerzo que implica conmutar de un sistema de base de datos a otro.

La siguiente sección aborda la extensión PDO con gran detalle, proporcionando información sobre la manera en que se utiliza para conectarse a diferentes sistemas de bases de datos, realizar consultas, procesar colecciones de resultados, además de manejar errores de consultas y conexión. La mayor parte de los ejemplos dan por hecho que se trabaja con una base de datos MySQL; sin embargo, como verás, los programas basados en PDO requieren mínimas modificaciones para trabajar con otros sistemas de bases de datos, incluyendo SQLite.

Recuperar datos

PDO trabaja proporcionando un conjunto estándar de funciones para realizar operaciones comunes de base de datos, como conexión, consultas, procesamiento de colecciones de resultados y manejo de errores; internamente traduce estas funciones a las invocaciones API nativas comprensibles para la base de datos en uso. Para mostrar la manera en que funciona, examina el siguiente ejemplo, que ejecuta una consulta `SELECT` y despliega los registros encontrados:

```
<?php
// intenta establecer conexión
try {
   $pdo = new PDO('mysql:dbname=musica;host=localhost', 'usuario',
'contra');
} catch (PDOException $e) {
   die("Error: No fue posible conectar: " . $e->getMessage());
}

// crea y ejecuta consulta SELECT
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $pdo->query($sql)) {
  while($row = $result->fetch()) {
    echo $row[0] . ":" . $row[1] . "\n";
  }
} else {
  echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo
->errorInfo());
```

```
}

// cierra conexión
unset($pdo);
?>
```

Como lo muestra este ejemplo, utilizar PDO para obtener datos de una base implica pasos semejantes a los que has visto en secciones previas:

1. El primer paso consiste en inicializar una instancia de la clase PDO y pasar al constructor del objeto tres argumentos: una cadena con el nombre del origen de datos (DSN, Data Source Name), indicando el tipo de base de datos al que se va a conectar, además de otras opciones específicas de la base, un nombre de usuario válido reconocido por la base en cuestión y su correspondiente contraseña. La cadena DSN varía de una a otra base de datos; por lo general puedes obtener el formato exacto para esta cadena de la documentación de la base que estés utilizando.

   La tabla 7-4 muestra algunos formatos de cadenas DSN comunes.

   Si el intento de conexión fracasa, se generará una excepción; esta excepción puede ser recogida y manejada utilizando el mecanismo de manejo de excepciones de PHP (puede obtenerse más información sobre el manejo de excepciones en el capítulo 10 de este libro).

2. Suponiendo que la conexión fue correcta, el siguiente paso consiste en formular una consulta SQL y ejecutarla utilizando el método `query()` de PDO. El valor de regreso de este método es una colección de resultados, representada por el objeto PDOStatement. El contenido de la colección de resultados puede procesarse utilizando el método `fetch()`

Base de datos	Cadena DSN
MySQL	`'mysql:host=`*host*`; port=`*puerto*`;dbname=`*db*`'`
SQLite	`'sqlite:`*ruta/a/archivo/basededatos*`'`
PostgreSQL	`'pgsql:host=`*host* `port=`*puerto* `dbname=`*db* `usuario=`*usuario* `password=`*contra*`'`
Oracle	`'oci:dbname=//`*host*`:`*puerto*`/`*db*`'`
Firebird	`'firebird:Usuario=`*usuario*`;Password=`*contra*`; Database=`*db*`; DataSource=`*host*`;Port=`*puerto*`'`

Tabla 7-4 Cadenas DSN comunes

del objeto, que regresa el siguiente registro en la colección de resultados como una matriz (tanto asociativa como indexada). Es posible acceder a los campos individuales del registro como elementos de la matriz en un bucle, utilizando el índice del campo o su nombre.

Pregunta al experto

P: ¿Cómo calculo el número de registros en mi colección de resultados con PDO?

R: A diferencia de las extensiones MySQL y SQLite, PDO no ofrece métodos ni propiedades integrados para recuperar directamente el número de registros en una colección de resultados. Esto se debe a que no todos los sistemas de bases de datos regresan esta información, por lo que PDO no puede proporcionar esta información de manera portátil. Sin embargo, si necesitas esta información, el manual PHP recomienda ejecutar la declaración `SELECT COUNT(*)...` para obtenerla, con la consulta deseada y recuperar el primer campo de la colección de resultados, que contendrá el número de registros que coinciden con la consulta. Para mayor información, revisa en el análisis de www.php.net/manual/en/function.PDOStatement-rowCount.php.

El método `fetch()` del objeto PDOStatement acepta un modificador adicional, que controla la manera en que se realiza la búsqueda en la colección de resultados. Algunos valores aceptados por este modificador se muestran en la tabla 7-5.

Modificador	Lo que hace
`PDO::FETCH_NUM`	Regresa cada registro como una matriz numérica indexada
`PDO::FETCH_ASSOC`	Regresa cada registro como una matriz asociativa cuya clave es el nombre de campo
`PDO::FETCH_BOTH`	Regresa cada registro de ambas maneras, como una matriz numérica indexada y como una matriz asociativa (el valor por defecto)
`PDO::FETCH_OBJ`	Regresa cada registro como un objeto con propiedades correspondientes a los nombres de campo
`PDO::FETCH_LAZY`	Regresa cada registro como una matriz numérica indexada, como una matriz asociativa y como un objeto

Tabla 7-5 Modificadores del método `fetch()` de PDO

El siguiente ejemplo muestra estos modificadores en acción:

```
<?php
// intenta establecer una conexión
try {
   $pdo = new PDO('mysql:dbname=musica;host=localhost', 'usuario',
'contra');
} catch (PDOException $e) {
   die("Error: No fue posible conectar: " . $e->getMessage());
}

// crea y ejecuta consulta SELECT
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $pdo->query($sql)){

  // regresa registro como matriz numérica
  $row = $result->fetch(PDO::FETCH_NUM);
  echo $row[0] . ":" . $row[1] . "\n";

  // regresa registro como matriz asociativa
  $row = $result->fetch(PDO::FETCH_ASSOC);
  echo $row['artista_id'] . ":" . $row['artista_nombre'] . "\n";

  // regresa registro como un objeto
  $row = $result->fetch(PDO::FETCH_OBJ);
  echo $row->artista_id . ":" . $row->artista_nombre . "\n";

  // regresa registro utilizando una combinación de estilos
  $row = $result->fetch(PDO::FETCH_LAZY);
  echo $row['artista_id'] . ":" . $row->artista_nombre . "\n";

} else {
  echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo-
>errorInfo());
}

// cierra conexión
unset($pdo);
?>
```

Añadir y modificar datos

PDO también facilita la ejecución de consultas `INSERT`, `UPDATE` y `DELETE` con su método `exec()`. Este método, que está diseñado para instrucciones que de alguna manera modifican la base de datos, regresa la cantidad de registros afectados por el consulta. He aquí un ejemplo de su uso para insertar y eliminar un registro:

```
<?php
// intenta establecer una conexión
try {
   $pdo = new PDO('mysql:dbname=musica;host=localhost', 'usuario',
'contra');
} catch (PDOException $e) {
   die("ERROR: No fue posible conectar: " . $e->getMessage());
}

// crea y ejecuta consulta INSERT
$sql = "INSERT INTO artistas (artista_nombre, artista_pais) VALUES
('Luciano Pavarotti', 'IT')";
$ret = $pdo->exec($sql);
if ($ret === false) {
  echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo
->errorInfo());
} else {
  $id = $pdo->lastInsertId();
  echo 'Nuevo artista con id: ' . $id . 'ha sido añadido.';
}

// crea y ejecuta un consulta DELETE
$sql = "DELETE FROM artistas WHERE artista_pais = 'IT'";
$ret = pdo->exec($sql);
if ($ret === false) {
  echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo
->errorInfo());
} else {
  echo 'Eliminados ' . $ret . ' registros.';
}

// cierra conexión
unset($pdo);
?>
```

Este código utiliza el método `exec()` dos veces, primero para insertar un nuevo registro y luego para eliminar registros que coincidan con una condición específica. Si la consulta transmitida a `exec()` falla, `excec()` regresa un valor falso; de lo contrario, regresa la cantidad de registros afectados por la consulta. Advierte también que el script utiliza el método `lastInsertId()` del objeto PDO, el cual regresa el ID generado por el último comando `INSERT` en caso de que la tabla contenga un campo de autoincremento.

TIP

Si no hay registros afectados por la ejecución de la consulta realizada por el método `exec()`, éste regresará un valor igual a cero. No confundas este valor con el valor booleano "false" (falso) regresado por `exec()` cuando falla la consulta. Para evitar confusiones, el manual de PHP recomienda siempre probar el valor de retorno de `exec()` con el operador === en lugar de utilizar el operador ==.

Utilizar declaraciones preparadas

PDO también da soporte a declaraciones preparadas, mediante sus métodos `prepare()` y `execute()`. El siguiente ejemplo lo ilustra, utilizando una declaración preparada para añadir varias canciones a la base de datos:

```
<?php
// define los valores que serán insertados
$canciones = array(
  array('Voulez-Vous', 2, 5),
  array('Take a Chance on Me', 2, 3),
  array('I Have a Dream', 2, 4),
  array('Thank You For The Music', 2, 4),
);

// intenta establecer una conexión
try {
   $pdo = new PDO('mysql:dbname=musica;host=localhost', 'usuario',
'contra');
} catch (PDOException $e) {
   die("ERROR: No fue posible conectar: " . $e->getMessage());
}

// crea y ejecuta consulta SELECT
sql = "INSERT INTO canciones (cancion_titulo, ex_cancion_artista, ex_
cancion_rating) VALUES (?, ?, ?)";
if ($stmt = $pdo->prepare($sql)) {
  foreach ($canciones as $s) {
    $stmt->bindParam(1, $s[0]);
    $stmt->bindParam(2, $s[1]);
    $stmt->bindParam(3, $s[2]);
    if ($stmt->execute()) {
      echo "Nueva canción con id: " . $pdo->lastInsertId() . "ha sido
añadida. \n";
    } else {
      echo "ERROR: No fue posible ejecutar consulta: $sql. " . print_r
($pdo->errorInfo());
    }
  }
} else {
  echo "ERROR: No fue posible preparar consulta: $sql. " . print_r($pdo
->errorInfo());
}

// cierra conexión
unset($pdo);
?>
```

Si comparas este último script con uno similar de MySQLi en las secciones anteriores, verás una similitud muy marcada. Como antes, este script también define una consulta SQL modelo que contiene una declaración `INSERT` formada por contenedores en lugar de valores, y luego convierte este modelo en una declaración preparada utilizando el método `prepare()` del objeto PDO.

En caso de tener éxito, `prepare()` regresa un objeto PDOStatement que representa la declaración preparada. Los valores que serán interpolados en la declaración se unen entonces a los contenedores invocando repetidamente el método `bindParam()` del objeto PDOStatement con dos argumentos, la posición del contenedor y la variable a la que será unida. Una vez que las variables están unidas, el método `execute()` del objeto PDOStatement se ocupa de ejecutar la declaración preparada con los valores correctos.

NOTA

Cuando utilices declaraciones preparadas con PDO, es importante considerar que los beneficios de este tipo de declaraciones sólo estarán disponibles si la base de datos con la que trabajas soporta estas declaraciones de forma nativa. Para bases de datos que no soportan las declaraciones preparadas, PDO convertirá internamente las invocaciones para `prepare()` y `execute()` en declaraciones SQL estándar y en estos casos no obtendrás ningún beneficio.

Manejar errores

Cuando se realizan operaciones de bases de datos con PDO, pueden ocurrir errores durante la fase de conexión o durante la ejecución de la consulta. PDO ofrece herramientas robustas para manejar ambos tipos de errores.

Errores de conexión

Si PDO no puede conectarse con la base de datos especificada utilizando el DSN, nombre de usuario y contraseña proporcionados, automáticamente generará una PDOException. Esta excepción puede capturarse utilizando el mecanismo estándar de manejo de excepciones de PHP (abordado con detalle en el capítulo 10), y es posible desplegar un mensaje de error explicando las causas del mismo en el objeto excepción.

Errores de ejecución de la consulta

Si ocurre un error durante la preparación o ejecución de la consulta, PDO proporciona información sobre el mismo con el método `errorInfo()`. Regresa una matriz que contiene tres elementos: el código de error SQL, el código de error de la base de datos y un mensaje de error legible para los humanos (también generado por la base de datos en uso). Es fácil procesar esta matriz y presentar los elementos apropiados que contiene desde la sección de manejo de errores de tu script.

Prueba esto 7-4 Construir un formulario de inicio de sesión

Pongamos ahora en práctica PDO con otra aplicación práctica, una que encontrarás una y otra vez durante tu desarrollo como programador PHP: un formulario de ingreso. Esta aplicación generará un formulario Web para que los usuarios ingresen su nombre de usuario y contraseña; luego verificará estos datos de entrada contra los almacenados en la base de datos y permitirá o rechazará su ingreso. Primero, la aplicación utilizará la base de datos MySQL; sin embargo, después verás qué tan portátil es el código PDO, cuando conmutemos la aplicación hacia una base de datos SQLite.

Utilizar una base de datos MySQL

Para comenzar, arranca el programa cliente de línea de comandos de MySQL y crea una tabla que contenga los nombres de usuario y las contraseñas, como sigue:

```
mysql> CREATE DATABASE app;
Query OK, 0 rows affected (0.07 sec)
mysql> USE app;
Query OK, 0 rows affected (0.07 sec)
mysql> CREATE TABLE usuarios (
    -> id int(4) NOT NULL AUTO_INCREMENT,
    -> nombredeusuario VARCHAR(255) NOT NULL,
    -> contrasena VARCHAR(255) NOT NULL,
    -> PRIMARY KEY (id));
Query OK, 0 rows affected (0.07 sec)
```

En este punto es buena idea “sembrar” la tabla ingresando algunos nombres de usuario y contraseñas. Para simplificarlo, los nombres de usuario serán iguales a las contraseñas, pero cifradas para que estén a salvo de mirones casuales (y de hackers no tan casuales):

```
mysql> INSERT INTO usuarios (nombredeusuario, contrasena)
    -> VALUES ('john', '$1$Tk0.gh4.$42EZDbQ4mOfmXMq.0m1tS1');
Query OK, 1 row affected (0.21 sec)

mysql> INSERT INTO usuarios (nombredeusuario, contrasena)
    -> VALUES ('jane', '$1$.15.tR/.$XK1KW1Wzqy0UuMFKQDHH00');
Query OK, 1 row affected (0.21 sec)
```

Ahora todo lo que se necesita es un formulario de inicio de sesión, y algo de código PHP para leer los valores ingresados en el formulario y compararlos contra los valores almacenados en la base de datos. He aquí el código (*ingreso.php*):

(*continúa*)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 7-4: Construir un formulario de ingreso</title>
  </head>
  <body>
    <h2>Proyecto 7-4: Construir un formulario de ingreso</h2>
<?php
    // si el formulario no ha sido enviado
    // genera un nuevo formulario
    if (!isset($_POST['submit'])) {
?>
    <form method="post" action="ingreso.php">
      Nombre de Usuario: <br />
      <input type="text" name="nombredeusuario" />
      <p>
      Contraseña: <br />
      <input type="contraword" name="contrasena" />
      <p>
      <input type="submit" name="submit" value="Ingresar" />
    </form>

<?php
    // si el formulario ha sido enviado
    // verifica los datos proporcionados
    // contra la base de datos
    } else {
      $nombredeusuario = $_POST['nombredeusuario'];
      $contrasena = $_POST['contrasena'];

      // verifica datos de entrada
      if(empty($nombredeusuario)) {
        die('ERROR: Por favor escriba su nombre de usuario');
      }
      if(empty($contrasena)) {
        die('ERROR: Por favor escriba su contraseña');
      }

      // intenta establecer conexión con la base de datos
      try {
         $pdo = new PDO('mysql:dbname=app;host=localhost', 'usuario',
'contra');
      } catch (PDOException $e) {
         die("ERROR: No fue posible conectar: " . $e->getMessage());
      }

      // limpia los caracteres especiales de los datos de entrada
      $nombredeusuario = $pdo->quote($nombredeusuario);
```

```
      // verifica si existe el nombre de usuario
      $sql = "SELECT COUNT(*) FROM usuarios WHERE nombredeusuario =
$nombredeusuario";
      if($result = $pdo->query($sql)) {
        $row = $result->fetch();
        // si es positivo, busca la contraseña cifrada
        if($row[0] == 1) {
          $sql = "SELECT contrasena FROM usuarios WHERE nombredeusuario =
$nombredeusuario";
          // cifra la contraseña ingresada en el formulario
          // la verifica contra la contraseña cifrada que reside en la
base de datos
          // si ambas coinciden, la contraseña es correcta
          if ($result = $pdo->query($sql)) {
            $row = $result->fetch();
            $salt = $row[0];
            if (crypt($contrasena, $salt) == $salt) {
              echo 'Sus credenciales de acceso fueron verificadas
positivamente.';
            } else {
              echo 'Ha ingresado una contraseña incorrecta.';
            }
          } else {
            echo "ERROR: No fue posible ejecutar $sql. " . print_r
($pdo->errorInfo());
          }
        } else {
          echo 'Ha ingresado un nombre de usuario incorrecto.';
        }
      } else {
        echo "ERROR: No fue posible ejecutar $sql. " . print_r
($pdo->errorInfo());
      }

      // cierra conexión
      unset($pdo);
    }
?>
  </body>
</html>
```

La figura 7-12 muestra la apariencia del formulario de ingreso.

(*continúa*)

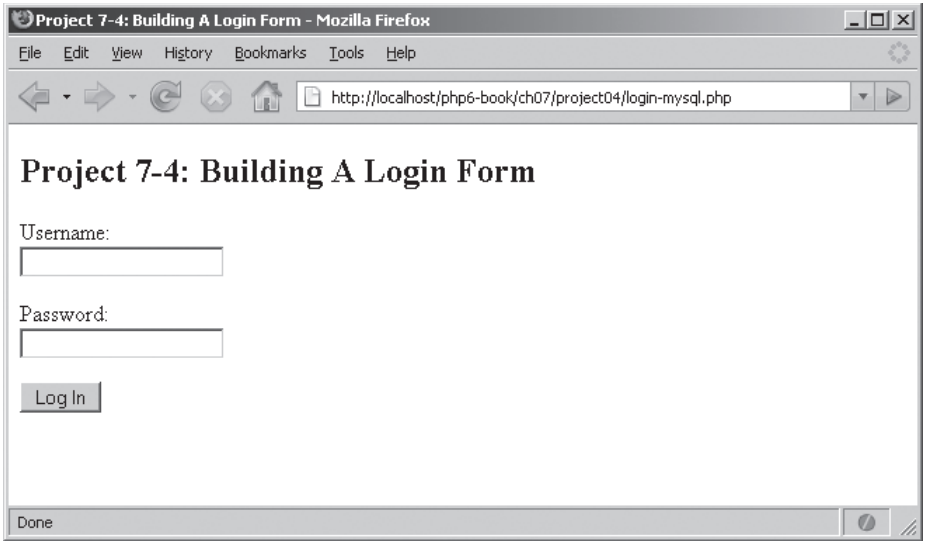


Figura 7-12 Formulario de ingreso en una página Web

Cuando se transmite este formulario, la segunda mitad del script entra en juego. Se realizan varios pasos:

1. Verifica los datos de entrada del formulario para asegurar que el nombre de usuario y la contraseña estén presentes, y detiene la ejecución del script con un mensaje de error si falta cualquiera de los dos. También limpia los caracteres especiales de los datos de entrada utilizando el método `quote()`.
2. Abre una conexión a la base de datos y ejecuta la consulta `SELECT` para verificar si existe una coincidencia con la información almacenada en la base de datos. En caso de que esta verificación dé como resultado un valor falso, genera el mensaje de error "Ha ingresado un nombre de usuario incorrecto".
3. Si el nombre de usuario existe, entonces el script procede a revisar la contraseña. Dado que ésta se cifró utilizando un esquema de cifrado de una sola vía, esta verificación no puede realizarse directamente; la única manera de realizar la tarea es volver a cifrar la contraseña del usuario, a partir de la manera en que ha sido ingresada en el formulario, y comparar esta versión contra la que se encuentra almacenada en la base de datos. En caso de que ambas cadenas de caracteres cifradas coincidan, significa que la contraseña ingresada es correcta.

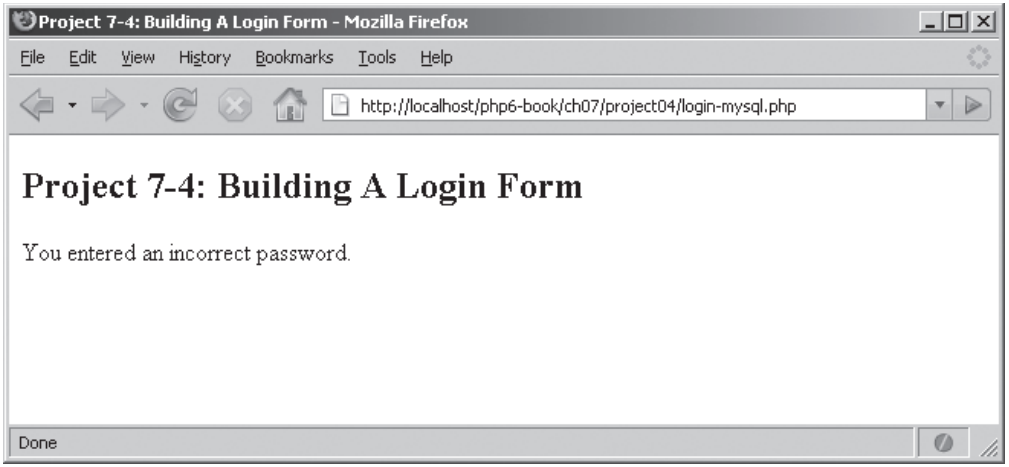


Figura 7-13 El resultado de ingresar una contraseña incorrecta en el formulario de registro

4. Dependiendo del resultado de la prueba, el script genera el mensaje '`Ha ingresado una contraseña incorrecta.`' o bien '`Sus credenciales de acceso fueron verificadas positivamente`'.

La figura 7-13 muestra el resultado de ingresar una contraseña incorrecta y la figura 7-14 muestra el resultado de ingresar correctamente la contraseña.

Figura 7-14 El resultado de ingresar una contraseña válida en el formulario de registro

Pregunta al experto

P: ¿Cómo genero la contraseña cifrada que se utiliza en el ejemplo?

R: Las cadenas de contraseña cifradas que se utilizan en el ejemplo fueron generadas por la función PHP `crypt()`, que utiliza un esquema de cifrado en un solo sentido que aplica a cualquier cadena de caracteres transmitida. El cifrado en un solo sentido hace que la contraseña original sea irrecuperable (de ahí el término "en un solo sentido"). Este cifrado se basa en una clave única o falseada (*salt*), que puede proporcionarse opcionalmente a la función como segundo argumento; en ausencia de ésta, PHP genera de manera automática una clave falseada (*salt*).

La contraseña original ya no es recuperable después de pasar por la función `crypt()`, por lo que realizar después la validación de la contraseña contra el valor proporcionado por el usuario es un proceso de dos pasos: primero, volver a cifrar el valor proporcionado por el usuario con la misma clave falseada utilizada en el proceso original de cifrado y luego comprobar si las dos cadenas cifradas coinciden. Esto es precisamente lo que hace el formulario del ejemplo cuando procesa la información.

Conmutar a una base de datos diferente

Ahora, supongamos que por causas de fuerza mayor te ves forzado a cambiar de MySQL a SQLite. Lo primero que necesitas hacer es crear una tabla en la base de datos para almacenar toda la información de las cuentas de usuario. Así que arranca tu programa cliente SQLite y replica la base de datos creada en la sección anterior (nombra el archivo de base de datos *app.db*):

```
sqlite> CREATE TABLE usuarios (
     -> id INTEGER PRIMARY KEY,
     -> nombredeusuario TEXT NOT NULL,
     -> contrasena TEXT NOT NULL,
     -> );

sqlite> INSERT INTO usuarios (nombredeusuario, contrasena)
     -> VALUES ('john', '$1$Tk0.gh4.$42EZDbQ4mOfmXMq.0m1tS1');
sqlite> INSERT INTO usuarios (nombredeusuario, contrasena)
     -> VALUES ('jane', '$1$.15.tR/.$XK1KW1Wzqy0UuMFQDHH00');
```

El siguiente paso consiste en actualizar el código de tu aplicación para que se comunique con esta base de datos en lugar de hacerlo con la base de MySQL. Como estás utilizando PDO, este cambio consiste en cambiar una (así es, *una*) línea en tu script PHP: el DSN transmitido al constructor del objeto PDO. He aquí el segmento relevante del código:

```
<?php
     // intenta establecer conexión con la base de datos
     try {
```

```
        $pdo = new PDO('sqlite:app.db');
      } catch (PDOException $e) {
         die("Error: No fue posible conectar: " . $e->getMessage());
      }
?>
```

Y ahora, cuando ingreses, tu aplicación trabajará exactamente como lo hizo antes, excepto que ahora utilizará la base de datos SQLite en lugar de MySQL. ¡Inténtalo por ti mismo y lo verás!

Esta portabilidad es precisamente la razón por la que PDO está ganando popularidad entre los desarrolladores de PHP; hace que todo el proceso de conmutar entre bases de datos sea demasiado sencillo, reduciendo tanto el desarrollo y el tiempo de prueba, como los costos asociados a esas tareas.

Resumen

El soporte para bases de datos de PHP es una de las grandes razones de su popularidad, y este capítulo cubrió todo el terreno necesario para que comiences a trabajar con esta importante característica del lenguaje. El capítulo comenzó presentándote una introducción a los conceptos básicos de las bases de datos, y enseñándote las bases del lenguaje estructurado de consultas (SQL). Pasamos rápidamente a presentar las extensiones PHP para bases de datos más populares: las extensiones MySQLi y las extensiones SQLite, además de la extensión de objetos de datos de PHP (PDO). Los temas abordados incluyeron información práctica y ejemplos de código para realizar consultas y modificaciones a las bases de datos con PHP, con el fin de que comiences a utilizar estas extensiones en tu programación diaria.

Para leer más acerca de los temas abordados en este capítulo, te recomiendo que visites los siguientes vínculos:

- Conceptos de bases de datos, en **www.melonfire.com/community/columns/trog/article.php?id=52**
- Bases de SQL, en **www.melonfire.com/community/columns/trog/article.php?id=39** y **www.melonfire.com/community/columns/trog/article.php?id=44**
- Más información sobre fusiones SQL, en **www.melonfire.com/community/columns/trog/article.php?id=148**
- El sitio Web de MySQL, en **www.mysql.com**
- El sitio Web de SQLite, en **www.sqlite.org**
- Extensiones PHP MySQLi, en **www.php.net/mysqli**
- Extensiones PHP SQLite, en **www.php.net/sqlite**
- Extensiones PHP PDO, en **www.php.net/pdo**

✓ Autoexamen Capítulo 7

1. Marca las siguientes declaraciones como verdaderas o falsas:

A Las fusiones de tablas con SQL sólo se realizan entre los campos de llave primaria y llave externa.

B El soporte de PHP para MySQL es reciente.

C La cláusula `ORDER BY` es utilizada para ordenar los campos de una colección de resultados SQL.

D Los campos `PRIMARY KEY` pueden aceptar valores nulos.

E Es posible reescribir el valor generado por SQLite para un campo `INTEGER PRIMARY KEY`.

F Las declaraciones preparadas pueden utilizarse únicamente para las operaciones `INSERT`.

2. Identifica correctamente el comando SQL para cada una de las siguientes operaciones de base de datos:

A Borrar una base de datos.

B Actualizar un registro.

C Borrar un registro.

D Crear una tabla.

E Seleccionar una base de datos para ser utilizada.

3. ¿Qué significa normalizar una base de datos y por qué es útil?

4. Menciona una ventaja y una desventaja de utilizar una biblioteca de abstracción como PDO en lugar de utilizar extensiones nativas de la base de datos.

5. Utilizando la extensión PDO, escribe un script PHP para añadir nuevas canciones a la tabla *canciones* desarrollada en este capítulo. Permite que los usuarios seleccionen el artista de la canción y el rating de una lista de selección desplegable, cuyo contenido sea alimentado por las tablas *artista* y *ratings*.

6. Utilizando la extensión MySQLi de PHP, escribe un script para crear una nueva tabla de base de datos con cuatro campos que tú selecciones. Después, realiza la misma tarea con una base de datos SQLite utilizando la extensión SQLite de PHP.

7. Alimenta manualmente la base de datos MySQL creada en la pregunta anterior con 7 o 10 registros de tu elección. Después, escribe un script basado en PDO que lea el contenido de esta tabla y que migre el contenido encontrado en ella a la tabla de la base de datos SQLite también creada en la pregunta anterior. Utiliza una declaración preparada.

Capítulo 8

Trabajar con XML

Habilidades y conceptos clave

- Comprender las tecnologías y los conceptos básicos de XML
- Comprender las extensiones de PHP SimpleXML y DOM
- Acceder a documentos XML con PHP y manipularlos
- Crear nuevos documentos XML desde cero usando PHP
- Integrar informes RSS de terceros en una aplicación PHP
- Convertir entre SQL y XML utilizando PHP

El lenguaje de marcado extensible (XML, eXtensible Markup Language) es un estándar muy aceptado para intercambio y descripción de datos. Permite que los autores de contenidos "marquen" sus datos con etiquetas personales comprensibles para las computadoras, y por lo mismo, facilita la clasificación y búsqueda de los datos. XML también obliga a dar una estructura formal al contenido, y proporciona un formato portátil que se utiliza para intercambiar información fácilmente entre diferentes sistemas.

PHP incluye soporte a XML desde la versión 4, pero es sólo en la versión 5 cuando las diferentes extensiones XML fueron estandarizadas en PHP para utilizar herramientas de segmentación comunes. Este capítulo te presenta una introducción a dos de las más útiles y poderosas extensiones de procesamiento XML de PHP: SimpleXML y DOM, e incluye numerosos ejemplos de código y ejercicios prácticos para utilizar XML en combinación con las aplicaciones basadas en PHP.

Introducción a XML

Antes de entrar en los detalles de la manipulación de archivos XML con PHP, resultará instructivo ocupar un tiempo familiarizándose con XML. Si eres nuevo en XML, la siguiente sección presenta los fundamentos básicos, incluida una visión panorámica de sus conceptos y tecnologías. Esta información será de utilidad para comprender el material más avanzado en las siguientes secciones.

Aspectos básicos de XML

Comencemos con una pregunta muy básica: ¿qué es XML y por qué es útil?

XML es un lenguaje que ayuda a los autores de documentos a describir datos dentro de éstos, al "marcarlos" con etiquetas personales. Estas etiquetas no provienen de una lista predefinida; en lugar de ello, XML alienta a los autores a crear sus propias etiquetas y su propia

estructura, acoplada a sus necesidades, como una manera de incrementar la flexibilidad y capacidad de uso. Este hecho, junto con las recomendaciones de W3C en 1998, han servido para hacer de XLM una de las maneras más populares para describir y almacenar información estructurada dentro y fuera de Web.

Los datos XML están almacenados en archivos de texto. Esto hace que esos documentos sean muy portátiles, porque todas las computadoras pueden leer y procesar archivos de texto. Esto no sólo facilita la distribución de información, también permite que XML se utilice en una gran cantidad de aplicaciones. Por ejemplo, los formatos de RSS y Atom Weblog son alimentados mediante una estructura XML, lo mismo que JavaScript y XML asincrónico (AJAX, Asynchronous JavaScript and XML) y el protocolo de acceso a objetos simples (SOAP, Simple Object Access Protocol).

Pregunta al experto

P: ¿Qué programas puedo utilizar para crear o ver un archivo XML?

R: En sistemas UNIX/Linux, tanto `vi` como `emacs` son útiles para crear documentos XML, mientras que el Bloc de notas sigue siendo el favorito en los sistemas Windows. Tanto Microsoft Internet Explorer como Mozilla Firefox tienen soporte integrado para XML y pueden leer y desplegar un documento XML como vista de árbol jerárquico.

Anatomía de un documento XML

Internamente, un documento XML está integrado por varios componentes, cada uno de ellos sirve a un propósito específico. Para comprender estos componentes, examina el siguiente documento XML, que contiene una receta para cocinar espagueti a la boloñesa:

```
1.   <?xml version='1.0'?>
2.   <receta>
3.     <ingredientes>
4.       <ingrediente cantidad="250" unidades="gm">Carne de res picada</
ingrediente>
5.       <ingrediente cantidad="200" unidades="gm">Cebollas</ingrediente>
6.       <ingrediente cantidad="75" unidades="ml">Vino rojo</ingrediente>
7.       <ingrediente cantidad="12">Tomates</ingrediente>
8.       <ingrediente cantidad="2" unidades="tbsp">Queso parmesano</ingre-
diente>
9.       <ingrediente cantidad="200" unidades="gm">Espagueti</ingrediente>
10.    </ingredientes>
11.    <método>
```

```
12.       <paso número="1">Corte y fría las cebollas.</paso>
13.       <paso número="2">Añada la carne picada a las cebollas fritas
&amp; continúe friendo.</paso>
14.       <paso número="3">Haga puré los tomates y combínelos con la mezcla
junto con el vino rojo.</paso>
15.       <paso número="4">Hiérvalos durante una hora.</paso>
16.       <paso número="5">Sírvalos encima de una pasta cocinada con queso
parmesano.</paso>
17.     </método>
18.   </receta>
```

Este documento XML contiene una receta, dividida en diferentes secciones; cada una de ellas está "marcada" con etiquetas descriptivas para identificar con toda precisión el tipo de dato que contiene en su interior. Veamos cada una en detalle.

1. Cada documento XML debe iniciar con una declaración que especifique la versión XML que se está utilizando; esta declaración es conocida como *prólogo del documento*, y puede verse en la línea 1 del documento XML anterior. Además del número de la versión, este prólogo del documento puede contener información sobre la codificación de caracteres y la referencia a la definición del tipo de documento (DTD) (para la validación de datos).
2. El prólogo del documento es seguido por una serie anidada de *elementos* (líneas 2 a la 18). Los elementos son básicamente unidades de XML; por lo general, están integrados por un par de etiquetas, una de apertura y otra de cierre, que incluyen cierto contenido en forma de texto. Los nombres de los elementos son definidos por el usuario; deben elegirse con cuidado, porque su intención es describir el contenido que se encuentra entre ellos. Los nombres de los elementos son sensibles a las mayúsculas y deben iniciar con una letra, opcionalmente seguidos por más letras o números. El primer elemento en un documento XML (en el ejemplo anterior, el elemento llamado `<receta>` que ocupa la línea 2) es conocido como *elemento del documento* o *elemento raíz*.
3. Los datos en forma de texto encerrados dentro de los elementos son conocidos, en la terminología de XML, como *datos literales*. Son cadenas de caracteres, números y caracteres especiales (con algunas excepciones: los corchetes angulados(< >) y los signos de unión (&) dentro del texto deben reemplazarse con sus códigos `<`, `>` y `&`, respectivamente, para evitar confusiones en el analizador sintáctico XML cuando lea el documento). La línea 13, por ejemplo, utiliza el código `&`; para representar el signo de unión dentro de los datos literales.
4. Por último, los elementos también pueden contener *atributos*, que son pares nombre-valor que contienen información adicional sobre el elemento. Los nombres de los atributos son sensibles a las mayúsculas y siguen las mismas reglas que los de elementos. El mismo

nombre de atributo no puede utilizarse más de una vez en el mismo elemento, y los valores del atributo deben ir siempre encerrados entre comillas dobles. De la línea 4 a la 9 y de la 12 a la 16 en el documento de ejemplo se muestra el uso de los atributos para proporcionar información descriptiva adicional sobre el elemento al que están adjuntados; por ejemplo, el atributo '`unidades`' especifica la unidad de medida para cada ingrediente.

Los elementos XML también pueden contener otros componentes: nombres de espacios, instrucciones de procesamiento y bloques CDATA. Éstos son un poco más complejos y no los verás en ninguno de los ejemplos utilizados en este capítulo; sin embargo, si estás interesado en saber más sobre ellos, visita los sitios recomendados al final de este capítulo para conocer mayor información al respecto y para conseguir algunos ejemplos.

Pregunta al experto

P: ¿Puedo crear elementos que no contengan nada?

R: Seguro. La especificación XML da soporte a elementos sin contenido y, por lo mismo, no requiere una etiqueta de cierre. Para cerrar estos elementos, simplemente añade una diagonal al final de la etiqueta de apertura, como en el siguiente código:

```
La línea se rompe <br /> aquí.
```

XML bien formado y válido

La especificación XML hace una importante distinción entre los documentos bien formados y los válidos.

- Un *documento bien formado* sigue todas las reglas para los nombres de elementos y atributos, contiene todas las declaraciones esenciales, incluye un (y sólo un) elemento raíz y sigue correctamente la jerarquía de elementos anidados debajo de este elemento. Todos los documentos XML que verás en este capítulo son documentos bien formados.
- Un *documento válido* es aquel que respeta todas las condiciones de un documento bien formado y además se apega a las reglas adicionales expresadas en una definición de tipo de documento (DTD) o esquema XML. Este capítulo no aborda ni las DTD ni los esquemas XML en detalle, por lo que no verás ningún ejemplo de este tipo de documento; sin embargo, encontrarás muchos ejemplos en línea y en los vínculos sugeridos al final de este capítulo.

Métodos de segmentación de XML

Por lo general, un documento XML es procesado por un programa de aplicación llamado analizador sintáctico XML. Éste lee el documento XML usando uno de dos métodos: simple API para XML (SAX) o el modelo de objeto de documento (DOM, Document Object Model):

- Un analizador sintáctico SAX opera recorriendo de manera secuencial un documento XML de principio a fin, e invocando funciones específicas definidas por el usuario mientras encuentra diferentes tipos de constructores XML. Así, por ejemplo, un analizador sintáctico SAX puede estar programado para invocar una función que procese un atributo, otro para procesar un elemento de arranque y un tercero para procesar los datos literales. Las funciones invocadas de esta manera son las responsables de procesar, en realidad, el constructor XML encontrado y cualquier información almacenada ahí.
- Un analizador sintáctico DOM, por otra parte, funciona leyendo el documento XML completo de una sola vez y convirtiéndolo en un árbol jerárquico estructurado en la memoria. Luego, es posible programar el analizador sintáctico para recorrer el árbol, brincando entre las diferentes "ramas" para acceder a piezas de información específicas.

Ambos métodos tienen pros y contras: SAX lee los datos XML en "fragmentos" y es eficiente para archivos grandes, pero requiere que el programador cree funciones personalizadas para manejar los diferentes elementos en un archivo XML. DOM requiere menos personalización, pero puede consumir mucha memoria en poco tiempo por sus acciones y, por lo mismo, no es recomendable para documentos XML grandes. La elección de uno u otro método depende en gran medida de las necesidades de la aplicación de que se trate.

Tecnologías XML

Al tiempo que aumenta la popularidad del lenguaje XML, también lo hace la lista de tecnologías que lo utilizan. He aquí unas cuantas de ellas de las que seguramente has escuchado:

- **Esquema XML** Define la estructura y el formato de documentos XML, permitiendo mayor flexibilidad en la validación y soporte de los tipos de datos, la herencia, el agrupamiento y la vinculación con bases de datos.
- **XLink** Especificación para vincular entre sí estructuras de datos XML. Permite la aplicación de tipos de vínculos más sofisticados que los hipervínculos regulares de HTML, incluidos vínculos con varias etiquetas.
- **XPointer** Especificación para navegar por el árbol jerárquico estructurado de un documento XML, para localizar con facilidad elementos, atributos y otras estructuras de datos dentro del documento.

- **XLS** El Lenguaje extensible de hojas de estilo (XLS, eXtensible Styleshet Language) aplica reglas de formato a los documentos XML y los "transforma" de un formato a otro.
- **XHTML** Combina la precisión de marcado XML con las sencillas etiquetas HTML para crear una versión nueva y más estándares de compilación para HTML.
- **XForms** Separa la información recopilada en un formulario Web de la apariencia del formulario, con lo que permite una validación más rígida, además de que permite el fácil reciclaje de formularios en diferentes medios.
- **XML Query** Permite que los desarrolladores apliquen consultas a un documento XML y generen colecciones de resultados, de manera muy similar a la aplicación de SQL para recuperar registros de una base de datos.
- **XML Encryption y XML Signature** La primera representa una serie de medios para cifrar y descifrar documentos XML, además de representar los datos resultantes. Está relacionada muy estrechamente con la segunda, XML Signature, que proporciona un medio para representar y verificar firmas digitales con XML.
- **SVG** Scalable Vector Graphics o imágenes vectoriales escalables utiliza XML para describir vectores o convertir información en imágenes, con soporte para máscaras alfa, filtros, rutas de acceso y transformaciones.
- **MathML** Utiliza XML para describir expresiones o fórmulas matemáticas, con el fin de representarlas fácilmente en exploradores Web.
- **SOAP** El protocolo de acceso a objetos simples utiliza XML para codificar solicitudes y respuestas entre servidores anfitrión utilizando HTTP.

Pregunta al experto

P: ¿Cuándo debo utilizar un atributo y cuándo un elemento?

R: Tanto los atributos como los elementos contienen datos descriptivos, por lo que es cuestión de criterio decidir si es mejor almacenar una pieza de información en particular como elemento o como atributo. En casi todos los casos, si la información está estructurada jerárquicamente, los elementos son contenedores más apropiados; por otra parte, los atributos son mejores para información subordinada o que no tiene en sí una estructura formal.

Para encontrar un análisis formal sobre el tema, visita el artículo de IBM, developerWorks (trabajo de desarrollo) en www.ibm.com/developerswork/xml/library/x-eleatt.html, que aborda el tema con gran detalle.

Prueba esto 8-1 Crear un documento XML

Ahora que conoces las bases de XML, pongamos esos conocimientos en práctica creando un documento XML bien formado y viéndolo en un explorador Web. Este documento presentará una colección de libros utilizando XML. Cada parte del documento contendrá información sobre título, autor, género y número de páginas de cada volumen.

Para crear este documento XML abre tu editor de texto favorito y escribe el siguiente código (*biblioteca.xml*):

```
<?xml version="1.0"?>
<biblioteca>
  <libro id="1" genero="horror" rating="5">
    <titulo>The Shining</titulo>
    <autor>Stephen King</autor>
    <paginas>673</paginas>
  </libro>
  <libro id="2" genero="suspenso" rating="4">
    <titulo>Shutter Island</titulo>
    <autor>Dennis Lehane</autor>
    <paginas>390</paginas>
  </libro>
  <libro id="3" genero="fantasía" rating="5">
    <titulo>The Lord of The Rings</titulo>
    <autor>J. R .R. Tolkien</autor>
    <paginas>3489</paginas>
  </libro>
  <libro id="4" genero="suspenso" rating="3">
    <titulo>Double Cross</titulo>
    <autor>James Patterson</autor>
    <paginas>308</paginas>
  </libro>
  <libro id="5" genero="horror" rating="4">
    <titulo>Ghost Story</titulo>
    <autor>Peter Straub</autor>
    <paginas>389</paginas>
  </libro>
  <libro id="6" genero="fantasía" rating="3">
    <titulo>Glory Road</titulo>
    <autor>Robert Heinlein</autor>
    <paginas>489</paginas>
  </libro>
  <libro id="7" genero="horror" rating="3">
    <titulo>The Exorcist</titulo>
    <autor>William Blatty</autor>
    <paginas>301</paginas>
  </libro>
  <libro id="8" genero="suspenso" rating="2">
```

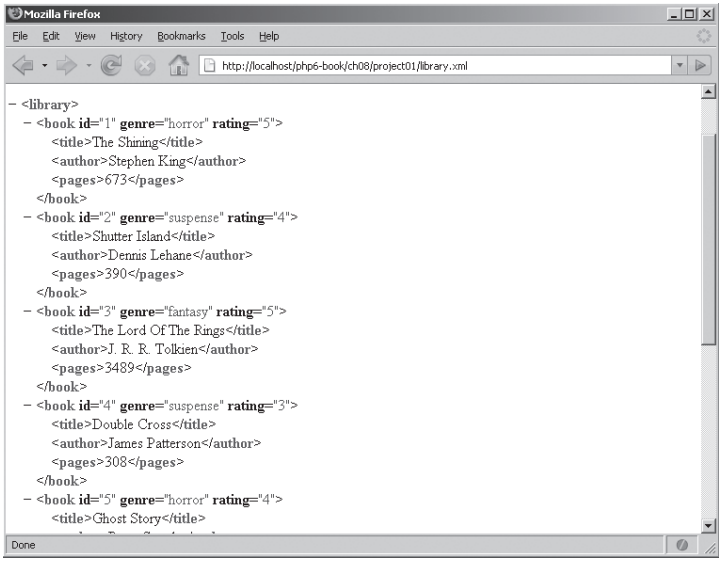


Figura 8-1 Documento XML, desplegado en Mozilla Firefox

```
    <titulo>The Camel Club</titulo>
    <autor>David Baldacci</autor>
    <paginas>403</paginas>
  </libro>
</biblioteca>
```

Guarda este archivo en la carpeta raíz para documentos de tu servidor Web y asígnale el nombre de *biblioteca.xml*. A continuación, abre tu explorador Web y escribe el URL correspondiente a la ubicación del archivo. Debes ver algo semejante a lo que aparece en la figura 8-1.

Utilizar las extensiones SimpleXML de PHP

Aunque PHP da soporte a DOM y SAX como métodos de segmentación, la manera más fácil de trabajar con XML en PHP es, por mucho, la extensión SimpleXML. Esta extensión, que está disponible por defecto en la versión 5 de PHP, proporciona una interfaz de usuario amigable e intuitiva para leer y procesar datos XML en aplicaciones PHP.

Aunque es posible guardar el texto con formato Unicode y hacer que el analizador sintáctico de todos los exploradores lo respete, aquí nos apegaremos a la práctica de no incluir acentos ni eñes, para evitar cualquier conflicto con la recuperación de datos en PHP. Si deseas usarlos, puedes recurrir a caracteres de escape para representarlos, pero su uso está más allá del alcance de este libro.

Trabajar con elementos

SimpleXML representa cada documento XML como un objeto y convierte sus elementos internos en un conjunto jerárquico de objetos y propiedades de objeto. Accesar un elemento se convierte así en cuestión de utilizar la notación `principal->secundario` para recorrer el árbol de objetos hasta alcanzar el elemento buscado.

Para mostrar cómo funciona en la práctica, examina el siguiente archivo XML (*dirección.xml*):

```
<?xml version='1.0'?>
<dirección>
  <calle>13 High Street</calle>
  <condado>Oxfordshire</condado>
  <ciudad>
    <nombre>Oxford</nombre>
    <cp>OX1 1BA</cp>
  </ciudad>
  <país>UK</país>
</dirección>
```

He aquí un script PHP que utiliza SimpleXML para leer este archivo y recuperar el nombre de la ciudad y el código postal:

```
<?php
// carga archivo XML
$xml = simplexml_load_file('dirección.xml') or die ("No fue posible
cargar XML!");

// accesa datos XML
// datos de salida: 'Ciudad: Oxford \n Código Postal: OX1 1BA\n'
echo "Ciudad: " . $xml->ciudad->nombre . "\n";
echo "Código Postal:" . $xml->ciudad->cp . "\n";
?>
```

Para leer un archivo XML con SimpleXML, utiliza la función `simplexml_load_file()` y pasa la ruta de acceso al disco del archivo como argumento. Entonces, esta función leerá e interpretará el archivo XML y, dando por hecho que está bien formado, regresará un objeto SimpleXML representando el elemento raíz del documento. Este objeto es sólo el nivel superior del árbol jerárquico, que es un espejo de la estructura interna de los datos XML: los elementos debajo del elemento raíz son representados como propiedades u objetos secundarios y así pueden accederse utilizando la notación `objetoPrincipal->objetoSecundario`.

TIP

Si tus datos XML están en una variable de cadena de caracteres y no en un archivo, utiliza la función `simplexml_load_string()` para leerla en un objeto SimpleXML.

Diferentes instancias del mismo elemento, ubicadas en el mismo nivel del documento dentro del árbol XML, son representadas como matrices. Es posible procesar estas instancias fácilmente utilizando un constructor PHP de bucles. Para aclararlo, examina el siguiente ejemplo, que lee el archivo *biblioteca.xml*, desarrollado en la sección anterior, y presenta los títulos y nombre de autores que encuentra en él:

```
<?php
// carga archivo XML
$xml = simplexml_load_file('biblioteca.xml') or die ("No fue posible
cargar XML!");
// reitera sobre los datos XML como una matriz
// presenta los títulos y autores de los libros
// datos de salida: 'The Shining fue escrito por Stephen King. \n...'
foreach ($xml->libro as $libro) {
  echo $libro->titulo . " fue escrito por " . $libro->autor . ".\n";
}
?>
```

Aquí, un bucle `foreach` itera sobre los objetos `<libro>` generados a partir de los datos XML, presentando las propiedades '`titulo`' y '`autor`' de cada uno.

También puedes contar la cantidad de elementos de un nivel en particular en el documento XML invocando la función `count()`. El siguiente código lo ejemplifica, contando la cantidad de `<libros>` en el documento XML:

```
<?php
// carga el documento XML
$xml = simplexml_load_file('biblioteca.xml') or die ("No fue posible
cargar XML!");

// recorre en bucle los datos XML como una matriz
// presenta la cantidad de libros
// datos de salida: '8 libro(s) encontrados.'
echo count($xml->libro) . ' libro(s) encontrados.';
?>
```

Trabajar con atributos

Si un elemento XML contiene atributos, SimpleXML cuenta con un medio para recuperarlos con igual facilidad: los atributos y los valores son transformados en llaves y valores de una matriz asociativa PHP y pueden accederse como elementos regulares de la matriz.

Para dejarlo en claro, analiza el siguiente ejemplo, que lee el archivo *biblioteca.xml* de la sección anterior y presenta cada título encontrado, junto con su '`genero`' y '`rating`':

```
<?php
// carga el documento XML
$xml = simplexml_load_file('biblioteca.xml') or die ("No fue posible
cargar XML!");
```

```
// accesa los datos XML
// para cada libro
// recupera y presenta los atributos 'genero' y 'rating'
// datos de salida: 'The Shining \n Género: horror \n Rating: 5 \n\n...'
foreach ($xml->libro as $libro) {
  echo $libro->título . "\n";
  echo "Género: " . $libro['genero'] . "\n";
  echo "Rating: " . $libro['rating'] . "\n\n";
}
?>
```

En este ejemplo, un bucle `foreach` itera sobre el elemento `<libro>` en los datos XML, convirtiendo cada uno de ellos en un objeto. Los atributos del elemento libro se representan como elementos de una matriz asociativa, de modo que una llave puede acceder a ellos: la llave '`genero`' regresa el valor del atributo '`genero`' y la llave '`rating`' regresa el valor del atributo '`rating`'.

Prueba esto 8-2 Convertir XML a SQL

Ahora que sabes leer elementos XML y atributos, veamos un ejemplo práctico de SimpleXML en acción. El siguiente programa lee un archivo XML y convierte los datos que contiene en una serie de declaraciones SQL, que pueden ser utilizadas para transferir los datos a MySQL o cualquier otro compilador de base de datos.

He aquí el archivo ejemplo XML (*inventario.xml*):

```
<?xml version='1.0'?>
<items>
  <item sku="123">
    <nombre>Queso cheddar</nombre>
    <precio>3.99</precio>
  </item>
  <item sku="124">
    <nombre>Queso azul</nombre>
    <precio>5.49</precio>
  </item>
  <item sku="125">
    <nombre>Tocino ahumado (paquete de 6 piezas)</nombre>
    <precio>1.99</precio>
  </item>
  <item sku="126">
    <nombre>Tocino ahumado (paquete de 12 piezas)</nombre>
    <precio>2.49</precio>
  </item>
  <item sku="127">
    <nombre>Paté de ganso</nombre>
    <precio>7.99</precio>
```

```
  </item>
  <item sku="128">
    <nombre>Paté de pato</nombre>
    <precio>6.49</precio>
  </item>
</items>
```

Y aquí está el código PHP que convierte estos datos XML en declaraciones SQL (*xmlAsql.php*):

```
<?php
// carga el documento XML
$xml = simplexml_load_file('inventario.xml') or die ("No fue posible
cargar XML!");

// recorre en bucle los elementos XML <item>
// accede a nodos secundarios e interpola con declaraciones SQL
foreach($xml as $item) {
  echo "INSERT INTO ítems (sku, nombre, precio) VALUES ('".
addslashes($item['sku']) . "','" . addslashes($item->nombre) . "','" .
addslashes($item->precio) . "');\n";
}
?>
```

Este script debe ser fácil de comprender si has seguido las lecciones: itera sobre todos los elementos `<item>` del documento XML, utilizando la notación `objeto->propiedad` para acceder a cada elemento `<nombre>` y `<precio>` de item. Se accede al atributo '`sku`' de cada `<item>` de manera similar con la llave '`sku`' de cada atributo item de la matriz. Los valores recuperados de esta manera son entonces interpolados en una declaración SQL `INSERT`.

Entonces, esta declaración puede ser proporcionada de manera normal a una función como `mysql_query()` o `sqlite_query()` para insertarlas en una base de datos MySQL o SQLite; para este ejemplo, es más sencillo desplegarlas en pantalla.

La figura 8-2 muestra los datos de salida de este script.

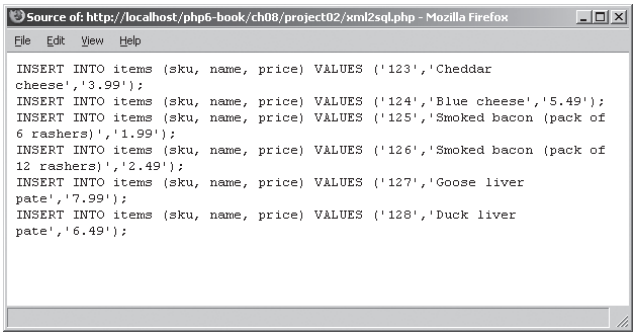


Figura 8-2 Conversión de XML a SQL con SimpleXML

Alterar elementos y valores de atributos

Con SimpleXML es fácil cambiar el contenido en un archivo XML: simplemente asigna un nuevo valor a la propiedad correspondiente del objeto utilizando el operador PHP de asignación (=). Para comprenderlo, examina el siguiente script PHP, que modifica el título y el autor del segundo libro en *biblioteca.xml* y después presenta el documento XML modificado:

```
<?php
// carga el documento XML
$xml = simplexml_load_file('biblioteca.xml') or die ("¡No fue posible
cargar XML!");

// cambia valores de elementos
// escribe un nuevo título y autor para el segundo libro
$xml->libro[1]->titulo = 'Invisible Prey';
$xml->libro[1]->autor = 'John Sandford';

// datos de salida de la nueva cadena XML
header('Content-Type: text/xml');
echo $xml->asXML();
?>
```

Aquí, se utiliza SimpleXML para acceder al segundo elemento `<libro>` por su índice, y los valores de los elementos `<titulo>` y `<autor>` son alterados suministrando nuevos valores para sus correspondientes propiedades de objeto. Pon atención al método `asXML()`, que es nuevo en este ejemplo: convierte la jerarquía anidada de objetos SimpleXML y las propiedades de los objetos en una cadena estándar de XML.

Cambiar los valores de los atributos es igual de fácil: asigna un valor nuevo a la llave correspondiente del atributo en la matriz. He aquí un ejemplo, que cambia el `'rating'` del sexto libro y presenta el resultado:

```
<?php
// carga el documento XML
$xml = simplexml_load_file('biblioteca.xml') or die ("No fue posible
cargar XML!");

// cambia los valores de atributos
// escribe un nuevo rating para el sexto libro
$xml->libro[5]{'rating'} = 5;

// muestra la nueva cadena XML
header('Content-Type: text/xml');
echo $xml->asXML();
?>
```

Añadir nuevos elementos y atributos

Además de modificar los valores de elementos y atributos, SimpleXML también te permite añadir dinámicamente nuevos elementos y atributos a un documento XML existente. Para ejemplificarlo, examina el siguiente script, que añade un nuevo `<libro>` a los datos XML de *biblioteca.xml*:

```
<?php
// carga el documento XML
$xml = simplexml_load_file('biblioteca.xml') or die ("No fue posible
cargar XML!");

// obtiene el último 'id' de los libros
$numLibros = count($xml->libro);
$lastID = $xml->libro[($numLibros-1)]{'id'};

// añade un nuevo elemento <libro>
$libro = $xml->addChild('libro');

// obtiene el atributo 'id'
// para el nuevo elemento <libro>
// incrementando 1 a $lastID
$libro->addAttribute('id', (lastID+1));

// añade atributos 'rating' y 'genero'
$libro-> addAttribute('genero', 'viajes');
$libro-> addAttribute('rating', 5);

// añade elementos <titulo>, <autor>, <página>
$título = $libro->addChild('titulo', 'Frommer\'s Italy 2007');
$autor = $libro->addChild('autor', 'Varios');
$página = $libro->addChild('paginas', 820);

// muestra la nueva cadena XML
header('Content-Type: text/xml');
echo $xml->asXML();
?>
```

Cada objeto SimpleXML presenta un método `addChild()`, para añadir nuevos elementos secundarios, y un método `addAttribute()`, para añadir nuevos atributos. Ambos métodos aceptan un nombre y un valor, generando el elemento o atributo correspondiente, y adjuntándolo al objeto principal dentro de la jerarquía XML.

Estos métodos se ilustran en el ejemplo anterior, que comienza leyendo el documento XML existente en un objeto SimpleXML. El elemento raíz de este documento se almacena en el objeto `$xml` de PHP. Entonces, el programa necesita calcular el ID que será asignado al nuevo elemento `<libro>`, y lo hace contando la cantidad de elementos que ya están presentes en el documento XML, accede al último de esos elementos, recupera su atributo '`id`' y le suma 1.

Pasada esa formalidad, el programa se dedica a la creación de elementos y atributos:

1. Comienza adjuntando un nuevo elemento `<libro>` al elemento raíz, invocando el método `addChild()` del objeto `$xml`. Este método acepta el nombre del elemento que será creado y (opcionalmente) un valor para ese elemento. El objeto XML resultante se almacena en el objeto PHP `$libro`.
2. Con el elemento creado, es tiempo de establecer sus atributos '`id`', '`genero`' y '`rating`'. Esto se hace con el método `addAttribute()` del objeto `$libro`, que también acepta dos argumentos: el nombre del atributo y su valor, y establece las correspondientes llaves de la matriz asociativa.
3. Una vez que el último elemento `<libro>` se ha definido por completo, es momento de añadir los elementos `<titulo>`, `<autor>` y `<página>` como secundarios del elemento `<libro>`. Esto se realiza fácilmente invocando de nuevo al método `addChild()`, esta vez del objeto `$libro`.
4. Una vez que los objetos secundarios están definidos, la jerarquía del objeto se transforma en una cadena del documento XML con el método `asXML()`.

La figura 8-3 muestra el resultado.

Crear nuevos documentos XML

También puedes utilizar SimpleXML para crear un nuevo documento XML de la nada, inicializando un objeto SimpleXML vacío a partir de una cadena de caracteres XML y utilizando después los métodos `addChild()` y `addAttribute()` para construir el resto del árbol XML. Examina el siguiente ejemplo, que muestra el proceso:

```
<?php
// carga XML a partir de una cadena de caracteres
$xmlStr = "<?xml version='1.0'?><persona></persona>";
$xml = simplexml_load_string($xmlStr);

// añade atributos
$xml->addAttribute('edad', '18');
$xml->addAttribute('sexo', 'masculino');

// añade elementos secundarios
$xml->addChild('nombre', 'Juan Pineda');
$xml->addChild('fdn', '04-04-1989');

// añade Segundo nivel de elementos secundarios
$direccion = $xml->addChild('direccion');
$direccion->addChild('calle', '12 A Road');
$direccion->addChild('ciudad', 'Londres');
```

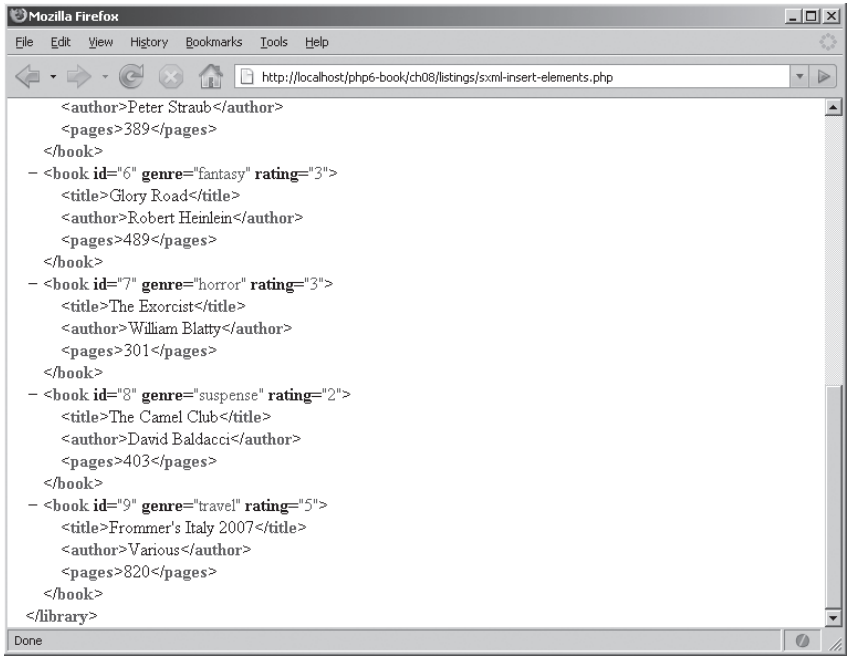


Figura 8-3 Insertar elementos en un árbol XML con SimpleXML

```
// añade un tercer nivel de elementos hijo
$pais = $direccion->addChild('pais', 'Reino Unido');
$pais->addAttribute('codigo', 'UK');

// muestra la nueva cadena XML
header('Content-Type: text/xml');
echo $xml->asXML();
?>
```

Este script PHP es similar a los que has visto en secciones anteriores, con una diferencia importante: en lugar de añadir nuevos elementos y atributos a un árbol XML ya existente, ¡este script lo genera por completo y de la nada!

El script comienza por inicializar una variable de cadena de texto para contener el prólogo y el elemento raíz del documento XML. El método `simplexml_load_string()` se encarga de convertir esta cadena en un objeto SimpleXML que representa el elemento raíz del documento. Una vez que este objeto se ha inicializado, sólo es cuestión de añadirle elementos secundarios y atributos, y construir el resto del árbol XML de manera programática. La figura 8-4 muestra el árbol XML resultante.

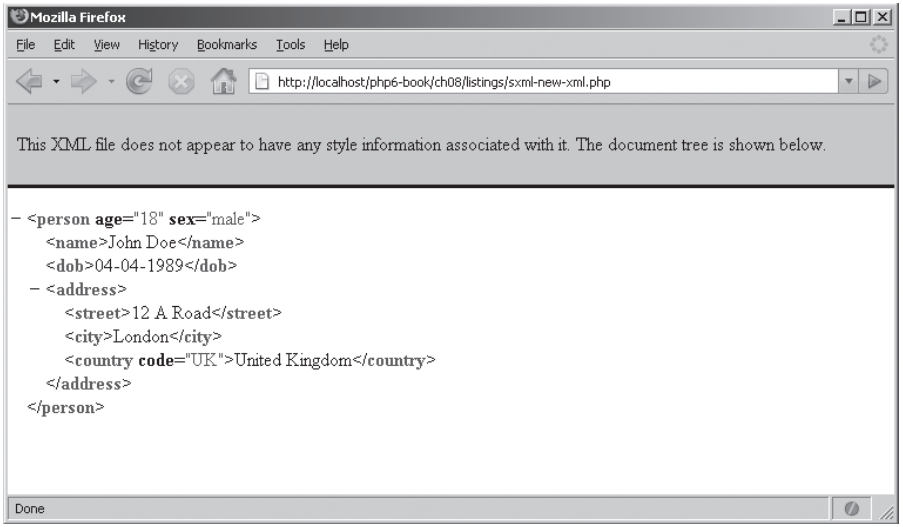


Figura 8-4 Generación dinámica de un nuevo documento XML con SimpleXML

Prueba esto 8-3 Leer informes RSS

RSS es un formato basado en XML, originalmente utilizado por Netscape para distribuir información sobre su contenido en el portal My.Netscape.com. Hoy en día, RSS es un medio muy popular en Web para distribuir información; muchos sitios Web ofrecen "informes" RSS que contienen vínculos y fragmentos de sus noticias más recientes y las actualizaciones de su contenido; casi todos los exploradores Web vienen con lectores RSS integrados, que se utilizan para leer o "suscribirse" a esos informes.

Un documento RSS sigue todas las reglas de marcado propias de XML y, por lo general, contiene una lista de recursos (URL), marcados con metadatos descriptivos. He aquí un ejemplo:

```
<?xml version='1.0' encoding="utf-8"?>
<rss>
  <channel>
    <title>El título del informe se escribe aquí</title>
    <link>El URL del informe se escribe aquí</link>
    <description>La descripción del informe ocupa este espacio</
description>
    <item>
```

```
      <title>Título de un tema particular</title>
      <description>Descripción del tema</description>
      <link>Liga al tema</link>
      <pubDate>Fecha de la publicación en formato de sello temporal</
       pubDate>
    </item>
    </item>
     ...
    <item>
  </channel>
</rss>
```

Como lo muestra este ejemplo, un documento RSS abre y cierra con un elemento `<rss>`. Un bloque `<channel>` contiene la información general sobre el sitio Web que proporciona el informe; esto es seguido por varios elementos `<item>`; cada uno de ellos representa una unidad de contenido diferente o una noticia particular. Todos los elementos `<item>` contienen su propio título, un URL y su respectiva descripción.

Dada esta estructura jerárquica bien definida, analizar sintácticamente el informe RSS con SimpleXML es de lo más sencillo. Eso es lo que hace el siguiente script: conecta un informe RSS activo a un URL anfitrión, recupera los datos del informe codificados en XML, los analiza y los convierte en una página HTML que puede desplegarse en cualquier explorador Web. He aquí el código (*rss2html.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 8-3: Lee un informe RSS</title>
    <style type="text/css">
    div.heading{
      font-weight: bolder;
    }
    div.story {
      background-color: white;
      border: 1px solid black;
      width: 320px;
      height: 200px;
      margin: 20px;
    }
    div.headline a {
      font-weight: bolder;
      color: orange;
      margin: 5px;
    }
    div.body {
      margin: 5px;
```

(*continúa*)

```
    }
    div.timestamp {
      font-size: smaller;
      font-style: italic;
      margin: 5px;
    }
    ul {
       list-style-type: none;
    }
    li {
       float: left;
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 8-3: Lee un informe RSS</h2>
<?php
// lee el informe RSS newsvine.com sobre avances tecnológicos
$xml = simplexml_load_file("http://www.newsvine.com/_feeds/rss2/
 tag?id=technology") or die ("ERROR: No es posible leer el informe RSS");
?>
    <h3 style="heading"><?php echo $xml->channel->title; ?></h3>
    <ul>
<?php
// reitera sobre la lista de noticias
// presenta el título de cada noticia, el URL y el sello cronológico
// y luego el cuerpo de la misma
foreach ($xml->channel->item as $item) {
?>
      <li>
        <div class="story">
          <div class="headline">
            <a href="<?php echo $item->link; ?>">
              <?php echo $item->title; ?>
            </a>
          </div>
          <div class="timestamp"><?php echo $item->pubDate; ?></div>
          <div class="body"><?php echo $item->description; ?></div>
        </div>
      </li>
<?php
}
?>
    </ul>
  </body>
</html>
```

Este script comienza utilizando el método `simplexml_load_file()` de SimpleXML para conectar un URL remoto (en este caso, un reporte RSS almacenado en NewsVi-

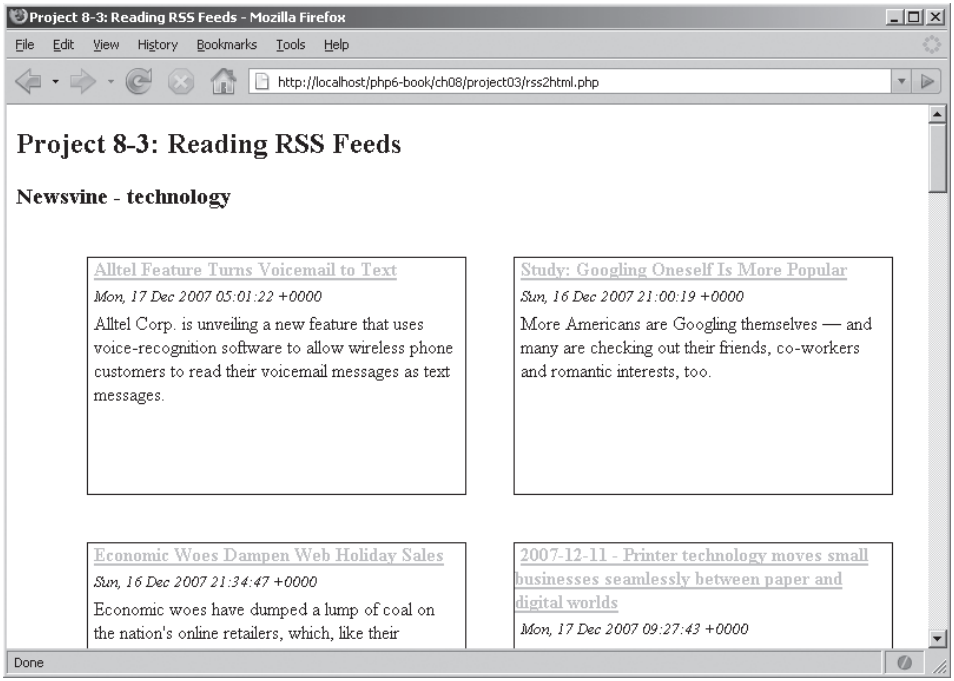


Figura 8-5 Interpretación de un informe RSS con SimpleXML

ne.com) y convertir el código XML encontrado ahí en un objeto SimpleXML. Después utiliza la capacidad de SimpleXML para hacer un bucle sobre la colección de nodos para recuperar rápidamente cada título, URL, sello cronológico y cuerpo de noticias; marca estas piezas de información con etiquetas HTML y las presenta en una página Web.

La figura 8-5 muestra los datos de salida.

Utilizar la extensión DOM de PHP

Ahora bien, la extensión de PHP SimpleXML es fácil de utilizar y comprender, pero no es útil para otra cosa que no sea el manejo básico de datos XML. Para operaciones más complejas con este lenguaje es necesario ver al siguiente paso, la extensión DOM de PHP. Esta extensión, que también está disponible por defecto desde la versión 5 de PHP, proporciona una serie de herramientas complejas que se ajusta con el estándar de nivel 3 DOM y proporciona a PHP capacidades completas para el análisis sintáctico de datos XML.

Trabajar con elementos

El analizador sintáctico DOM funciona leyendo un documento XML y creando elementos que representen las diferentes partes del mismo. Cada uno de estos objetos incluye métodos y propiedades específicas, que pueden utilizarse para acceder a su información intrínseca y manipularla. De esta manera, todo el documento XML se representa como un "árbol" de estos objetos, y el analizador sintáctico DOM proporciona una sencilla API para moverse por las diferentes ramas del árbol.

Para mostrar la manera en que funciona, revisemos el archivo *direccion.xml* de la sección anterior:

```
<?xml version='1.0'?>
<direccion>
  <calle>13 High Street</calle>
  <condado>Oxfordshire</condado>
  <ciudad>
    <nombre>Oxford</nombre>
    <cp>OX1 1BA</cp>
  </ciudad>
  <pais>UK</pais>
</direccion>
```

A continuación aparece el script PHP que utiliza la extensión DOM para analizar sintácticamente este archivo y recuperar varios componentes de la dirección:

```
<?php
// inicializa el nuevo documento
$doc = new DOMdocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('direccion.xml');

// obtiene el elemento raíz
$root = $doc->firstChild;
// obtiene el nodo 'UK'
echo "País: " . $root->childNodes->item(3)->nodeValue . "\n";

// obtiene el nodo 'Oxford'
echo "Ciudad: " . $root->childNodes->item(2)-> childNodes->item(0)
->nodeValue . "\n";
```

```
// obtiene el nodo 'OX1 1BA'
echo "Código Postal: " . $root->childNodes->item(2)-> childNodes-
>item(1)->nodeValue . "\n";

// datos de salida: 'País: UK \n Ciudad: Ofxord \n Codigo Postal: OX1
1BA'
?>
```

A primera vista se nota claramente que ya no estamos en el territorio de SimpleXML. Con la extensión DOM de PHP el primer paso es siempre inicializar una instancia del objeto DOMDocument, que representa al documento XML. Una vez que este objeto se ha inicializado, puede utilizarse para analizar sintácticamente un archivo XML con su método `load()`, que acepta la ruta de acceso en disco del archivo XML de objetivo.

El resultado del método `load()` es un árbol que contiene objetos DOMNode, donde cada objeto presenta varias propiedades y métodos para acceder a sus nodos principal y secundarios. Por ejemplo, cada objeto DOMNode presenta la propiedad `parentNode`, que se utiliza para acceder al nodo principal que corresponde al nodo en uso, lo mismo que la propiedad `childNodes`, que regresa la colección de nodos secundarios pertenecientes al nodo en uso. De manera similar, cada objeto DOMNode también presenta las propiedades `nodeName` y `nodeValue`, utilizadas para acceder, obviamente, al nombre y el valor del nodo, respectivamente. De esta manera, resulta muy fácil navegar de nodo en nodo por el árbol, recuperando valores de nodo en cada estrato.

Para ilustrar el proceso, revisa con cuidado el ejemplo anterior. Una vez que el documento XML ha sido cargado [`load()`], invoca la propiedad `firstChild` del objeto DOMDocument, que regresa un objeto DOMNode que representa el elemento raíz `<direccion>`. Este objeto DOMNode, a su vez, cuenta con la propiedad `childNodes`, que regresa una colección con todos los elementos secundarios de `<direccion>`. Se accede a los elementos individuales de esta colección a través de su ubicación en el índice utilizando el método `item()`, donde el índice comienza a partir de 0. Estos elementos son representados de nuevo como objetos DOMNode; de tal manera que sus nombres y valores son accesibles a través de las propiedades `nodeName` y `nodeValue`.

Por tanto, el elemento `<pais>`, que es el cuarto de `<direccion>`, resulta accesible a través de la ruta de acceso `$root->childNodes->item(3)`, y el valor de este elemento, `'UK'`, es accesible a través de la ruta de acceso `$root->childNodes->item(3) ->nodeValue`. De manera similar, el elemento `<nombre>`, que es el primer hijo del elemento `<ciudad>`, es accesible a través de la ruta de acceso `$root->childNodes- >item(2)->childNodes->item(0)`, y el valor `'Oxford'` es accesible a través de la ruta de acceso `$root->childNodes->item(2)->childNodes->item(0)-> nodeValue`.

```
<?xml?>                                  DOMDocument
|                                        |
|                                        |
|--<direccion>                           |-- firstChild
        |                                        |
        |-----<calle>                            |----- childNodes–> item (0)
        |                                        |
        |-----<condado>                          |----- childNodes–> item (1)
        |                                        |
        |-----<ciudad>                           |----- childNodes–> item (2)
        |         |                              |                          |
        |         |----<nombre>                  |                          |---- childNodes–> item (0)
        |         |                              |                          |
        |         |----<codigo>                  |                          |---- childNodes–> item (1)
        |                                        |
        |                                        |
        |-----<pais>                             |----- childNodes–> item (3)
```

Figura 8-6 Relaciones DOM

La figura 8-6 debe aclarar estas relaciones; presenta un mapa del árbol XML *direccion.xml* con los métodos y propiedades DOM utilizados en esta sección.

Pregunta al experto

P: Cuando proceso un documento XML utilizando DOM, se muestran lo que parecen ser nodos de texto adicionales en cada una de las colecciones de nodos. Sin embargo, cuando accedo a estos nodos parece que están vacíos. ¿Qué está sucediendo?

R: Por la especificación DOM, todos los espacios en blanco del documento, incluyendo los retornos de carro, deben tratarse como nodos de texto. Si tu documento XML contiene espacios en blanco adicionales, o si tus elementos XML están limpiamente formados e incluyen sangrías con líneas separadas, esos espacios en blanco serán representados en tu colección de nodos como nodos de texto aparentemente vacíos. En la API DOM de PHP, puedes inhibir este comportamiento al establecer la propiedad `DOMDocument->preserveWhiteSpace` como `'false'`, como en los ejemplos de esta sección.

Un método opcional (y que resulta muy útil cuando te enfrentas a un árbol XML muy anidado) consiste en utilizar el método `getElementsByTagName()`, del objeto DOMDocument, para recuperar todos los elementos con un nombre en particular. Los datos de salida que presenta este método son una colección de objetos DOMNode que coinciden con ciertos criterios; de ahí es fácil hacer reiteraciones con un bucle `foreach` y recuperar el valor de cada nodo.

Si resulta que tu documento sólo cuenta con una instancia de cada elemento (como en el caso de *direccion.xml*), utilizar el método `getElementsByTagName()` puede servir como un atajo efectivo en comparación con la navegación tradicional por el árbol XML. Examina el siguiente ejemplo, que produce los mismos datos de salida que el ejemplo anterior, sólo que en este caso se utiliza un atajo:

```
<?php
// inicializa el nuevo documento
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('direccion.xml');

// obtiene la colección de elementos <pais>
$pais = $doc->getElementsByTagName('pais');
echo "Pais: " . $pais->item(0)->nodeValue . "\n";

// obtiene la colección de elementos <nombre>
$ciudad = $doc->getElementsByTagName('ciudad');
echo "Ciudad: " . $ciudad->item(0)->nodeValue . "\n";

// obtiene la colección de elementos <cp>
$cp = $doc->getElementsByTagName('cp');
echo "Codigo Postal: " . $cp->item(0)->nodeValue . "\n";

// datos de salida: 'País: UK \n Ciudad: Oxford \n Código Postal: OXI IBA'
?>
```

En este ejemplo, el método `getElementsByTagName()` es utilizado para regresar una colección DOMNode que representa todos los elementos con el nombre `<pais>` en la primera instancia. Desde el árbol XML resulta claro que esta colección contendrá un solo objeto DOMNode. Para acceder al valor de este nodo basta con invocar al método `item()` de la colección con el argumento 0 (la primera posición en el índice) para traer el objeto DOMNode y después leer su propiedad `nodeValue`.

Sin embargo, en casi todos los casos tu documento XML no tendrá una sola instancia de cada elemento. Toma por ejemplo el archivo *biblioteca.xml* que estudiaste en secciones anteriores y que contiene varias instancias del elemento `<libro>`. Incluso en tales situaciones, el método `getElementsByTagName()` es útil para crear con eficiencia y rapidez un subgrupo de nodos que coincidan con cierto criterio, mismos que pueden procesarse con un bucle PHP. Para dejarlo en claro, examina el siguiente ejemplo, que lee el archivo *biblioteca.xml* y presenta los títulos y autores que encuentra:

```
<?php
// inicializa el nuevo documento
$doc = new DOMdocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
// el bucle foreach <libro> obtiene el valor de los elementos <titulo> y
<autor>
// datos de salida: 'The Shining fue escrito por Stephen King. \n ...'
$libros = $doc->getElementsByTagName('libro');
foreach ($libros as $libro){
  $titulo = $libro->getElementsByTagName('titulo')->item(0)->nodeValue;
  $autor = $libro->getElementsByTagName('autor')->item(0)->nodeValue;
echo "$titulo fue escrito por $autor. \n";
}
?>
```

En este caso, la primera invocación a `getElementsByTagName()` regresa una colección que representa todos los elementos `<libro>` del documento XML. Después es fácil hacer reiteraciones sobre esta colección con el bucle `foreach()`, procesando así cada objeto DOMNode y recuperando el valor correspondiente a los elementos `<titulo>` y `<autor>` con posteriores invocaciones a `getElementsByTagName()`.

TIP

Puedes recuperar una colección con todos los elementos de un documento con la invocación `DOMDocument -> getElementsByTagName(*)`.

Para saber cuántos elementos fueron regresados por `getElementsByTagName()`, utiliza la propiedad `length` de la colección resultante. He aquí un ejemplo:

```
<?php
// inicializa el nuevo documento
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;
// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
// regresa un conteo del total de elementos <libro>
// datos de salida: '8 libro(s) encontrados.'
```

```
$libros = $doc->getElementsByTagName('libro');
echo $libros->length . ' libro(s) encontrados.';
?>
```

Trabajar con atributos

DOM también incluye amplio soporte a los atributos: cada objeto DOMElement viene con un método `getAttribute()`, que acepta un nombre de atributo y regresa el valor respectivo. He aquí un ejemplo, que presenta cada `rating` y `genero` de libro del documento *biblioteca.xml*:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
// por cada libro
// recupera y presenta los atributos 'genero' y 'rating'
// datos de salida: 'The Shining \n Genero: horror \n Rating: 5 \n\n ...'
$libros = $doc->getElementsByTagName('libro');
foreach ($libros as $libro) {
  $titulo = $libro->getElementsByTagName('titulo')->item(0)->nodeValue;
  $rating = $libro->getAttribute('rating');
  $genero = $libro->getAttribute('genero');
  echo "$titulo\n";
  echo "Género: $genero\n";
  echo "Rating: $rating\n\n";
}
?>
```

¿Qué sucede si no conoces el nombre del atributo y simplemente quieres procesar todos los atributos de un elemento? Bueno, cada DOMElement tiene una propiedad `attributes`, que regresa una colección con todos los atributos del elemento. Es fácil hacer reiteraciones sobre esta colección para recuperar cada `name` y `value` del atributo.

El siguiente ejemplo muestra el uso de esta propiedad, con una variación sobre el código anterior:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument();
```

```
// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
// por cada libro
// recupera y presenta todos los atributos
// datos de salida: 'The Shining \n id: 1 \n genero: horror \n rating: 5
\n\n ...'
$libros = $doc->getElementsByTagName('libro');
foreach ($libros as $libro) {
  $titulo = $libro->getElementsByTagName('titulo')->item(0)->nodeValue;
  echo "$titulo\n";
  foreach ($libro->attributes as $attr) {
    echo "$attr->name: $attr->value \n";
  }
  echo "\n";
}
?>
```

Prueba esto 8-4 Procesar recursivamente un documento árbol de XML

Si planeas trabajar con XML y PHP en el futuro, con toda seguridad el siguiente proyecto te será de utilidad algún día: es un programa sencillo que comienza en la raíz del documento árbol XML y recorre todas sus ramas, procesando cada elemento y atributo que encuentra en el camino. Dada la naturaleza de tipo árbol del documento XML, la manera más eficiente de realizar esta tarea es con una función recursiva, y dada la riqueza informativa proporcionada por DOM, escribir la función es una tarea sencilla.

Supongamos por un momento que el documento XML que habrá de procesarse tiene el siguiente (*inventario.xml*):

```
<?xml version='1.0'?>
<objetos>
  <objeto color="rojo" forma="cuadrado">
  <longitud unidades="cm">5</longitud>
</objeto>
<objeto color="rojo" forma="círculo">
  <radio unidades="px">7</radio>
</objeto>
<objeto color="verde" forma="triángulo">
  <base unidades="in">1</base>
```

```
  <altura unidades="in">2</altura>
  </objeto>
  <objeto color="azul" forma="triángulo">
    <base unidades="mm">100</base>
    <altura unidades="mm">50</altura>
  </objeto>
  <objeto color="amarillo" forma="círculo">
    <radio unidades="cm">18</radio>
  </objeto>
</objetos>
```

Y aquí está el codigo PHP para procesar recursivamente este (o cualquier otro) documento XML utilizando DOM:

```
<!DOCTYPE html PUBLIC "-//W3C// DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 8-4: Procesa recursivamente un documento XML</title>
  </head>
  <body>
    <h2>Proyecto 8-4: Procesa recursivamente un documento XML</h2>
    <pre>
<?php
// función recursiva para procesar una colección de nodos XML
function xmlProcess($node, $depthMarker) {

  // procesa los hijos de este nodo
  foreach ($node->childNodes as $n) {
    switch ($n->nodeType) {

      // para los elementos, presenta el nombre del elemento
      case XML_ELEMENT_NODE:
        echo "$depthMarker <b>$n->nodeName</b> \n";
        // si el elemento tiene atributos
        // lista sus nombres y valores
        if ($n->attributes->length > 0) {
          foreach ($n->attributes as $attr) {
            echo "$depthMarker <i>attr</i>: $attr->name => $attr->
value \n";
          }
      }
      break;
      // para datos de texto, presenta valores
      case XML_TEXT_NODE:
        echo "depthMarker <i>text</i>: \"$n->nodeValue\" \n";
```

(*continúa*)

```
        break;
    }

    // si este nodo tiene un nivel inferior o subnodos
    // incrementa el marcador de profundidad
    // lo ejecuta recursivamente
    if ($n->hasChildNodes()) {
      xmlProcess($n, $depthMarker . DEPTH_CHAR);
    }
  }
}
// finaliza definición de función

// define el carácter utilizado para la indentación
define ('DEPTH_CHAR', ' ');

// inicializa DOMDocument
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('objetos.xml');

// invoca la función recursiva con el elemento raíz
xmlProcess($doc->firstChild, DEPTH_CHAR);
?>
    </pre>
  </body>
</html>
```

En este programa, la función personalizada `xmlProcess()` es una función recursiva que acepta un objeto DOMNode, recupera la colección de elementos secundarios de este nodo al leer la propiedad `childNodes` del objeto, e itera sobre esta colección con un bucle `foreach`. Si el nodo en uso es un nodo elemento, presenta el nombre del nodo y si es un nodo de texto, presenta su valor. En caso de que se trate de un nodo de elemento, el programa realiza un paso adicional para verificar los atributos y presentarlos, si es necesario. Utiliza una instrucción "cadena inferior" para indicar la posición jerárquica del nodo en los datos de salida; esta cadena se incrementa automáticamente cada vez que se ejecuta el bucle.

Cuando finaliza todas esas tareas, la última acción de la función es verificar si el nodo en uso tiene elementos secuandarios; en caso positivo, se invoca recursivamente para procesar el siguiente nivel del árbol de nodos. El proceso continúa hasta que se agotan los nodos que habrán de procesarse.

La figura 8-7 presenta los datos de salida del programa cuando se invoca `xmlProcess()` con el elemento raíz del documento como argumento de inicio.

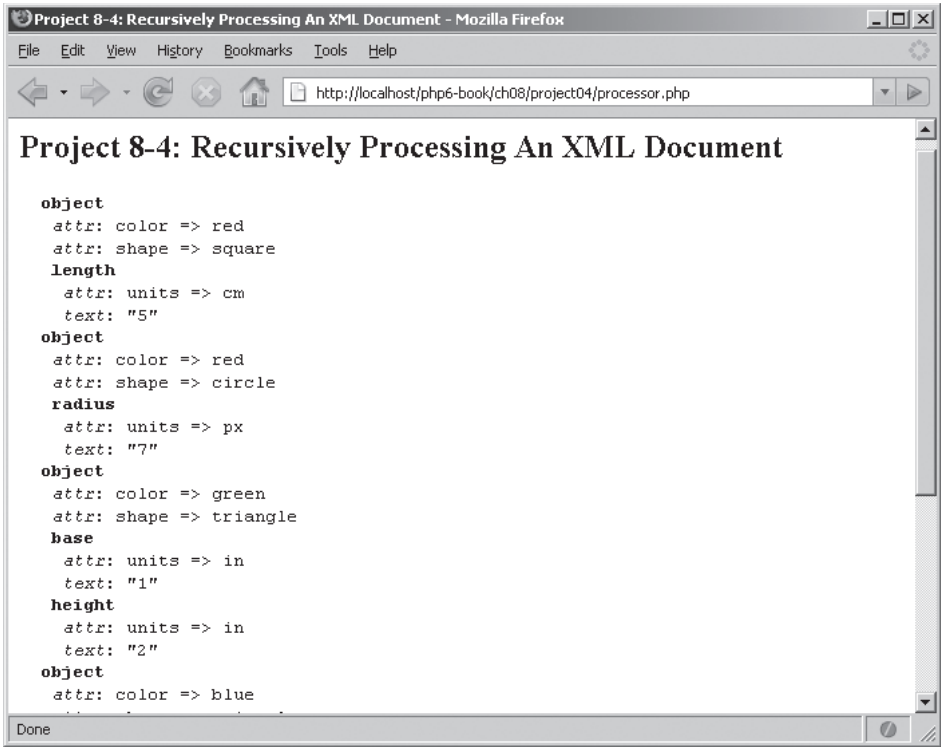


Figura 8-7 Procesamiento recursivo de un documento XML con DOM

Alterar elementos y valores de atributos

Con DOM, cambiar el valor de un elemento XML es muy sencillo: navega hasta el objeto DOMNode que representa el elemento y altera su propiedad nodeValue para insertar el nuevo valor. Como ejemplo, considera el siguiente script PHP, que cambia el título y el autor del segundo libro en *biblioteca.xml* y luego presenta el documento modificado:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');
```

```
// obtiene la colección de elementos <libro>
$libros = $doc->getElementsByTagName('libro');

// cambia el elemento <titulo> del segundo <libro>
$libros->item(1)->getElementsByTagName('titulo')->item(0)->nodeValue =
'Invisible Prey';

// cambia el elemento <autor> del segundo <libro>
$libros->item(1)->getElementsByTagName('autor')->item(0)->nodeValue =
'John Sandford';

// presenta la nueva cadena XML
header('Content-Type: text/xml');
echo $doc->saveXML();
?>
```

Aquí, el método `getElementsByTagName()` es utilizado primero para obtener la colección de elementos `<libro>` y para navegar hacia el segundo elemento de esa colección (cuyo lugar en el índice es 1). Después se vuelve a utilizar para obtener referencias para el objeto DOMNode que representa los elementos `<titulo>` y `<autor>`. A las propiedades `nodeValue` de esos objetos se les asigna entonces un nuevo valor utilizando el operador de asignación de PHP, y el árbol XML modificado es transformado de nueva cuenta a una cadena con el método `saveXML()` del objeto DOMDocument.

Cambiar valores de atributo es igual de fácil: asigna un nuevo valor a un atributo utilizando el método `setAttribute()` del objeto DOMDocument. He aquí un ejemplo, que cambia el '`genero`' del quinto libro y presenta el resultado:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument();

// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
$libros = $doc->getElementsByTagName('libro');

// cambia el elemento 'genero' del quinto <libro>
$libros->item(4)->setAttribute('genero', 'horror-suspenso');

// presenta la nueva cadena XML
header('Content-Type: text/xml');
echo $doc->saveXML();
?>
```

Crear nuevos documentos XML

DOM contiene un API completamente marcado para crear nuevos documentos XML o para señalar elementos, atributos y otras estructuras similares ya existentes en un árbol XML. Esta API, que es mucho más compleja que la ofrecida por SimpleXML, debe ser tu primera opción cuando crees o modifiques dinámicamente un árbol XML con PHP.

La mejor manera de ejemplificar esta API es con un ejemplo. Examina el siguiente listado, que crea un archivo XML desde cero:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument('1.0');

// crea y adjunta el elemento raíz <programa>
$root = $doc->createElement('programa');
$programa = $doc->appendChild($root);

// crea y adjunta el elemento <curso> bajo <programa>
$curso = $doc->createElement('curso');
$programa->appendChild($curso);

// crea y adjunta el elemento <materia> bajo <curso>
// y añade un valor para el elemento <materia>
$materia = $doc->createElement('materia');
$materiaData = $doc->createTextNode('Macroeconomía');
$curso->appendChild($materia);
$materia->appendChild($materiaData);

// crea y adjunta el elemento <maestro> bajo <curso>
// y añade un valor para el elemento <maestro>
$maestro = $doc->createElement('maestro');
$maestroData = $doc->createTextNode('Profesor Q. Draw');
$curso->appendChild($maestro);
$maestro->appendChild($maestroData);

// crea y adjunta el elemento <creditos> bajo <curso>
// y añade un valor para el elemento <creditos>
$creditos = $doc->createElement('creditos');
$creditosData = $doc->createTextNode('4');
$curso->appendChild($creditos);
$creditos->appendChild($creditosData);

// adjunta un atributo 'transferible' al elemento <creditos>
// establece un valor para el atributo
$transferible = $doc->createAttribute('transferible');
```

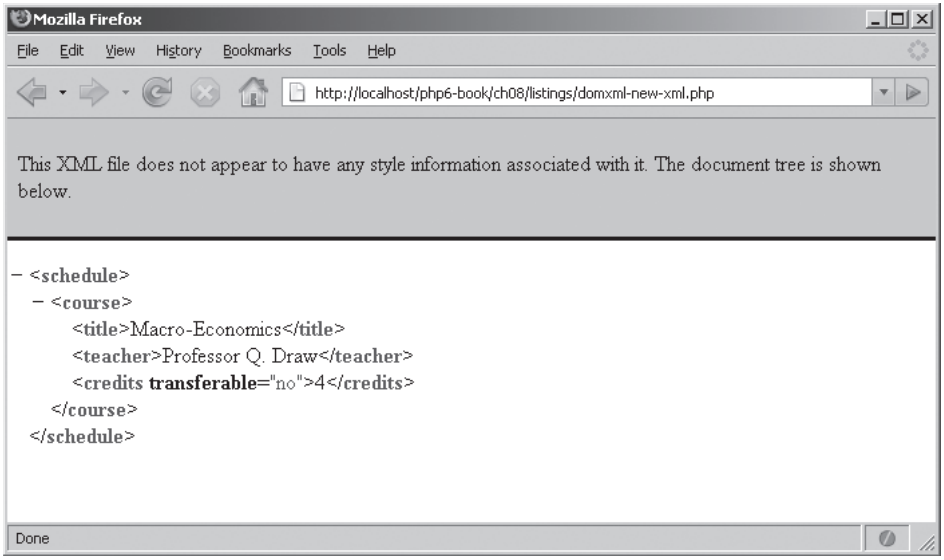


Figura 8-8 Generación dinámica de un nuevo documento XML con DOM

```
$creditos->appendChild($transferible);
$creditos->setAttribute('transferible', 'no');

// forma los datos XML de salida
$doc->formatOutput = true;

// presenta el documento XML
header('Content-Type: text/xml');
echo $doc->saveXML();
?>
```

La figura 8-8 muestra el documento XML generado por el script.

Este script presenta nuevos métodos, todos ellos relacionados con la creación dinámica de nodos y la manera de adjuntarlos al árbol XML. El proceso presentado requiere dos pasos básicos:

1. Crear un objeto que representa la estructura XML que quieres añadir. La base del objeto DOMDocument presenta los métodos `create...()` correspondientes a cada una de las estructuras primarias XML: `createElement()` para elementos, `createAttribute()` para atributos y `createTextNode()` para datos de texto.

2. Adjuntar un nuevo objeto en el punto apropiado dentro del árbol, invocando el método del padre `appendChild()`.

El ejemplo anterior presenta estos pasos, siguiendo secuencias específicas para conseguir el árbol resultante que se muestra en la figura 8-8.

1. Comienza inicializando un objeto DOMDocument llamado `$doc`, para luego invocar su método `createElement()` y generar un nuevo elemento DOMElement llamado `$programa`. Este objeto representa el elemento raíz del documento; como tal, se adjunta a la base del árbol DOM invocando el método `$doc->appendChild()`.
2. Un nivel debajo del elemento `<programa>` viene el elemento `<curso>`. En términos DOM, esto se realiza creando un nuevo objeto DOMElement llamado `$curso` con el método `createElement()` del objeto DOMDocument, y luego adjuntando este objeto al árbol debajo del elemento `<programa>` al invocar `$programa->appendChild()`.
3. Un nivel abajo del elemento `<curso>` viene el elemento `<materia>`. De nuevo, esto se lleva a cabo creando un objeto DOMElement llamado `$materia` y adjuntándolo bajo `<curso>` con la invocación `$curso->appendChild()`. Sin embargo, aquí hay una variación: el elemento `<materia>` contiene un valor de texto llamado '`Macroeconomía`'. Para crear este valor de texto, el script crea un nuevo objeto DomTextNode a través del objeto `createTextNode()`, lo rellena con la cadena de texto y luego lo adjunta como hijo del elemento `<materia>` con la invocación `$materia->appendChild()`.
4. Lo mismo sucede un poco más adelante, cuando se crea el elemento `<creditos>`. Una vez que el elemento y su valor de texto han sido definidos y adjuntados al árbol debajo del elemento `<curso>`, se utiliza el método `createAttribute()` para crear un nuevo objeto DOMAttr para representar el atributo '`transferible`'. Luego, este atributo es adjuntado al elemento `<creditos>` con la invocación `$creditos->appendChild()`, y se le asigna un valor al atributo de manera normal, invocando `$creditos->setAttribute()`.

Conversión entre DOM y SimpleXML

Una característica interesante de PHP es su capacidad para convertir datos XML entre DOM y SimpleXML. Esto se realiza con dos funciones: `simplexml_import_dom()`, que acepta un objeto DOMElement y regresa un objeto SimpleXML, y la función `dom_import_simplexml()`, que hace lo inverso. El siguiente ejemplo muestra esta interpolaridad:

```
<?php
// inicializa un nuevo DOMDocument
$doc = new DOMDocument();
```

```
// desecha los nodos que contienen sólo espacios en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene la colección de elementos <libro>
$libros = $doc->getElementsByTagName('libro');

// convierte el sexto <libro> en objeto SimpleXML
// presenta el titulo del sexto libro
// datos de salida: 'Glory Road'
$sxml = simplexml_import_dom($libros->item(5));
echo $sxml->titulo;
?>
```

Prueba esto 8-5 Leer y escribir archivos de configuración XML

Ahora que ya sabes leer y crear documentos XML de manera programática, utilicemos este conocimiento en una aplicación que se está haciendo muy popular en estos días: archivos de configuración basados en XML, que utiliza este lenguaje para marcar los datos de configuración de una aplicación.

El siguiente ejemplo lo pone en acción: genera un formulario Web que permite a los usuarios configurar un horno en línea, ingresando datos de configuración sobre temperatura, modo y fuente de calor. Cuando el formulario se envía, los datos proporcionados por el usuario son convertidos en XML y guardados en un archivo de disco. Cuando los usuarios vuelven a ver el formulario, los datos guardados con anterioridad se utilizan para llenar los campos del formulario.

He aquí el código (*configura.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
   <title>Proyecto 8-5: Leer y escribir archivos de configuración XML</
title>
  </head>
  <body>
    <h2>Proyecto 8-5: Leer y escribir archivos de configuración XML</h2>
    <h3 style="background-color: silver">Configuración de un horno</h3>
<?php
```

```
    // define el nombre y la ruta de acceso del archivo de configuración
    $configFile = 'config.xml';

    // si el formulario no ha sido enviado
    // despliega el formulario
    if (!isset($_POST['submit'])) {

      // establece matriz con parámetros por defecto
      $datos = array();
      $datos['modo'] = null;
      $datos['temperatura'] = null;
      $datos['duración'] = null;
      $datos['direccion'] = null;
      $datos['autoapagado'] = null;

      // lee los valores de configuración en uso
      // utiliza los valores para rellenar el formulario
      if (file_exists($configFile)) {
        $doc = new DOMDocument();
        $doc->preserveWhiteSpaces = false;
        $doc->load($configFile);
        $horno = $doc->getElementsByTagName('horno');
        foreach ($horno->item(0)->childNodes as $nodo) {
          $datos[$nodo->nodeName] = $nodo->nodeValue;
        }
      }
?>
    <form method="post" action="configura.php">
      Modo: <br />
      <select name="data[modo]">
        <option value="asado" <?php echo ($datos['modo'] == 'asado') ?
'selected' : null; ?>>Asado</option>
        <option value="cocido" <?php echo ($datos['modo'] == 'cocido') ?
'selected' : null; ?>>Cocido</option>
        <option value="tostado" <?php echo ($datos['modo'] == 'tostado')
? 'selected' : null; ?>>Tostado</option>
      </select>

      <p>
      Temperatura: <br />
      <input type="text" size="2" name="data[temperatura]" value="<?php
echo $datos['temperatura']; ?>"/>

      <p>
      Duración (minutos): <br />
      <input type="text” size="2" name="data[duración]" value="<?php echo
$datos['duración']; ?>"/>
```

(*continúa*)

```
      <p>

      Fuente del calor y direccion: <br />
      <input type="radio" name="data[direccion]" value="arriba-abajo"
<?php echo ($datos['direccion'] == 'arriba-abajo') ? 'checked' : null;
?>>De arriba hacia abajo</input>
      <input type="radio" name="data[direccion]" value="abajo-arriba"
<?php echo ($datos['direccion'] == 'abajo-arriba') ? 'checked' : null;
?>>De abajo hacia arriba</input>
      <input type="radio" name="data[direccion]" value="ambos" <?php echo
($datos['direccion'] == 'ambos') ? 'checked' : null; ?>>Ambos</input>

      <p>

      Apagar automáticamente cuando termine:
      <input type="checkbox" name="data[autoapagado]" value="yes" <?php
echo ($datos['autoapagado'] == 'yes') ? 'checked' : null; ?>/>

      <p>

      <input type="submit" name="submit" value="Enviar" />
      </form>
<?php
    // si el formulario ha sido enviado
    // procesa los datos de entrada
    } else {
      // lee los datos enviados
      $config = $_POST['data'];

      // valida los datos enviados como sea necesario

      if ((trim($config['temperatura']) == '') || (trim($config
['temperatura']) != '' && (int)$config['temperatura'] <= 0)) {
        die ('ERROR: Por favor ingrese una temperatura de horno válida');
      }

      if ((trim($config['duración']) == '') || (trim($config['duración'])
!= '' && (int)$config['duración'] <= 0)) {
        die ('ERROR: Por favor ingrese una duración válida');
      }

      // genera un nuevo documento XML
      $doc = new DOMDocument();

      // crea y adjunta el elemento raíz <configuración>
      $root = $doc->createElement('configuración');
      $configuración = $doc->appendChild($root);
      // crea y adjunta elemento <horno> bajo <configuración>
      $horno = $doc->createElement('horno');
      $configuración->appendChild($horno);
```

```
      // escribe cada valor de configuración en el archivo
      foreach ($config as $key => $value) {
        if (trim ($value) != '') {
          $elem = $doc->createElement($key);
          $texto = $doc->createTextNode($value);
          $horno->appendChild($elem);
          $elem->appendChild($texto);
        }
      }

      // forma datos de salida XML
      // guarda el archivo XML
      $doc->formatOutput = true;
      $doc->save($configFile) or die ('ERROR: No fue posible escribir el
archivo de configuración');
      echo 'Los datos de configuración se escribieron correctamente en el
archivo.';
    }
?>
  </body>
</html>
```

La figura 8-9 muestra el formulario Web generado por el script.

Una vez que el formulario se ha enviado, los datos ingresados llegan en forma de una matriz asociativa, cuyas claves corresponden a los nombres de los elementos XML. Primero, estos datos son validados, y después la API DOM es utilizada para generar un nuevo árbol XML que contiene esos elementos y sus valores. Una vez que el árbol se genera completamente, la función `save()` del objeto DOMDocument se utiliza para escribir el archivo XML en disco.

He aquí un ejemplo de la apariencia que tendría el documento *config.xml* después de haber enviado el formulario de la figura 8-9:

```
<?xml version="1.0"?>
<configuración>
  <horno>
    <modo>tostado</modo>
    <temperatura>22</temperatura>
    <duración>1</duración>
    <direccion>De abajo hacia arriba</direccion>
    <autoapagado>yes</autoapagado>
  </horno>
</configuración>
```

Si un usuario vuelve a visitar el formulario Web, el script primero verifica la existencia de un archivo de configuración llamado *config.xml*; en caso positivo, se leen los datos XML del

(*continúa*)

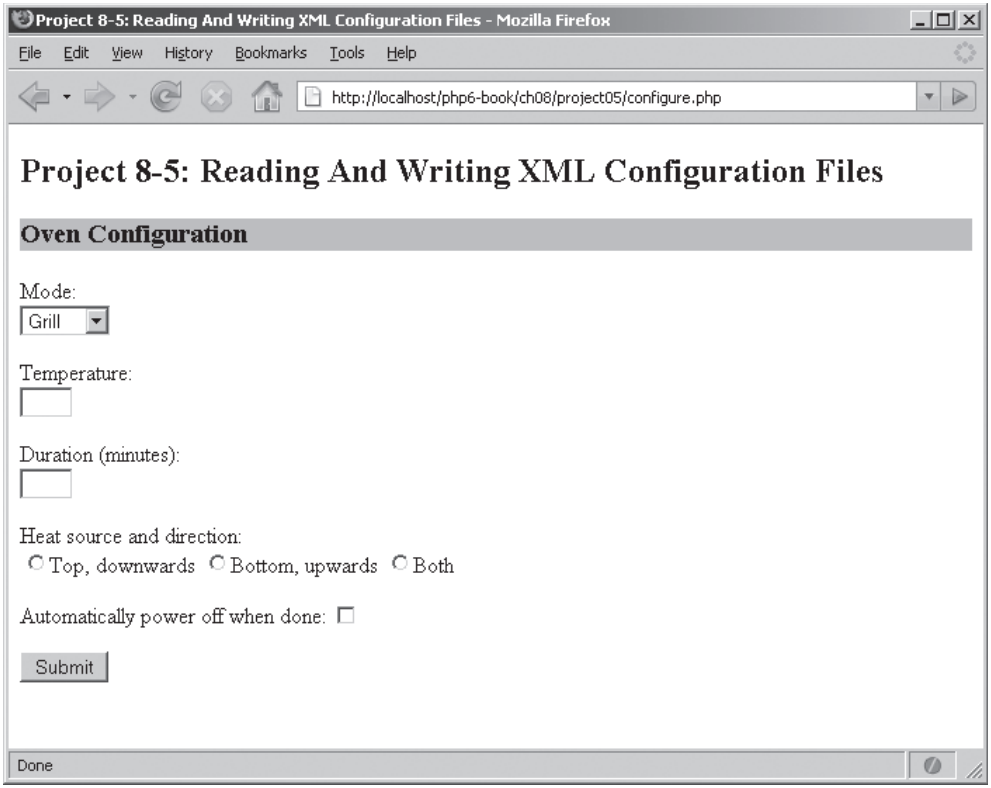


Figura 8-9 Formulario Web para datos de configuración

archivo en un nuevo objeto DOMDocument con el método `load()` y se convierten en una matriz asociativa para iterar sobre la lista de nodos secundarios con un bucle. Los diferentes botones de opción, casillas de verificación y listas de selección en el formulario ya están activados o preseleccionados, dependiendo de los valores de la matriz.

La figura 8-10 muestra el formulario precompletado con los datos leídos del archivo de configuración XML.

Si el usuario envía el formulario Web con nuevos datos, éstos serán codificados de nuevo en formato XML y utilizados para reescribir el archivo de configuración. Ya que la configuración está expresada en XML, todo explorador Web que tenga capacidad para interpretar este lenguaje puede leer y utilizar los datos. Cuando se utiliza de esta manera, XML proporciona un medio para transferir información entre aplicaciones, aunque éstas sean escritas en diferentes lenguajes de programación o se ejecuten en sistemas operativos incompatibles.

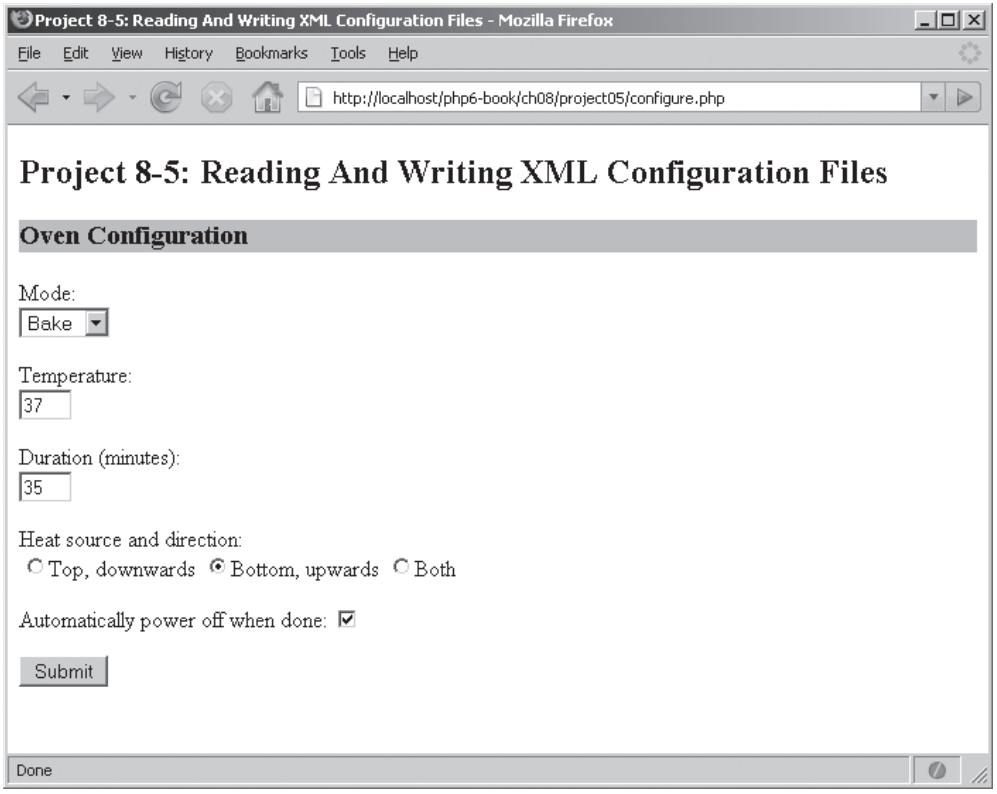


Figura 8-10 El mismo formulario Web, precompletado con los datos de configuración

Resumen

Al finalizar este capítulo ya debes saber lo suficiente para comenzar a escribir programas PHP que puedan interactuar correctamente con datos XML. Este capítulo comenzó con una introducción a XML, explicando sus estructuras básicas como elementos, atributos y datos de carácter, además de proporcionarte un curso relámpago sobre las tecnologías XML y los métodos de análisis sintáctico. Después, siguió una explicación sobre las dos extensiones PHP más populares para procesar datos XML: SimpleXML y DOM, y se explicó la manera en que cada una de ellas puede utilizarse para accesar valores de elemento y atributo, crear colecciones de nodos, además de generar o modificar de manera programática árboles de documento XML. Se utilizaron varios proyectos, desde convertir de XML a SQL hasta la lectura de documentos RSS, para ilustrar de modo práctico la interacción entre XML y PHP.

XML es un tema muy extenso y el material en este libro apenas delinea un esbozo de la superficie. Sin embargo, hay muchos tutoriales y artículos excelentes sobre XML y PHP en Web; aquí se presentan algunos vínculos hacia ellos; consúltalos si quieres saber más sobre este tema tan interesante y en continuo cambio:

- Bases de XML, en **www.melonfire.com/community/columns/trog/article.php?id=78**
- Bases de XPath, en **www.melonfire.com/community/columns/trog/article.php?id=83**
- Bases de XSL, en **www.melonfire.com/community/columns/trog/article.php?id=82**
- Funciones SimpleXML, en **www.php.net/simplexml**
- Funciones API DOM en PHP, en **www.php.net/dom**
- La especificación DOM, en **www.w3.org/DOM/**
- Construcción de documentos XML utilizando PHP y PEAR, en **www.melonfire.com/community/columns/trog/article.php?id=180**
- Serialización de XML, en **www.melonfire.com/community/columns/trog/article.php?id=244**
- Realizar Procedimiento de Invocación Remota (RCP) con PHP, en **www.melonfire.com/community/columns/trog/article.php?id=274**

✓ Autoexamen Capítulo 8

1. ¿Cuáles son los dos métodos para analizar sintácticamente un documento XML y en qué se diferencian?
2. Nombra dos características de un documento XML bien formado.
3. Dado el siguiente documento XML (*email.xml*), escribe un programa para recuperar y presentar todas las direcciones de correo electrónico del documento utilizando SimpleXML:

```
<?xml version='1.0'?>
<data>
  <persona>
    <nombre>Clon Uno</nombre>
    <email>uno@domain.com</email>
  </persona>
  <persona>
    <nombre>Clon Sesenta y cuatro</nombre>
    <email>sesentaycuatro@domain.com</email>
  </persona>
```

```
  <persona>
    <nombre>Clon Tres</nombre>
    <email>tres@domain.com</email>
  </persona>
  <persona>
    <nombre>Clon Noventa y Nueve </nombre>
    <email>noventaynueve@domain.com</email>
  </persona>
</data>
```

4. Dado el siguiente documento XML (*arbol.xml*), sugiere tres maneras diferentes para recuperar el texto con valor `'John'` utilizando DOM:

```
<?xml version='1.0'?>
<arbol>
  <persona type="abuelo" />
  <persona type="abuela" />
  <hijos>
    <persona type="papá" />
    <persona type="mamá" />
    <hijos>
      <persona type="hermano">
        <nombre>John</nombre>
      </persona>
    <persona type="hermana">
      <nombre>Jane</nombre>
    </persona>
    </hijos>
  </hijos>
</arbol>
```

5. Escribe un programa que cuente el número de elementos en un archivo XML. Utiliza DOM.

6. Escribe un programa que procese el archivo *biblioteca.xml* que aparece al principio de este capítulo, para incrementar en 1 el rating de cada libro; presenta el resultado de la modificación. Utiliza SimpleXML.

7. Escribe un programa que se conecte a una base de datos MySQL y recupere el contenido de cualquier tabla como un archivo XML. Utiliza DOM.

Capítulo 9

Trabajar con cookies, sesiones y encabezados

Habilidades y conceptos clave

- Comprender cómo funcionan las cookies
- Escribir y utilizar tus propias cookies para crear páginas "desprendibles"
- Compartir datos entre páginas con sesiones y variables de sesión
- Manipular el explorador del usuario enviándole encabezados HTTP personalizados

Tal vez ya sepas que el protocolo de transferencia de hipertexto (http, Hypertext Transfer Protocol) es el protocolo estándar para transferir datos entre tu explorador y los diferentes sitios Web que visitas. Sin embargo, lo que tal vez no sepas es que HTTP es un protocolo "sin estado", que trata cada solicitud de página Web como una transacción única e independiente, sin ninguna relación con la transacción antecedente. Para solucionar este inconveniente, casi todas las estaciones Web utilizan cookies o sesiones para "preservar el estado", y poder ofrecer mejores servicios a los usuarios, como las transacciones comerciales en línea y la restauración automática de configuración personal de la página.

PHP incluye soporte completo a cookies y sesiones. Al utilizar este soporte es fácil crear ambas, almacenar y recuperar los datos específicos del usuario que se encuentran en ellas, manipularlas desde tu aplicación PHP e incluso enviar encabezados personalizados al explorador del usuario para alterar su comportamiento estándar. Este capítulo te enseñará cómo hacerlo, con algunos ejemplos prácticos que muestran lo útil que pueden ser estas características cuando construyes sitios y aplicaciones Web.

Trabajar con cookies

Las cookies no son difíciles de comprender, pero hay algunos aspectos básicos con los que necesitas familiarizarte antes de que comiences a escribir el código que las maneje. La siguiente sección te presentará una introducción a estos aspectos básicos.

Aspectos básicos de las cookies

En su forma más sencilla, una *cookie* es un archivo de texto guardado en el equipo del usuario por un sitio Web. El archivo contiene información que el sitio puede recuperar durante la siguiente visita del usuario, con lo que le permite "reconocerlo" para proporcionarle un conjunto enriquecido de características personalizadas para ese usuario específico. Ejemplos comunes de estas características son: mostrar el contenido del sitio de modo personalizado, de acuerdo con

las preferencias del usuario; mantener un registro del contenido visitado, además de integrar información personal del usuario en la disposición de los elementos mostrados en el sitio.

Como las cookies facilitan la transferencia de datos entre el usuario y el sitio Web remoto, se les ha vilipendiado en los medios de información masiva tildándolas de "inseguras" y "malas". La verdad es que se trata de una exageración: las cookies (como otras tecnologías) pueden ser mal utilizadas; la mayor parte de los sitios Web las ocupan sin causar daño y pueden vincularse directamente para mejorar la experiencia del usuario al navegar por Web. Además, las cookies tienen importantes características de seguridad, como las que se muestran a continuación:

- Una cookie sólo puede ser leída por el sitio Web o el dominio que la creó.
- Un solo dominio no puede colocar más de 20 cookies.
- Una sola cookie no puede exceder de 4 kilobytes de tamaño.
- El número máximo de cookies que se pueden establecer en el equipo de un usuario es de 300.

PRECAUCIÓN

Como las cookies se almacenan en el disco duro del usuario, los desarrolladores tienen poco control sobre ellas. Si el usuario decide "apagar" el soporte a cookies en su explorador, tus cookies simplemente no almacenarán. Por ello, si la persistencia de los datos es una característica importante en tu sitio Web, debes tener listo un plan de respaldo (como cookies del lado del servidor o trabajar con sesiones).

Atributos de las cookies

Una cookie típica de un sitio Web contiene menos ingredientes que una galleta cocinada; para ser preciso, son cinco. La tabla 9-1 presenta la lista.

1. Cada cookie contiene una pareja nombre-valor, que representa el nombre de la variable y su correspondiente valor que debe ser almacenado en la cookie.

Atributo	Descripción	Ejemplo
`name=`*`valor`*	El nombre y valor de la cookie	`'email=yo@algún.dominio.com'`
`expires`	La validación de la cookie	`'expires=Viernes, 25-Ene-08 23:59:50 IST'`
`domain`	El dominio asociado con la cookie	`'domain=estesitioweb.com'`
`path`	La ruta de acceso del dominio asociado con la cookie	`'path=/'`
`secure`	Si está presente, se requiere una conexión segura HTTP para leer la cookie	`'secure'`

Tabla 9-1 Atributos de una cookie

2. El atributo `'expires'` de una cookie define su tiempo válido de duración. Si se establece este atributo con una fecha atrasada, el explorador borrará la cookie.
3. El atributo `'domain'` define el nombre de dominio asociado con la cookie. Sólo este dominio podrá tener acceso a la información almacenada por la cookie.
4. El atributo `'path'` define cuáles secciones del dominio especificado en el atributo `'domain'` pueden tener acceso a la cookie. Al establecerlo en la raíz del servidor (`/`) se permite que todo el dominio tenga acceso a la información almacenada en la cookie.
5. El atributo `'secure'` indica si una conexión HTTP segura es indispensable, antes de que se tenga acceso a la cookie.

Encabezados de cookies

Las cookies se transmiten entre el explorador del usuario y el sitio Web remoto a través de encabezados HTTP. Por ejemplo, al establecer una cookie, el sitio Web debe enviar al explorador cliente un encabezado `'Set-Cookie:'` que contenga los atributos necesarios. El siguiente código ejemplifica los encabezados enviados para crear dos cookies para un dominio:

```
Set-Cookie: username=john; path=/; domain=.estesitio.com;
expires=Friday, 25-Jan-08 23:59:50 IST
Set-Cookie: location=UK; path=/; domain=.estesitio.com;
expires=Friday, 25-Jan-08 23:59:50 IST
```

De manera similar, si una cookie en particular es válida para un sitio Web y su ruta de acceso, el explorador del usuario automáticamente incluirá la información de esta cookie en un encabezado `'Cookie:'` cuando solicite el URL del sitio en cuestión. En el ejemplo anterior, la siguiente ocasión que el usuario visite el sitio "estesitio.com" su explorador incluirá automáticamente el siguiente encabezado en la solicitud:

```
Cookie: username=john; location=UK
```

Pregunta al experto

P: ¿Puedo leer las cookies almacenadas en mi computadora?

R: Las cookies son archivos de texto almacenados en tu computadora y, como tales, puedes leerlos con cualquier procesador de texto. La ubicación exacta del lugar donde se almacenan depende del explorador y el sistema operativo que estés utilizando. Por ejemplo, en Microsoft Windows, Internet Explorer almacena las cookies como archivos independientes en la ruta de acceso *C:/Documents and Settings/[nombreusuario]/cookies*, mientras Mozilla Firefox almacena todas sus cookies en un solo archivo en *C:/Documents and Settings/[nombreusuario]/Application Data/Mozilla/Firefox/Profiles/[nombreperfil]/cookies.txt*.

Establecer cookies

La API de PHP para manipular cookies es muy sencilla; consiste en una sola función: `setcookie()`, que puede utilizarse para establecer y eliminar cookies. Esta función acepta seis argumentos: el nombre y valor de la cookie, la fecha de expiración en formato sello cronológico de UNIX, el dominio y ruta de acceso y un valor booleano para establecer el atributo `'secure'` como verdadero o falso. Esta función regresa un valor verdadero si el encabezado de la cookie fue transmitido con éxito a la computadora del usuario; sin embargo, esto no es indicación de que la cookie haya sido establecida con éxito (en caso de que la computadora del usuario haya sido configurada para rechazar todas las cookies, el encabezado pudo ser transmitido con éxito pero es posible que la cookie en sí no haya sido establecida en la computadora del usuario).

He aquí un ejemplo, que establece una cookie que contiene el correo electrónico del usuario:

```
<?php
// establece cookie
setcookie('email', 'john@sitioweb.com', mktime()+129600, '/');
?>
```

Es posible establecer múltiples cookies, invocando una función `setcookie()` para cada una de ellas. He aquí un ejemplo que establece tres cookies con diferentes periodos de validez y rutas de acceso:

```
<?php
// establece múltiples cookies
setcookie('username', 'ballenablanca', mktime()+129600, '/');
setcookie('email', 'john@sitioweb.com', mktime()+86400, '/');
setcookie('puesto', 'moderador', mktime()+3600, '/admin');
?>
```

PRECAUCIÓN

Dado que las cookies se establecen con el uso de encabezados HTTP, la invocación `setcookie()` debe anteceder a cualquier dato de salida generado por tu script. Cualquier violación a esta regla no sólo impedirá que la cookie se establezca en el equipo del usuario; además generará una serie de mensajes de error PHP.

Leer cookies

Las cookies establecidas por un dominio pueden ser leídas con la matriz asociativa especial `$_COOKIE` utilizando los scripts PHP que se ejecutan en dicho dominio. Es posible acceder a estas cookies con el uso de la notación de matrices estándar. He aquí un ejemplo:

```
<?php
// leer cookie
if(isset($_COOKIE['email'])) {
  echo 'Bienvenido de nuevo, ' . $_COOKIE['email'] . '!';
```

```
} else {
  echo '¡Hola, nuevo usuario!';
}
?>
```

Eliminar cookies

Para eliminar una cookie utiliza `setcookie()` con su nombre para establecer su fecha de validez con un valor pasado, como se muestra aquí:

```
<?php
// eliminar cookie
$ret = setcookie('puesto', 'moderador', mktime()-1600, '/admin');
if ($ret) {
  echo 'Encabezados de cookie transmitidos con éxito.';
}
?>
```

Prueba esto 9-1 Guardar y restablecer preferencias del usuario

Construyamos ahora una aplicación sencilla que utilice cookies para guardar y restaurar las preferencias de un usuario. El formulario Web del siguiente ejemplo solicita al usuario que seleccione sus preferencias para un viaje largo en avión y guarda estas preferencias en una cookie en el equipo del usuario. Cuando regresa a esta página, las preferencias establecidas con anterioridad son leídas de la cookie y restauradas automáticamente.

He aquí el código (*preferencias-vuelo.php*):

```
<?php
// si la forma ya ha sido enviada
// escribe la cookie con la configuración
if(isset($_POST['submit'])) {
  $ret1 = (isset($_POST['nombre'])) ? setcookie('nombre',
$_POST['nombre'], mktime() + 36400, '/') : null;
  $ret2 = (isset($_POST['asiento'])) ? setcookie('asiento',
$_POST['asiento'], mktime() + 36400, '/') : null;
  $ret3 = (isset($_POST['comida'])) ? setcookie('comida',
$_POST['comida'], mktime() + 36400, '/') : null;
  $ret4 = (isset($_POST['ofertas'])) ? setcookie('ofertas',
implode(',',$_POST['ofertas']), mktime() + 36400, '/') : null;
}

// lee la cookie y asigna sus valores
//  a variables PHP
$nombre = isset($_COOKIE['nombre']) ? $_COOKIE['nombre’] : '';
```

```
$asiento = isset($_COOKIE['asiento']) ? $_COOKIE['asiento'] : '';
$comida = isset($_COOKIE['comida']) ? $_COOKIE['comida'] : '';
$ofertas = isset($_COOKIE['ofertas']) ? explode (',',$COOKIE['ofertas'])
: array();
?>
<!DOCTYPE html PUBLIC «-//W3C//DTD XHTML 1.0 Transitional//EN»
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 9-1: Guardar y restaurar preferencias del usuario</
title>
  </head>
  <body>
    <h2>Proyecto 9-1: Guardar y restaurar preferencias del usuario</h2>
    <h3>Seleccione sus Preferencias de Vuelo</h3>
    <?php
    // si el formulario ya ha sido enviado
    // despliega mensaje de éxito
    if (isset($_POST['submit'])) {
    ?>
    Gracias por su selección.
    <?php
    // si el formulario no ha sido enviado
    // muestra el formulario
    } else {
    ?>
    <form method="post" action="preferencias-vuelo.php">
      Nombre: <br />
      <input type="text" size="20" name="nombre" value"<?php echo $name;
?>" />

      <p>

      Selección de asiento: <br />
      <input type="radio" name="asiento" value="pasillo" <?php echo
($asiento == 'pasillo') ? 'checked' : ''; ?>>Pasillo</input>
      <input type="radio" name="asiento" value="ventana" <?php echo
($asiento == 'ventana') ? 'checked' : ''; ?>>Ventana</input>
      <input type="radio" name="asiento" value="centro" <?php echo
($asiento == 'centro') ? 'checked' : ''; ?>>Centro</input>

      <p>

      Selección de comida: <br />
      <input type="radio" name="comida" value="normal-veg" <?php echo
($comida == 'normal-veg') ? 'checked' : ''; ?>>Vegetariana</input>
      <input type="radio" name="comida" value="normal-nveg" <?php echo
($comida == 'normal-nveg') ? 'checked' : ''; ?>>No Vegetariana</input>
      <input type="radio" name="comida" value="diabética" <?php echo
```

(*continúa*)

```
($comida == 'diabética') ? 'checked' : ''; ?>>Diabética</input>
      <input type="radio" name="comida" value="niño" <?php echo
($comida == 'niño') ? 'checked' : ''; ?>>Niño</input>
      <p>

      Estoy interesado en ofertas especiales de los vuelos a: <br />
      <input type="checkbox" name="ofertas[]" value="LHR" <?php echo in_
array('LHR', $ofertas) ? 'checked' : ''; ?>>Londres (Heathrow)</input>
      <input type="checkbox" name="ofertas[]" value="CDG" <?php echo in_
array('CDG', $ofertas) ? 'checked' : ''; ?>>París</input>
      <input type="checkbox" name="ofertas[]" value="CIA" <?php echo in_
array('CIA', $ofertas) ? 'checked' : ''; ?>>Roma (Ciampino)</input>
      <input type="checkbox" name="ofertas[]" value="IBZ" <?php echo in_
array('IBZ', $ofertas) ? 'checked' : ''; ?>>Ibiza</input>
      <input type="checkbox" name="ofertas[]" value="SIN" <?php echo in_
array('SIN', $ofertas) ? 'checked' : ''; ?>>Singapur</input>
      <input type="checkbox" name="ofertas[]" value="HKG" <?php echo in_
array('HKG', $ofertas) ? 'checked' : ''; ?>>Hong Kong</input>
      <input type="checkbox" name="ofertas[]" value="MLA" <?php echo in_
array('MLA', $ofertas) ? 'checked' : ''; ?>>Malta</input>
      <input type="checkbox" name="ofertas[]" value="BOM" <?php echo in_
array('BOM', $ofertas) ? 'checked' : ''; ?>>Bombay</input>

      <p>

      <input type="submit" name="submit" value="Enviar" />
    </form>
    <?php
   }
   ?>
  </body>
</html>
```

Este formulario Web contiene varios campos para que el usuario ingrese su nombre, seleccione un asiento y tipo de comida y acepte recibir ofertas especiales. La figura 9-1 muestra su apariencia.

Cuando este formulario es enviado, las opciones seleccionadas por el usuario son guardadas en cookies en su computadora. Como resultado, cada vez que el usuario vuelve a visitar el formulario, los datos de las cookies son leídos por PHP en `$_COOKIE`. El formulario Web puede usar entonces estos datos de cookies para llenar de manera automática los campos de acuerdo con el último envío del usuario. De esta manera, la página parece "recordar" las preferencias del usuario en cada visita.

Las cookies son almacenadas en el equipo del usuario y pueden ser vistas utilizando cualquier procesador de textos. Algunos exploradores también te permiten ver las cookies con ayuda de herramientas integradas. Por ejemplo, en Mozilla Firefox, puedes ver el contenido de todas las cookies guardadas en tu computadora con el menú Herramientas | Opciones | Privacidad | Mostrar Cookies, como se presenta en la figura 9-2.

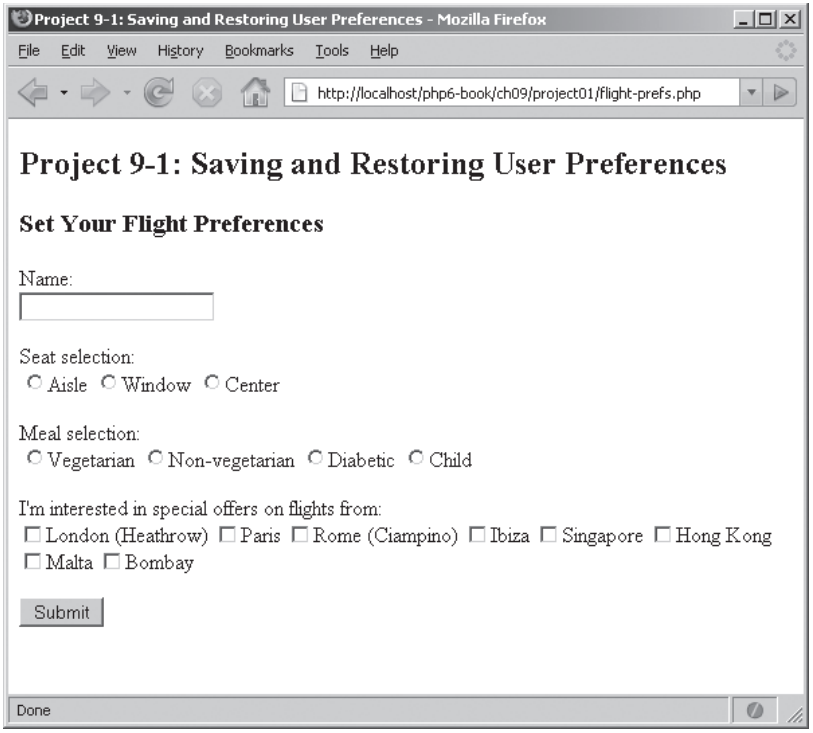


Figura 9-1 Formulario Web para que el usuario ingrese sus preferencias para el vuelo

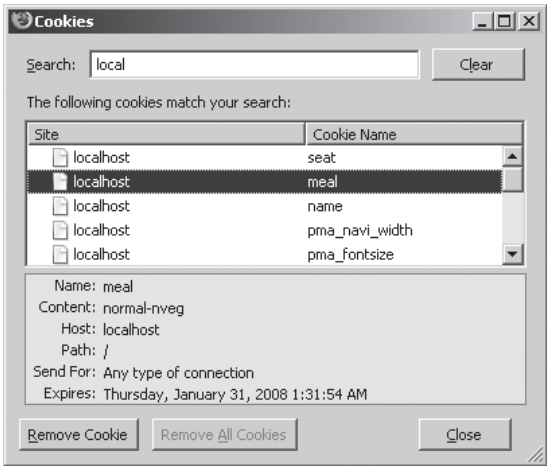


Figura 9-2 Ver cookies en Mozilla Firefox

Trabajar con sesiones

Como las cookies, las sesiones también ofrecen un método para mantener el estado de un sitio Web o una aplicación, sólo que utiliza un método algo diferente. Las siguientes secciones explican el funcionamiento de las sesiones con PHP, además de las funciones de PHP para crear y utilizar sesiones dentro de una aplicación Web.

Aspectos básicos de las sesiones

Ahora ya sabes lo que son las cookies: archivos de texto almacenados en el equipo del usuario que ayudan al sitio Web o la aplicación a reconocer al usuario y recuperar información específica sobre el mismo. El problema con las cookies es que no son muy seguras; como se almacenan en el equipo cliente, es posible que el usuario abra el archivo de las cookies, las lea o modifique la información que contienen, en ocasiones con malas intenciones.

Por ello muchos sitios Web prefieren utilizar *sesiones*. Las sesiones funcionan de manera muy similar a las cookies, salvo que la información utilizada para mantener el estado se almacena en el servidor y no en el equipo cliente. En un ambiente basado en sesiones, cada cliente es identificado por un número único (llamado *identificador de sesión*) y este número se utiliza para vincular a cada cliente con su información almacenada en el servidor. Cada vez que el cliente visita el sitio Web o despliega la aplicación, el sitio lee el identificador de sesión del cliente y restaura la información correspondiente desde un repositorio de datos ubicado en el servidor.

Bajo este sistema, la información se almacena en una base de datos SQL o en un archivo de texto en el servidor; como resultado, los usuarios no pueden tener acceso ni modificar la información, por lo que el sistema entero es mucho más seguro. El *identificador de sesión* en sí puede almacenarse en el equipo del usuario como cookie, o puede pasarse de página en página en el URL. Con PHP, esta cookie es llamada `PHPSESSID`.

Las siguientes secciones abordan las funciones PHP para crear sesiones, registrar y utilizar variables de sesión y destruir sesiones.

Crear sesiones y variables de sesión

Es fácil iniciar una nueva sesión con PHP: simplemente invoca la función `session_start()` para crear una nueva sesión y generar un ID de sesión para el cliente. Una vez que se ha creado la sesión, es posible crear y adjuntar cualquier número de *variables de sesión* para ella; éstas son variables regulares, en el sentido de que pueden almacenar información en forma de texto o números, pero también son especiales porque sólo existen durante la sesión, mientras el usuario navega por el sitio.

Las variables de sesión son "registradas" al guardarlas como pareja clave-valor en una matriz asociativa `$_SESSION`. Como `$_POST` y `$_GET`, esta matriz siempre está disponible de manera global y puede accederse directamente desde cualquier punto de tu script PHP.

Para ver cómo funcionan las sesiones y las variables de sesión, examina el siguiente script, que crea una nueva sesión de cliente y registra dos variables de sesión:

```
<?php
// inicia sesión
session_start();

// registra variables de sesión
$_SESSION['nombre'] = 'Ronald';
$_SESSION['especie’] = 'Conejo';
?>
```

PRECAUCIÓN

Por lo general, la función `session_start()` establece una cookie que contiene el ID de sesión en el equipo cliente. Por ello, al igual que con la función `setcookie()`, la invocación `session_start()` debe anteceder a cualquier dato de salida generado por el script. Esto se debe a restricciones en el protocolo HTTP que requiere que los encabezados de las cookies se envíen antes que los datos de salida del script.

Ahora es posible acceder a estas variables de sesión desde cualquier página del mismo dominio Web. Para ello, crea un nuevo script PHP, recrea la sesión invocando `session_start()` y luego trata de accesar los valores de la matriz asociativa `$_SESSION`, como en el siguiente ejemplo:

```
<?php
// inicia sesión
session_start();

// lee las variables de sesión
// datos de salida: 'Bienvenido de nuevo, Ronald Conejo'
echo 'Bienvenido de nuevo, ' . $_SESSION['nombre'] . ' ' . $_
SESSION['especie'] ;
?>
```

Pregunta al experto

P: ¡Ayuda! Mis sesiones de variables no se están guardando. ¿Qué hago ahora?

R: Si una variable de sesión no se está registrando correctamente, existen un par de posibilidades que debes revisar antes de darte por vencido y llamar a la policía de sesiones:

- Por defecto, PHP guarda los datos de sesión en el directorio */tmp* del servidor. Sin embargo, si estás utilizando Microsoft Windows, este directorio no existe por defecto y tus sesiones no se almacenarán correctamente. Para rectificarlo, abre el archivo de configuración de PHP, *php.ini*, y edita la variable `'session.save_path'` para escribir el directorio temporal de tu computadora o servidor.

(*continúa*)

- Por defecto, PHP establece una cookie en el equipo cliente con el identificador de sesión. Si el explorador del usuario está configurado para rechazar todas las cookies, este identificador de sesión no se almacenará y la información de la sesión no se mantendrá de página en página. Para corregir esto, puedes transmitir el identificador de sesión de una página a otra como parte de la cadena URL (aunque esto es menos seguro) estableciendo como verdadera la variable `'session.use_trans_sid'` en el archivo de configuración de PHP.
- Cada sesión PHP tiene un *valor de término* (la duración, medida en segundos) de la sesión en ausencia de cualquier actividad por parte del usuario. En algunos casos, este valor de término puede establecerse demasiado bajo para la aplicación, y el resultado es que la aplicación se destruye demasiado pronto. Puedes ajustar este valor desde la variable `'session.gc_maxlifetime'` en el archivo de configuración PHP.

Eliminar sesiones y variables de sesión

Para eliminar una variable de sesión específica, simplemente aplica la función `unset()` a la correspondiente clave dentro de la matriz `$_SESSION`:

```
<?php
// inicia sesión
session_start();

// elimina variable de sesión
unset($_SESSION['nombre']);
    ?>
```

Como opción, para eliminar todas las variables de sesión y la sesión misma, utiliza la función `session_destroy()`:

```
<?php
// inicia sesión
session_start();

// eliminar sesión
session_destroy();
?>
```

Es importante advertir que antes de destruir la sesión con `session_destroy()`, primero necesitas recrear el ambiente de sesión (para que haya algo que destruir) con la función `session_start()`.

Prueba esto 9-2 Rastrear visitas previas a la página

Pongamos ahora toda esta teoría en contexto, construyendo una aplicación sencilla que muestre el funcionamiento de las sesiones. El siguiente ejemplo utiliza una sesión para registrar cada visita hecha por un usuario particular a una página Web. En cada visita, el script presenta las fechas y horas de todas las visitas anteriores y añade a la sesión un registro para la visita actual.

He aquí el código (*visitas.php*):

```
<?php
// inicia sesión
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 9-2: Rastrear visitas previas a la página</title>
  </head>
  <body>
    <h2>Proyecto 9-2: Rastrear visitas previas a la página</h2>
    <?php
    if (!isset($_SESSION['visitas'])) {
      echo 'Esta es su primera visita.';
    } else {
      echo 'Visitó esta página en: <br />';
      foreach ($_SESSION['visitas'] as $v) {
        echo date('d M Y h:i:s', $v) . '<br />';
      }
    }
    ?>
  </body>
</html>
<?php
// añade el sello temporal de la visita actual
$_SESSION['visitas'] [] = mktime();
?>
```

Para ver este script en acción, visita la página Web unas cuantas veces y verás una lista creciente que contiene los registros de las anteriores visitas. Esta lista se mantiene disponible mientras no cierres el explorador, de manera que aunque visites algunos otros sitios y luego regreses al script, seguirás viendo los registros de tus visitas anteriores. Sin embargo, una vez que cierres el explorador, la cookie de sesión será destruida y la lista comenzará de nuevo.

(*continúa*)

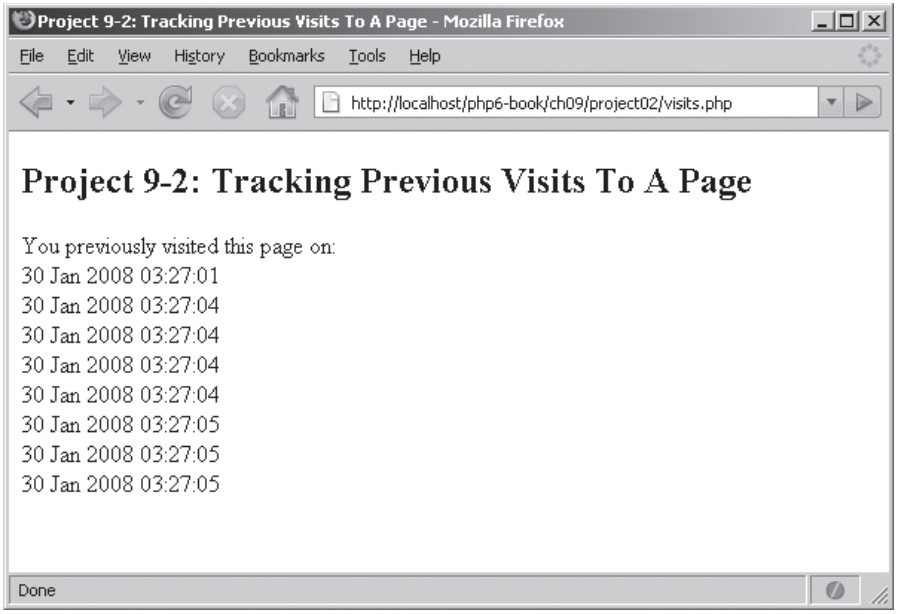


Figura 9-3 Rastreo de visitas previas a una página Web

La figura 9-3 muestra un ejemplo de lo que probablemente verás.

Este script funciona creando una sesión cada vez que el usuario visita la página y almacena el sello cronológico de la visita en la variable de sesión `$_SESSION['visitas']`. En cada visita subsecuente la sesión se recrea, la matriz que contiene el sello temporal de las visitas previas se restaura y se utiliza un bucle `foreach` para iterar sobre la matriz y presentar los registros en formato de fecha y hora legible para los humanos.

Utilizar encabezados HTTP

En secciones anteriores viste que PHP envía automáticamente encabezados al explorador cliente para establecer cookies. Como desarrollador PHP, éstos no son los únicos encabezados que puedes enviar. PHP te permite enviar al explorador cliente cualquier encabezado soportado por el protocolo HTTP, a través de la función `header()`.

Quizás el encabezado de mayor uso sea `'Location:'`, utilizado para redireccionar el explorador del usuario a un URL diferente sin que el usuario lo note. He aquí un ejemplo:

```
<?php
// redireccionar a www.php.net
header('Location: http://www.php.net');
?>
```

También puedes enviar otros encabezados, por ejemplo: `'Cache-Control'` o `'Content-Encoding'`, como en el siguiente ejemplo:

```
<?php
// controla el caché
header('Cache-Control: no-cache');

// establece el tipo de contenido
header('Content-type: text/xml');

// establece la codificación del contenido
echo "<?xml version='1.0'?><doc><element/></doc>";
?>
```

Como sabes, enviar un encabezado HTTP después de que el script haya generado datos de salida producirá un error. Puedes evitar este error probando primero si algún encabezado ya ha sido enviado; para ello utiliza la función de PHP `headers_sent()`. Examina el siguiente ejemplo, que lo ilustra:

```
<?php
// comprueba si algún encabezado ha sido enviado
// de no ser así, envía los encabezados
// en caso de envío anterior muestra un mensaje de error
if (!headers_sent()) {
  header('Cache-Control: no-cache');
  header('Content-type: text/plain');
  echo 'Encabezados enviados.';
} else {
  die ('ERROR: ¡No es posible enviar encabezados!');
}
?>
```

TIP

Para una lista completa de los encabezados que soporta el protocolo HTTP, consulta la especificación del mismo en www.w3.org/Protocols/rfc2616/rfc2616.html, o en Wikipedia en.wikipedia.org/wiki/List_of_HTTP_headers.

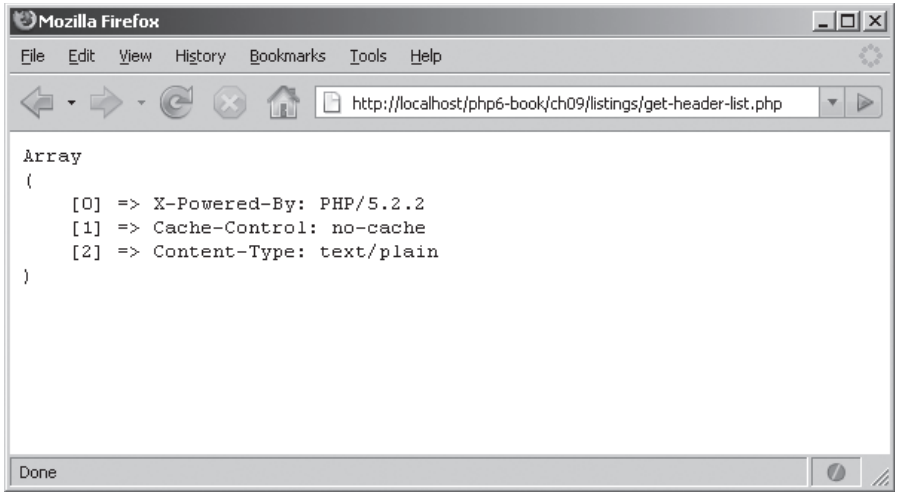


Figura 9-4 Lista de los encabezados HTTP que serán enviados al cliente

Para obtener una lista completa de los encabezados que serán enviados al explorador del usuario, utiliza la función `headers_list()`. He aquí un ejemplo:

```
<?php
// controla el caché
header('Cache-Control: no-cache');

// establece el tipo de contenido
header('Content-type: text/plain');

// presenta la lista de encabezados
print_r(headers_list());
?>
```

La figura 9-4 muestra los datos de salida de este script.

Prueba esto 9-3 Construir un formulario de ingreso mejorado

En el capítulo 7 construiste un formulario de inicio de sesión que actuaba dinámicamente con la base de datos MySQL para verificar los permisos del usuario. Ahora, mejoremos un poco ese formulario con sesiones, cookies y encabezados. El siguiente ejemplo enriquecerá el formulario creado en el capítulo 7 para recordar el nombre de usuario que se haya ingresado y para restringir el acceso a ciertas páginas a las que pueden acceder sólo los usuarios que han iniciado sesión.

Dando por hecho que configuraste la base de datos MySQL de acuerdo con las instrucciones del capítulo 7, aquí está el formulario de ingreso modificado (*ingreso.php*):

```
<?php
// si el formulario no ha sido enviado
// muestra el formulario
if (!isset($_POST['submit'])) {
  $nombredeusuario = (isset($_COOKIE['nombre'])) ? $_COOKIE['nombre'] :
'';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
   <title>Proyecto 9-3: Construir un formulario de ingreso mejorado</title>
  </head>
  <body>
    <h2>Proyecto 9-3: Construir un formulario de ingreso mejorado</h2>
    <form method="post" action="ingreso.php">
      Nombre de Usuario: <br />
      <input type="text" name="nombredeusuario" value="<?php echo
$nombredeusuario; ?>"/>
      <p>
      Contraseña: <br />
      <input type="password" name="contrasena" />
      <p>
      <input type="checkbox" name="rastreo" checked />
      Recuérdame
      <p>
      <input type="submit" name="submit" value="Ingresar" />
    </form>
  </body>
</html>
<?php
// si el formulario ha sido enviado
// verifica los datos proporcionados
// contra la base de datos
} else {
  $nombredeusuario = $_POST['nombredeusuario'];
  $contrasena = $_POST['contrasena'];

  // verifica datos de entrada
  if (empty($nombredeusuario)) {
    die('ERROR: Por favor escriba su nombre de usuario');
  }
  if (empty($contrasena)) {
    die('ERROR: Por favor escriba su contraseña');
  }
```

(*continúa*)

```
  // intenta establecer conexión con la base de datos
  try {
     $pdo = new PDO('mysql:dbname=app;host=localhost', 'user', 'pass');
  } catch (PDOException $e) {
     die("Error: No fue posible conectar: " . $e->getMessage());
  }

  // limpia los caracteres especiales de los datos de entrada
$nombredeusuario = $pdo->quote($nombredeusuario);

  // verifica si existe el nombre de usuario
  $sql = "SELECT COUNT(*) FROM usuarios WHERE nombredeusuario =
$nombredeusuario";
  if($result = $pdo->query($sql)) {
    $row = $result->fetch();
    // si es positivo, busca la contraseña cifrada
    if($row[0] == 1) {
      $sql = "SELECT contrasena FROM usuarios WHERE nombredeusuario =
$nombredeusuario";
      // cifra la contraseña ingresada en el formulario
      // la verifica contra la contraseña cifrada que reside en la base
de datos
      // si ambas coinciden, la contraseña es correcta
      if ($result = $pdo->query($sql)) {
        $row = $result->fetch();
        $salt = $row[0];
        if (crypt($contrasena, $salt) == $salt) {
          // contraseña correcta
          // inicia una nueva sesión
          // guarda el nombre del usuario para la sesión
          // de requerirse, establece una cookie con el nombre de usuario
          // redirecciona el explorador a la página principal de la
aplicación
          session_start();
          $_SESSION['nombredeusuario'] = $nombredeusuario;
          if ($_POST['rastreo']) {
            setcookie('nombre', $_POST['nombredeusuario'],
mktime()+86400);
          }
          header('Location: principal.php');
        } else {
          echo 'Ha ingresado una contraseña incorrecta.';
        }
      } else {
           echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo-
>errorInfo());
      }
    } else {
      echo 'Ha ingresado un nombre de usuario incorrecto.';
    }
```

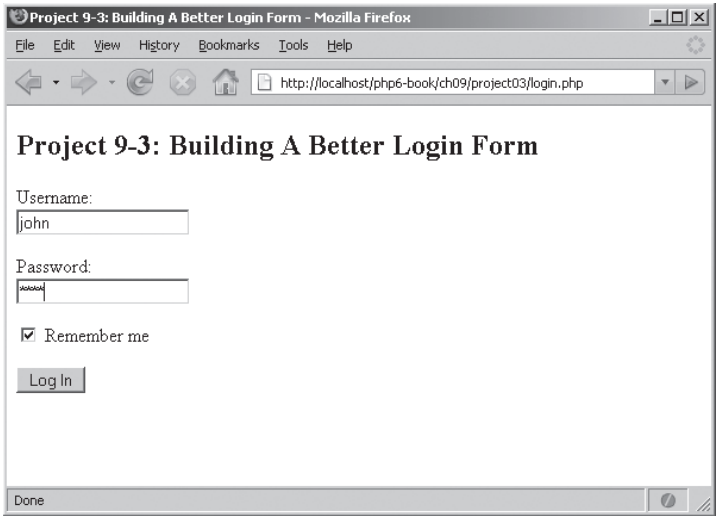


Figura 9-5 Un formulario Web para ingresar en una aplicación

```
  } else {
    echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo-
>errorInfo());
  }

  // cierra conexión
  unset($pdo);
}
?>
```

La figura 9-5 muestra el formulario de ingreso.

Cuando este formulario es enviado, la segunda mitad del script verifica el nombre de usuario y la contraseña contra los valores almacenados en la base de datos, utilizando el procedimiento explicado en el capítulo 7. Sin embargo, hay diferencias importantes: en esta versión, en lugar de simplemente generar un mensaje de éxito en caso de que el nombre de usuario y la contraseña sean válidos, el script comienza una nueva sesión y registra el nombre de usuario en una variable de sesión.

A continuación, el script verifica si se ha seleccionado la opción "Recuérdame" del formulario Web. De ser así, el script coloca una cookie en el equipo cliente con el fin de almacenar el nombre del usuario para posteriores visitas. La siguiente vez que el usuario visite esta página, la cookie establecida en el paso anterior será leída automáticamente y aparecerá el nombre de usuario en el campo correspondiente dentro del formulario. Una vez que se han establecido la sesión y la cookie, el script utiliza la función `header()` para redireccionar el explorador del usuario a la página principal de la aplicación, *principal.php*.

Pero esto es sólo la mitad de la historia. Con el fin de restringir el acceso sólo a los usuarios que han iniciado sesión, es necesario verificar la existencia de una sesión válida en otras páginas del sitio. Para ejemplificarlo, observa la página principal de la aplicación (*principal.php*), que implementa esta verificación:

```
<?php
// recrea sesión
// verifica si el usuario firmadota iniciado sesión
// de no ser así, muestra un mensaje de error y detiene el proceso
session_start();
if (!isset($_SESSION['nombredeusuario'])) {
  die('ERROR: Ha intentado ingresar a una página restringida. Por favor
<a href="login.php">Regístrese</a>.');
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
   <title>Proyecto 9-3: Construir un formulario de ingreso mejorado</title>
  </head>
  <body>
    <h2>Proyecto 9-3: Construir un formulario de ingreso mejorado</h2>
    Ésta es la página principal de la aplicación.
    <p/>
    Verás esta página después de ingresar exitosamente.
    <p/>
  </body>
</html>
```

La figura 9-6 muestra lo que verán los usuarios que ingresan con éxito.

Figura 9-6 El resultado de acceder con éxito

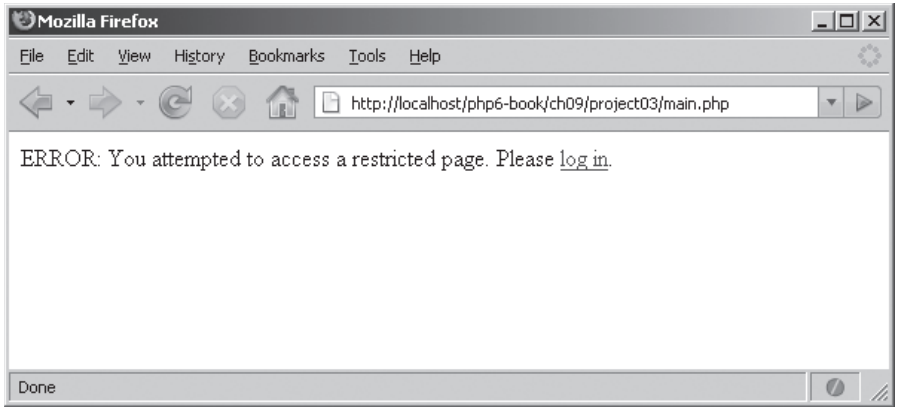


Figura 9-7 El resultado de acceder a una página segura sin haber firmado el acceso con anterioridad

La figura 9-7 muestra lo que verán los usuarios que intentan ingresar a la página sin haber iniciado sesión.

Resumen

Este capítulo mostró dos de las características más útiles y refinadas de PHP: la capacidad de trabajar con restricciones en el protocolo HTTP y la facultad de mantener el estatus a través de sesiones y cookies. Estas características suelen ser utilizadas por los desarrolladores para mejorar y enriquecer la experiencia de los usuarios en los sitios basados en PHP, y como comprobaste en las páginas anteriores, son muy fáciles de implementar.

Además de la información teórica sobre las sesiones, cookies y encabezados, este capítulo también incluyó ejemplos prácticos para poner en contexto real la teoría. Una tarea común en Web (permitir el acceso a ciertas partes del sitio sólo a los usuarios que han iniciado sesión) se utilizó para demostrar la manera en que pueden utilizarse sesiones, cookies y encabezados de manera conjunta para construir una aplicación segura y funcional.

Para leer más sobre los temas abordados en este capítulo, visita los siguientes sitios:

- Cookies en PHP, **www.php.net/setcookie**
- Sesiones en PHP, **www.php.net/session**
- Encabezados HTTP en PHP, **www.php.net/header**

Autoexamen Capítulo 9

1. ¿Cuál es la diferencia entre una sesión y una cookie?
2. ¿Cómo eliminas una cookie establecida con anterioridad?
3. ¿Cómo se registra una variable de sesión? ¿Y cómo se accede a su valor desde una página diferente?
4. ¿Qué errores tiene el siguiente script PHP? Sin ejecutarlo, imagina cuáles serían los datos de salida.

```
<?php
echo 'Redireccionándolo ...';
header('Location: http://www.php.net');
?>
```

5. Escribe un programa que determine cuántas veces ha visitado una página en particular un usuario:
   - **A** Dentro de la misma sesión.
   - **B** En diferentes sesiones.
6. Revisa el último proyecto de este capítulo. Después, enriquécelo con un script PHP que saque a los usuarios de la aplicación y los redireccione al formulario de inicio de sesión.

Parte III

Seguridad y solución de problemas

Capítulo 10

Manejo de errores

Habilidades y conceptos clave

- Comprender los niveles de error de PHP
- Controlar cuáles errores son desplegados en tu script PHP
- Desviar el manejador de errores por defecto de PHP y desviar los errores a una función personalizada
- Comprender cómo generar y manejar excepciones
- Enrutar automáticamente los errores a un archivo o una dirección de correo electrónico
- Generar una ruta alterna para depurar errores de script

Una mala interpretación común, sobre todo entre los desarrolladores poco experimentados, es concebir que un "buen" programa es el que funciona sin errores. En realidad, esto no es completamente cierto; una mejor definición sería que un buen programa es el que anticipa todas las posibles condiciones que provocarían errores y lidia con estas posibilidades de manera consistente y correcta.

Escribir programas "inteligentes" conforme a esta última definición es, a la vez, un arte y una ciencia. Experiencia e imaginación desempeñan un importante papel en la anticipación de causas potenciales de error y su respectiva acción correctiva, pero no menos importante es el lenguaje de programación, que define las herramientas y funciones que están disponibles para atrapar y corregir los errores.

Por fortuna, PHP no es perezoso en este aspecto: el lenguaje viene acompañado de un conjunto de sofisticadas herramientas que ayuda a los desarrolladores a capturar los errores y ponerles remedio. Este capítulo te presenta una introducción a este conjunto de herramientas; te mostrará el modelo de excepciones de PHP 5.3 y te enseñará a crear rutinas personalizadas para el manejo de errores a la medida de las necesidades de tu aplicación PHP.

Manejo de errores de script

A medida que has recorrido los proyectos de este libro, sin duda alguna has tenido algunos accidentes: una llave mal colocada por aquí, un punto y coma omitido por allá, tal vez alguna invocación equívoca en algún otro lugar. Y habrás notado que PHP es muy efectivo para señalar estos errores. En algunos casos, habrá generado un mensaje de error pero siguió ejecutan-

do tu script; en otros casos, más serios, habrá detenido la ejecución del script con un mensaje que indica el número de la línea que causa el error.

Los tipos de error descritos son de "nivel de script"; surgen cuando el motor de PHP encuentra defectos en la sintaxis o la estructura de un script PHP. Por lo general, sólo se hacen visibles una vez que PHP comienza con la segmentación y ejecuta el script. Para ejemplificarlo, intenta crear y ejecutar el siguiente script:

```
<?php
// intenta dividir entre cero
echo 45/0;

// intenta invocar una función no definida
echo unaFuncion();
?>
```

Los datos de salida de este script deben ser semejantes a lo que aparece en la figura 10-1.

Como muestra la figura 10-1, este script genera dos tipos de errores: una "advertencia" por el intento de dividir entre cero, y un "error fatal" por el intento de invocar una función indefinida. En realidad, los errores de PHP pueden clasificarse en gran medida en tres grandes categorías, que se presentan en la tabla 10-1.

Existe una clara jerarquía de los mensajes de error en PHP: las notificaciones son menos serias que las advertencias, que a su vez son menos serias que los errores fatales. Por defecto, PHP sólo muestra advertencias y errores fatales en los datos de salida del script (aunque, como verás en unos momentos, puedes cambiar este comportamiento por defecto de manera que aun las notificaciones sean visibles en los datos de salida del script). Los errores pueden surgir en varias etapas durante la vida del script (al inicio, en la segmentación, en la compila-

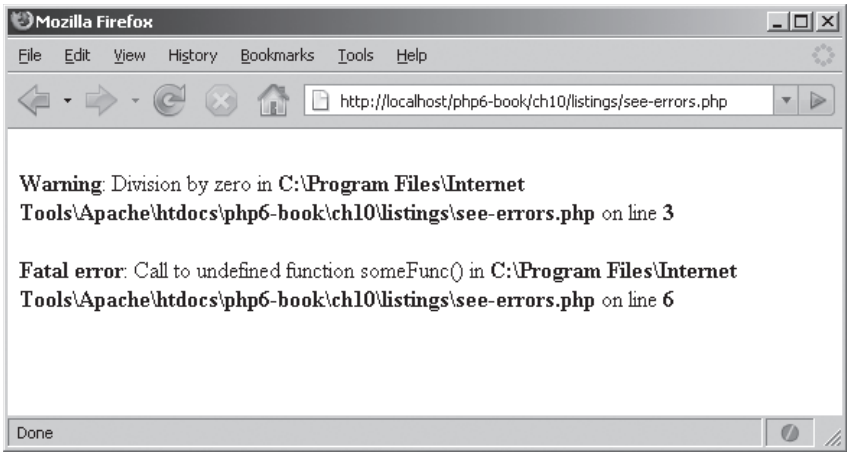


Figura 10-1 Ejemplo de página de error PHP

Tipo de error	Descripción	Ejemplo
Notificaciones	Errores no críticos que no detienen la ejecución del script	Acceder a una variable que no se ha inicializado
Advertencias	Errores más serios que requieren atención, pero no detienen la ejecución del script (aunque es posible que algunas partes del script no funcionen correctamente)	Leer un archivo que no existe en la ruta de acceso declarada
Errores fatales	Errores de sintaxis, o errores críticos que obligan a PHP a detener la ejecución del script	Crear una instancia de objeto de una clase indefinida

Tabla 10-1 Categorías de errores en PHP

ción o en la ejecución) y por lo mismo, PHP también hace distinciones internas de estas etapas. En conjunto, son doce diferentes niveles de error (más dos niveles "especiales"), representados por constantes. Puedes obtener una lista completa de estos niveles de error en www.php.net/manual/en/ref.errorfunc.php#errorfunc.constants; la tabla 10-2 presenta las constantes que verás con mayor frecuencia.

Es fácil entender casi todos estos niveles de error. Tal vez los únicos que presenten problemas sean los niveles `E_USER`, que se clasifican aparte de los errores personalizados en el nivel de la aplicación. No debes preocuparte por ellos, ya que fueron sustituidos por el nuevo modelo de excepciones introducido en PHP 5.

Nivel de error	Descripción
`E_PARSE`	Errores fatales de análisis sintáctico
`E_NOTICE`	Errores no fatales durante la etapa de ejecución (notificaciones)
`E_WARNING`	Errores no fatales durante la etapa de ejecución (advertencias)
`E_ERROR`	Errores fatales durante la etapa de ejecución que imponen la interrupción del script
`E_USER_NOTICE`	Errores no fatales definidos por el usuario (notificaciones)
`E_USER_WARNING`	Errores no fatales definidos por el usuario (advertencias)
`E_USER_ERROR`	Errores de aplicación fatal definidos por el usuario
`E_STRICT`	Errores no fatales durante la etapa de ejecución que surgen por errores de sintaxis PHP obsoleta
`E_ALL`	Todos los errores

Tabla 10-2 Niveles de error en PHP

Controlar el reporte de errores

Puedes controlar cuáles errores mostrará el script con la función PHP integrada `error_reporting()`. Esta función acepta una o más de las constantes que aparecen en la tabla 10-2 y le indica al script únicamente los errores que coinciden con cierto tipo. Sin embargo, hay una excepción: los errores de segmentación (`E_PARSE`) que surgen por defectos en la sintaxis en el script PHP no pueden ocultarse con la función `error_reporting()`.

Para ver cómo funciona, considera el siguiente código modificado a partir de un ejemplo anterior; en este caso "oculta" los errores que no son fatales:

```
<?php
// muestra sólo los errores fatales
error_reporting(E_ERROR);
echo 45/0;
?>
```

En este caso, cuando el script se ejecuta no se generará ninguna advertencia, aunque esté intentando hacer una división entre cero.

Pregunta al experto

P: ¿Qué hace el nivel de error `E_STRICT`?

R: En el nivel de error `E_STRICT`, PHP inspecciona tu código durante la ejecución y genera recomendaciones automáticas sobre la manera en que puede mejorarse. El uso de `E_STRICT` puede proporcionar recomendaciones sobre funciones que dejarán de existir en futuras versiones de PHP; aplicar tales recomendaciones puede mejorar el mantenimiento de tu código a largo plazo.

También puedes utilizar `error_reporting()` para evitar que aparezcan errores fatales durante la ejecución del script, como en el siguiente ejemplo:

```
<?php
//  muestra sólo las advertencias
error_reporting(E_WARNING);
echo unaFunción();
?>
```

Es importante destacar que `error_reporting()` no hace que el script quede libre de errores automáticamente; todo lo que hace es ocultar cierto tipo de errores. En el ejemplo anterior, aunque no se muestre un mensaje, se generará un error fatal y el script detendrá su ejecución en el punto donde se presenta el error.

También puedes transmitir a `error_reporting()` una combinación de niveles de error, para personalizar aún más el sistema de reporte de errores de PHP. Observa el siguiente ejemplo, que sólo reporta notificaciones y errores fatales, pero no advertencias:

```
<?php
// notifica sólo notificaciones y errores fatales
error_reporting(E_NOTICE | E_ERROR);
echo $var; //  notificación
echo 45/0; // advertencia
echo unaFunción(); // fatal
?>
```

También es posible deshabilitar de forma selectiva el reporte de errores, con base en funciones específicas, insertando un prefijo en la invocación de la función con el operador `@`. Por ejemplo, en condiciones normales, el siguiente código generaría un error fatal porque `unaFunción()` no existe:

```
<?php
// invoca una función inexistente
echo unaFunción();
?>
```

Sin embargo, este error puede omitirse insertando el símbolo @ antes de invocar la función, de esta manera:

```
<?php
// invoca una función inexistente
@echo unaFunción();
?>
```

Utilizar un controlador de errores personalizado

Por defecto, cuando se dispara un error, el controlador de errores integrado de PHP identifica su tipo, muestra el mensaje de error apropiado [basándose en la configuración `error_reporting()`] y, como opción, detiene la ejecución del script (en caso de que el error sea fatal). El mensaje generado por el controlador de errores utiliza una hoja modelo estándar: indica el tipo de error, la razón del mismo, el nombre del archivo y el número de línea donde se generó (ver figura 10-1 como ejemplo).

Sin embargo, mientras tus aplicaciones PHP se hagan más complejas, este mecanismo para el manejo de errores puede terminar siendo inadecuado. Por ejemplo, tal vez quieras personalizar la hoja modelo utilizada por el controlador de errores para mostrar mayor o menor cantidad de información, o quizá desees enviar el error a un archivo o a una base de datos, en lugar de mostrarlo al usuario. Para todas estas situaciones, PHP ofrece la función `set_error_handler()`, que te permite reemplazar el controlador de errores integrado de PHP por uno propio.

La función `set_error_handler()` acepta un solo argumento: el nombre de la función definida por el usuario que debe invocarse cuando ocurra el error. Esta función personalizada debe ser capaz de aceptar al menos dos argumentos obligatorios (el tipo de error y el mensaje descriptivo correspondiente) y un máximo de tres argumentos adicionales (el nombre del archivo, el número de línea donde ocurre el error y un lugar donde almacenar la variable en el momento del error).

NOTA

El controlador de errores personalizado no puede interceptar errores fatales `(E_ERROR)`, errores de análisis sintáctico `(E_PARSE)` ni notificaciones de sintaxis obsoleta `(E_STRICT)`.

Para mostrar cómo funciona, examina el siguiente ejemplo. Aquí, una función personalizada definida por el usuario reemplaza al controlador de errores integrado de PHP y genera dinámicamente una página de error personalizada:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title></title>
    <style type="text/css">
    .notice {
      font-weight: bolder;
      color: purple;
    }
    .warning {
      font-weight: bolder;
      font-size: larger;
      color: red;
    }
    </style>
  </head>
  <body>
<?php
// envía errores a un controlador personalizado
set_error_handler('miControlador');

// reporta todos los errores
error_reporting(E_ALL);

// genera algunos errores
echo $var;    // notificación
echo 23/0;    // advertencia
```

```
// controlador de errores personalizado
function miControlador($type, $msg, $file, $line, $context) {
  $text = "Ha ocurrido un error en la línea $line mientras se procesaba
su solicitud. <p>
         Por favor visite nuestra <a href=http://www.dominio.com>página
de inicio</a> y vuelva a intentarlo.";
  switch($type) {
    case E_NOTICE:
      echo "<div class=\"notice\">$text</div><p>";
      break;

    case E_WARNING:
      echo "<div class=\"warning\">$text</div><p>";
      break;
  }
}
?>
  </body>
</html>
```

Aquí, la función `set_error_handler()` envía automáticamente todos los errores a la función `miControlador()` definida por el usuario. Cuando ocurre un error, esta función envía el tipo de error, mensaje, archivo, número de línea donde ocurrió el error, además de una variable de contexto. Luego muestra una página de errores personalizada que contiene el número de línea del error ya sea en color púrpura (notificaciones) o rojo (advertencias), y detiene manualmente la ejecución del script. La figura 10-2 muestra los datos de salida.

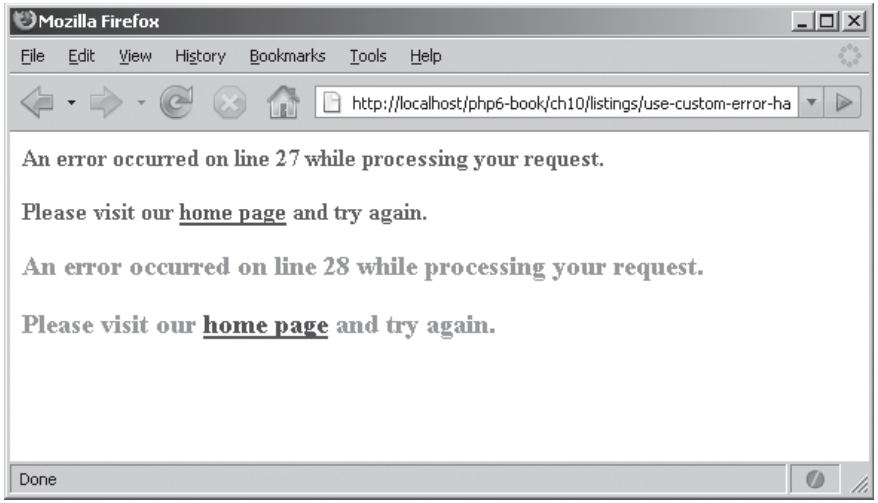


Figura 10-2 Página de error generada por un controlador de errores personalizado

Pregunta al experto

P: ¿Cómo restauro el mecanismo controlador de errores por defecto de PHP una vez que haya invocado `set_error_handler()`?

R: Existen dos maneras. El método más sencillo es utilizar la función `restore_error_handler()`. Esta función restaura el último controlador de errores en uso antes de la invocación de `set_error_handler()`; en casi todos los casos, este controlador será el de PHP por defecto.

Otra opción es hacer que el controlador de errores personalizado regrese un valor falso; esto obligaría al error a transferirse de regreso al controlador PHP por defecto para ser procesado de manera normal. Esta técnica es muy útil si necesitas transmitir primero errores a través de una función personalizada para preprocesamiento o registro, y verás un ejemplo en la sección titulada "Registrar errores".

Prueba esto 10-1 Generar una página de errores legible

El controlador de errores de PHP sólo muestra información sobre un error. Muchas veces, este mensaje de error aparece mientras se genera la página que contiene los datos de salida, con lo que se destruye el diseño de la página y se crean confusión y estrés innecesarios para el usuario (ver figura 10-3 para conocer un ejemplo de este tipo de página). Sin embargo, al reemplazar el controlador de errores estándar por una función personalizada, es posible resolver este problema con facilidad generando una página de errores legible y enviar al mismo tiempo los errores del script a una base de datos para su posterior revisión. El siguiente ejemplo te muestra cómo hacerlo.

Para comenzar, crea una nueva base de datos SQLite, y una tabla que almacene los errores, como se muestra aquí:

```
shell> sqlite app.db
sqlite> CREATE TABLE errores (
...> id INTEGER PRIMARY KEY,
...> date TEXT NOT NULL,
...> error TEXT NOT NULL,
...> script TEXT NOT NULL,
...> line TEXT NOT NULL,
...>);
```

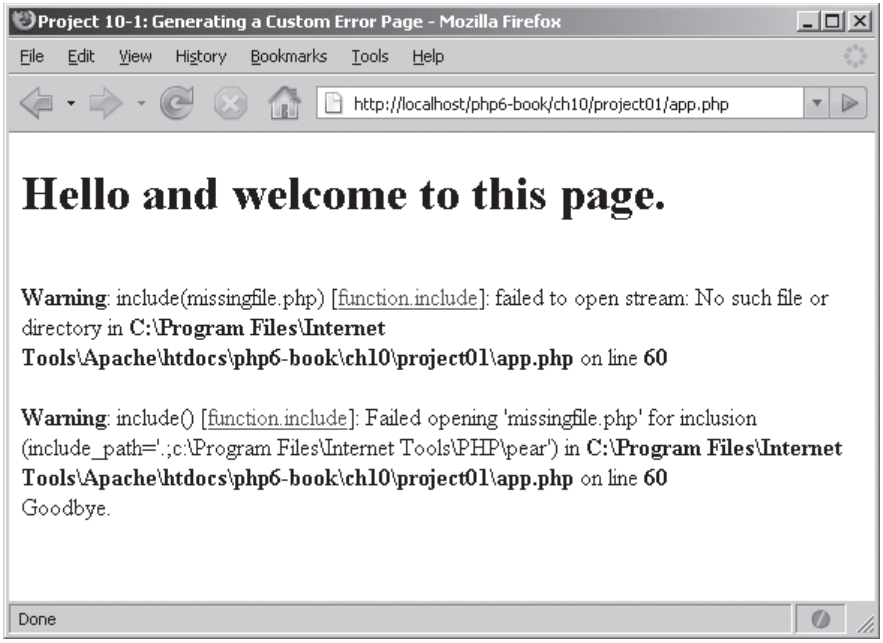


Figura 10-3 Datos de salida de errores entremezclada con el contenido del sitio

Después, define un controlador de errores personalizado que intercepte todos los errores del script y que los escriba en la tabla utilizando PDO, como en el siguiente script (*app.php*):

```
<?php
// reporta todos los errores
error_reporting(E_ALL);
// utiliza controlador personalizado
set_error_handler('miControlador');

// crea una región temporal de memoria (buffer)
ob_start();

// define un controlador personalizado
// que envíe los datos de error a la base de datos
// luego genera una página de error
function miControlador($type, $msg, $file, $line, $context) {
  // envía error a la base de datos
  $db = 'app.db';
  $pdo = new PDO("sqlite:$db");
  $msg = $pdo->quote($msg);
  $file = $pdo->quote($file);
  $line = $pdo->quote($line);
  $date = $pdo->quote(date('d-M-Y h:i:s', mktime()));
```

```
  $sql = "INSERT INTO errors (date, error, script, line) VALUES ($date,
$msg, $file, $line)";
  $pdo->exec($sql);

  // restablece y cierra el buffer
  // genera una nueva página de errores
  ob_end_clean();
  $errorPage = '
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "DTD/xhtml1-transitional.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
      <title>Página de Errores</title>
    </head>
    <body>
      <div style="border:solid 1px black; padding:10px; width:50%;
height:50%; margin:auto; top:0; bottom:0; right:0; left:0; position:
absolute">
        <h2>¡Perdón!</h2>
        Este script encontró un error interno y no es posible ejecutarlo.
        El error ya fue enviado y será rectificado a la brevedad.
        Hasta ese momento, por favor regrese a la página principal y
seleccione otra actividad.
      </div>
    </body>
  </html>';
  echo $errorPage;
  exit();
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 10-1: Generar una página de errores personalizada
</title>
  </head>
  <body>
<?php
// presenta algún texto de la página
echo "<h1>Hola y bienvenido a esta página.</h1>";

// genera advertencia (no se encuentra archivo)
include('archivofaltante.php');

// muestra alguna página de texto
echo "Adiós.";

// elimina el segmento de memoria reservado
ob_end_flush();
```

(*continúa*)

```
?>
  </body>
</html>
```

Además de los componentes con los que ya estás familiarizado (PDO y controladores de errores personalizados), este script también presenta un nuevo elemento: funciones de control para los datos de salida de PHP. Como su nombre lo sugiere, estas funciones proporcionan un medio para que los desarrolladores apliquen un alto grado de control sobre los datos de salida generados por un script PHP.

Las funciones de control sobre los datos de salida funcionan desviando todos los datos de salida generados por el script hacia una *parte reservada de la memoria para datos de salida*, en lugar de mandarlos directamente al explorador cliente. El contenido de esta memoria reservada permanece activo hasta que los datos se hacen visibles de manera explícita para el usuario y pueden eliminarse al restablecer la memoria reservada para ellos.

Es necesario aprender tres funciones principales de la API de PHP sobre el control de los datos de salida:

- La función `ob_start()` inicializa la memoria reservada para datos de salida y se prepara para interceptarlos de un script. No es necesario decir que esta función debe invocarse antes de que se genere cualquier dato de salida por parte del script.
- La función `ob_end_flush()` termina con la memoria reservada para los datos de salida y envía su contenido a un dispositivo de salida (por lo general el explorador del usuario).
- La función `ob_end_clean()` termina con la memoria reservada para los datos de salida y limpia su contenido.

Con toda esta teoría en mente, regresemos al script anterior para ver cómo la memoria reservada para los datos de salida te ayuda a producir una página de errores más legible. Una rápida mirada al script y verás que comienza por inicializar una nueva memoria reservada para los datos de salida con `ob_start()`, y un controlador de errores personalizado con `set_error_handler()`. Ahora, todo error generado por el script será almacenado en la memoria reservada hasta que los datos se liberen hacia el cliente utilizando la invocación a `ob_end_flush()`.

Ahora, considera lo que sucede cuando ocurre un error en el script. Primero, el controlador de errores personalizado interceptará este error, abrirá un controlador PDO direccionado hacia la base de datos SQLite creada en el paso anterior, y convertirá el error (notificación o advertencia) en una consulta SQL `INSERT`. Luego, esta consulta se utilizará para guardar el error en la base de datos utilizando el método `exec()` de PDO, junto con la fecha y hora exactas. Después, la invocación de `ob_end_clean()` transmitirá la memoria reservada para los datos de salida, enviará un mensaje de error personalizado al explorador del usuario y terminará la ejecución del script.

Figura 10-4 Página de errores legible

Como resultado de estas acciones, aunque existiera una página Web construida al vuelo cuando el error ocurre, nunca verá la luz del día, porque será descartada y reemplazada por la página de errores personalizada. Por otra parte, si el script se ejecuta sin errores, la invocación final a `ob_end_flush()` se encargará de enviar la página final completa al explorador.

Ahora, cuando ejecutes el script, en lugar de los datos de salida revueltos que se muestran en la figura 10-3, debes ver una página de errores legible. La figura 10-4 muestra lo que probablemente verás.

Mientras estás ahí, también mira la tabla de la base de datos SQLite. Debes encontrar los mensajes de error lanzados por el script, como se muestra a continuación:

```
1|13-Feb-2008 06:25:30|Undefined variable: myVar|/ch10/project01/app.
php|28
2|13-Feb-2008 06:25:30|include(archivofaltante.php): failed to open
steam: No such file or directory|/ch10/project01/app.php|31
3|13-Feb-2008 06:25:30| include(): Failed opening 'archivofaltante.php'
for inclusion (include_path='. ')|/ch10/project01/app.php|31
```

Una configuración como ésta facilita al administrador o desarrollador el mantenimiento de un registro permanente de los errores generados por el script y puede revisar el registro en cualquier momento para analizar o auditar errores; también asegura que la experiencia del usuario en el sitio sea consistente y sin complicaciones.

Pregunta al experto

P: ¿Qué sucede si el propio controlador de errores personalizado contiene errores?

R: Si el controlador de errores personalizado contiene errores, el mecanismo controlador de errores por defecto de PHP los manejará. De tal manera que, por ejemplo, si el código dentro del controlador personalizado de errores genera una advertencia, ésta será reportada dentro del nivel primario de PHP y será manejada por su controlador de errores por defecto.

Utilizar excepciones

Además de errores, PHP 5 también introduce un nuevo modelo de excepciones, similar al utilizado por otros lenguajes de programación como Java y Python. En este método basado en excepciones, el código del programa encerradote encierra en un bloque `try`, y las excepciones generadas por él son "atrapadas" y resueltas por uno o más bloques `catch`. Como es posible tener múltiples bloques `catch`, los desarrolladores pueden atrapar distintos tipos de excepciones y manejar cada una de ellas de manera diferente.

Para mostrar cómo funciona, examina el siguiente código, que intenta acceder a un elemento inexistente en una matriz utilizando un ArrayIterator:

```
<?php
// define una matriz
$ciudades = array(
  "Reino Unido" => "Londres",
  "Estados Unidos" => "Washington",
  "Francia" => "París",
  "India" => "Delhi",
);

// intenta accesar a un elemento inexistente en la matriz
// genera un OutOfBoundsException
// datos de salida: 'Excepción: Posición 10 buscada y fuera de rango'
try{

  $iterator = newArrayIterator($ciudades);
  $iterator->seek(10);
} catch (Exception $e){
 echo 'ERROR: Ocurrió una excepción en tu script.';
}
?>
```

Cuando PHP encuentra código encerrado dentro de un bloque `try`, primero intenta ejecutar ese código. Si se procesa sin que se generen excepciones, el control se transfiere a las líneas que siguen al bloque `try-catch`. Sin embargo, si se genera una excepción mientras se

ejecuta el código dentro del bloque `try` (como sucede en el ejemplo anterior), PHP detiene la ejecución del bloque en ese punto y comienza a verificar cada bloque `catch` para ver si hay un controlador para la excepción. Si se encuentra un controlador, el código dentro del bloque `catch` apropiado se ejecuta y luego se ejecutan las líneas que siguen al bloque `try`; en caso contrario, se genera un mensaje de error fatal y se detiene la ejecución del script en el punto donde se localiza el error.

Cada objeto Exception incluye información adicional que puede utilizarse para depurar la fuente del error. Es posible acceder a esta información mediante los métodos integrados en el objeto Exception e incluyen un mensaje de error descriptivo, un código de error, el nombre del archivo y el número de línea donde se encontró el error, así como un seguimiento de las invocaciones de la función que llevaron al error. La tabla 10-3 presenta una lista de estos métodos.

El siguiente código modificado a partir del ejemplo anterior muestra estos métodos en uso:

```
<?php
// define una matriz
$ciudades = array(
  "Reino Unido" => "Londres",
  "Estados Unidos" => "Washington",
  "Francia" => "París",
  "India" => "Delhi"
);

// intenta accesar a un elemento inexistente en la matriz
// genera un OutOfBoundsException
try {

  $iterator = new ArrayIterator($ciudades);
  $iterator->seek(10);
```

Nombre del método	Lo que hace
`getMessage()`	Regresa un mensaje describiendo lo que salió mal
`getCode()`	Regresa un código de error numérico
`getFile()`	Regresa la ruta de acceso en disco y el nombre del script que generó la excepción
`getLine()`	Regresa el número de línea que generó la excepción
`getTrace()`	Regresa el seguimiento de las invocaciones que llevaron al error, como una matriz
`getTraceAsString()`	Regresa el seguimiento de las invocaciones que llevaron al error, como una cadena de texto

Tabla 10-3 Métodos del objeto de excepción PHP

```
} catch (Exception $e) {
  echo "ERROR: ¡Algo salió mal!\n";
  echo "Error message: " . $e->getMessage() . "\n";
  echo "Error code: " . $e->getCode() . "\n";
  echo "File name: " . $e->getFile() . "\n";
  echo "Line: " . $e->getLine() . "\n";
  echo "Seguimiento: " . $e->getTraceAsString() . "\n";
}
?>
```

TIP

Es posible manejar diferentes tipos de excepciones de manera diferente, creando múltiples bloques `catch` y asignándoles diferentes acciones a cada uno. Verás un ejemplo de esto un poco más adelante en este mismo capítulo.

Ahora, si lo piensas, puedes estar tentado a creer que las excepciones son vino viejo en una botella nueva. Después de todo, parece que las capacidades descritas pueden replicarse fácilmente utilizando un controlador de errores personalizado, como se describió en la sección anterior. Sin embargo, en la práctica, este método basado en excepciones es mucho más sofisticado de lo que parece, porque ofrece los siguientes beneficios adicionales:

- En el modelo tradicional es necesario revisar el valor regresado de cada función invocada para verificar si ocurrió un error y realizar la acción correctiva. Esto puede producir código innecesariamente complicado, además de un anidamiento profundo de bloques de código. En el modelo basado en excepciones, puede utilizarse un solo bloque `catch` para atrapar cualquier error que ocurra mientras se procesa el bloque de código. Esto elimina la necesidad de múltiples cascadas de verificaciones para buscar errores, y produce un código más sencillo y fácil de leer.
- El modelo tradicional es prescriptivo por naturaleza: requiere que el desarrollador piense en todos los errores que pueden ocurrir, y que escriba el código que maneje cada una de estas posibilidades. En contraste, el método basado en excepciones es más flexible. Un controlador genérico de excepciones funciona como una red de seguridad, atrapando y manejando incluso los errores para los cuales no fue escrito ningún controlador específico. Esto ayuda a producir una aplicación más robusta y resistente a situaciones imprevistas.
- Dado que el modelo de excepciones utiliza un método orientado a objetos, los desarrolladores pueden utilizar conceptos de programación orientada a objetos como la herencia y extensibilidad para hacer subclases a partir del objeto Exception base y crear así diferentes objetos de excepción para múltiples tipos de excepciones. Esto hace posible distinguir entre diferentes tipos de errores, manejando cada uno de ellos de manera particular.
- El método basado en excepciones obliga a los desarrolladores a tomar decisiones cruciales sobre el manejo de diferentes tipos de errores. Al contrario del modelo tradicional, donde

los desarrolladores pueden omitir fácilmente (por accidente o designio) las pruebas de verificación para los valores que regresa una función, las excepciones no son tan fáciles de ignorar. Al requerir que los desarrolladores creen y alimenten bloques `catch`, el modelo de excepciones los obliga a pensar en las causas y consecuencias de los errores y, a fin de cuentas, da como resultado un mejor diseño y una implementación más robusta.

¿El único inconveniente? Las excepciones se introdujeron hasta PHP 5 y, como resultado, son generadas de forma nativa sólo en las más recientes extensiones del lenguaje, como SimpleXML, objetos de datos PHP (PDO), objetos de servicio de datos (SDO, Service Data Objects) y la biblioteca estándar PHP (SPL, Standard PHP Library). Para el resto de las funciones de PHP es necesario verificar el valor que regresa la función para buscar errores y convertirlos manualmente en excepciones.

Crear, o *levantar*, manualmente excepciones de esta manera es una tarea para la declaración PHP `throw`. Ésta necesita transmitir un mensaje de error descriptivo y un código de error opcional, luego de lo cual genera un objeto Exception utilizando los parámetros y poniendo a la disposición del controlador de excepciones el mensaje y el código enviados. El proceso se ejemplifica en el siguiente código:

```
<?php
// establece el nombre del archivo
// intenta copiar y eliminar el archivo
$archivo = 'ejemplo.txt';
try {

  if (!file_exists($archivo)) {
    throw new Exception("Archivo '$archivo' no fue localizado.");
  }
  if (file_exists("$archivo.new")) {
    throw new Exception("Archivo destino '$archivo.new' ya existe.");
  }
  if (!copy($archivo, "$archivo.new")) {
    throw new Exception("Archivo '$archivo' no pudo ser copiado.");
  }
  if (!unlink ($archivo)) {
    throw new Exception("Archivo '$archivo' no pudo ser borrado.");
  }

} catch (Exception $e) {
  echo '¡Perdón! Algo malo ha ocurrido en la línea ' . $e->getLine() . ': ' .
$e->getMessage ();
  exit();
}
echo 'ÉXITO: la operación con el archivo fue correcta.';
?>
```

Aquí, dependiendo del resultado de varias operaciones, el script lanza una nueva excepción manualmente. Un controlador estándar de excepciones atrapa más adelante la excepción, y se extrae la información específica del error y se muestra al usuario.

Utilizar excepciones personalizadas

Un método más avanzado consiste en crear subclases a partir del objeto genérico Exception y crear objetos específicos para cada posible error. Este método es útil cuando necesitas tratar diferentes tipos de excepciones de manera única, porque te permite utilizar bloques `catch` separados (y separar el código controlador) para cada tipo de excepción. He aquí una versión modificada del ejemplo anterior, que ilustra este método:

```
<?php
// subclases de Exception
class ExcepArchivoPerdido extends Exception {}
class ExcepArchivoDuplicado extends Exception {}
class ExcepArchivoIO extends Exception {}

// establece el nombre del archivo
// intenta copiar y eliminar el archivo
$archivo = 'ejemplo.txt';
try {

  if (!file_exists($archivo)) {
    throw new ExcepArchivoPerdido($archivo);
  }
  if (file_exists("$archivo.new")) {
    throw new ExcepArchivoDuplicado ("$archivo.new");
  }
  if (!copy($archivo, "$archivo.new")) {
    throw new ExcepArchivoIO("$archivo.new");
  }
  if (!unlink ($archivo)) {
    throw new ExcepArchivoIO($archivo);
  }

} catch (ExcepArchivoPerdido $e) {
  echo 'ERROR: No se encontró el archivo \'' . $e->getMessage() . ' \";
  exit();
} catch (ExcepArchivoDuplicado $e) {
  echo 'ERROR: El archivo destino \'' . $e->getMessage() . ' \' ya
existe';
exit();
} catch (ExcepArchivoIO $e) {
  echo 'ERROR: No fue posible realizar operación de entrada/salida en el
archivo \'' . $e->getMessage() . '\'';
  exit();
} catch (Exception $e) {
```

```
  echo '¡Oops! Algo malo ha ocurrido en la línea ' . $e->getLine() . ': '
. $e->getMessage();
  exit();
}
echo 'ÉXITO: la operación con el archivo fue correcta.';
?>
```

Este script extiende la clase base Exception para crear tres nuevos tipos de excepciones, cada uno de los cuales representa un posible error. Un bloque `catch` individual para cada excepción ahora posibilita la personalización del tratamiento en cada caso específico. El último bloque `catch` es un controlador genérico "atrapatodo": las excepciones que no son controladas por alguno de los bloques específicos superiores caerán en este controlador que lidiará con ellos.

Pregunta al experto

P: He visto que las excepciones que no son atrapadas generan un error fatal y hacen que el script termine abruptamente. ¿Puedo cambiar esto?

R: Sí y no. PHP ofrece la función `set_exception_handler()`, que te permite reemplazar el controlador de excepciones por defecto de PHP con tu propio código personalizado, de manera muy similar a lo que hace `set_error_handler()`. Sin embargo, aquí hay un elemento muy importante a considerar. Como has visto, el controlador de excepciones por defecto de PHP muestra una notificación y luego da por terminada la excepción del script. Utilizar un controlador de excepciones personalizado tiene un control limitado sobre este comportamiento: puedes cambiar el modo y la apariencia de la notificación que aparece, pero no puedes hacer que el script continúe ejecutándose más allá del punto donde se generó la excepción.

La moraleja de la historia es que las excepciones que no son atrapadas *siempre* darán como resultado la terminación abrupta del script. Por eso es una buena idea incluir siempre un bloque `catch` genérico en tu código de manejo de excepciones para atrapar *todas* las excepciones que genera el script, independientemente de su tipo.

Prueba esto 10-2 Validar datos de entrada en un formulario

Ahora que ya tienes un conocimiento básico sobre el funcionamiento de excepciones, además de lanzar y atrapar excepciones personalizadas, apliquemos estos conocimientos en un pequeño proyecto que pone en práctica estas herramientas. El siguiente ejemplo permite que el usuario haga una solicitud para adquirir obras de arte, seleccionando un artista, el medio y el rango de precio. Esta orden se valida, se forma en un mensaje de correo electrónico y se

(*continúa*)

entrega al servidor de correo para su envío. Los errores en el proceso se manejan mediante controladores personalizados, produciendo un código que es fácil de leer y mantener.

He aquí el código (*arte.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 10-2: Validar datos de entrada de un formulario</title>
    <style type="text/css">
    div.correcto {
      width:200px;
      padding:5px;
      border:solid 1px black;
      color:green;
    }
    div.error {
      width:400px;
      padding:5px;
      border:solid 1px black;
      color:red;
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 10-2: Validar datos de entrada de un formulario</h2>
    <h3 style="background-color: silver">Compra de Arte Fino</h3>
<?php
    // si el formulario no ha sido enviado
    // muestra el formulario
    if (!isset($_POST['submit''])) {
?>
    <form method="post" action="arte.php">
      Artista: <br />
      <select name="artista">
        <option value="">--Seleccione uno--</option>
        <option value="Picasso">Picasso</option>
        <option value="Van Gogh">Van Gogh</option>
        <option value="Chagall">Chagall</option>
        <option value="Degas">Degas</option>
        <option value="Monet">Monet</option>
        <option value="Matisse">Matisse</option>
      </select>

      <p>

      Medio: <br />
      <select name="medio">
        <option value="">--Seleccione uno--</option>
        <option value="óleo">Óleo</option>
```

```
        <option value="acuarela">Acuarela</option>
        <option value="tinta»"Tinta</option>
      </select>

      <p>

      Rango de precio: <br />
        <input type="text" size="4" name="min" /> y <input type="text"
size="5" name="max" />
      <p>

      Dirección de correo electrónico: <br />
      <input type="text" size="25" name="email" />

      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
<?php
    // si la forma ya fue enviada
    // procesa los datos de entrada
    } else {
      error_reporting(E_ERROR);

      // define clases de excepción
      class ExcepDatosEntrada extends Exception {}
      class ExcepLogica extends Exception {}
      class ExcepCorreo extends Exception {}

      // obtiene los datos de entrada del formulario
      $artista = $_POST['artista'];
      $medio = $_POST['medio'];
      $min = $_POST['min'];
      $max = $_POST['max'];
      $email = $_POST['email'];

      try {

        // valida datos de entrada del formulario
        if (empty($artista)) {
          throw new ExcepDatosEntrada('Artista');
        }
        if (empty($medio)) {
          throw new ExcepDatosEntrada('Medio');
        }
        if (empty($email)) {
          throw new ExcepDatosEntrada('Dirección de Correo');
        }
       if (empty($min) || empty ($max) || (int)$min <= 0 || (int)$max <=0) {
          throw new ExcepDatosEntrada('Precio');
        }
        if($max < $min) {
```

(*continúa*)

```
          throw new ExcepLogica('El precio máximo no puede ser menor que
el precio mínimo');
        }
        // envía correo electrónico con elección
        $subject = 'Orden de compra';
        $to = $email;
        $from = $email;
        $body = "
          DETALLES DE LA ORDEN: \r\n\r\n
          Artista: $artista \r\n
          Medio: $medio \r\n
          Precio: entre $min y $max \r\n
        ";
        if (!mail($to, $subject, $body, "From:$from")) {
          throw new ExcepCorreo();
        }

        // presenta mensaje de éxito
       echo '<div class="correcto">ÉXITO: ¡La orden fue procesada!</div>';

      } catch (ExcepDatosEntrada $e) {
        echo '<div class="error">ERROR: Por favor proporcione datos
válidos para el campo marcado \'' . $e->getMessage() . '\'</div>';
        exit();
      } catch (ExcepLogica $e) {
        echo '<div class="error"> ERROR: '. $e->getMessage() . ' </div>';
        exit();
      } catch (ExcepCorreo $e) {
        echo '<div class="error"> ERROR: Imposible enviar correo
electrónico</div>’;
        file_put_contents('error.log', '[' . date("d-M-Y h:m:s",
mktime()) . '] Error de envío de correo a: ' . $e->getMessage() . "\n",
FILE_APPEND);
        exit();
      } catch (Exception $e) {
        echo '<div class="error">' . $e->getMessage() . ' en línea ' .
$e->getLine() . '</div>';
        exit();
      }
    }
?>
  </body>
</html>
```

La primera parte del script simplemente genera un formulario Web para que el usuario haga su elección, como se muestra en la figura 10-5.

Una vez que el formulario es enviado, el script desactiva el reporte de errores para todos los errores fatales y define tres nuevos tipos de excepciones: ExcepDatosEntrada para los datos de entrada en el formulario, ExcepLogica para los errores lógicos en los datos de entrada

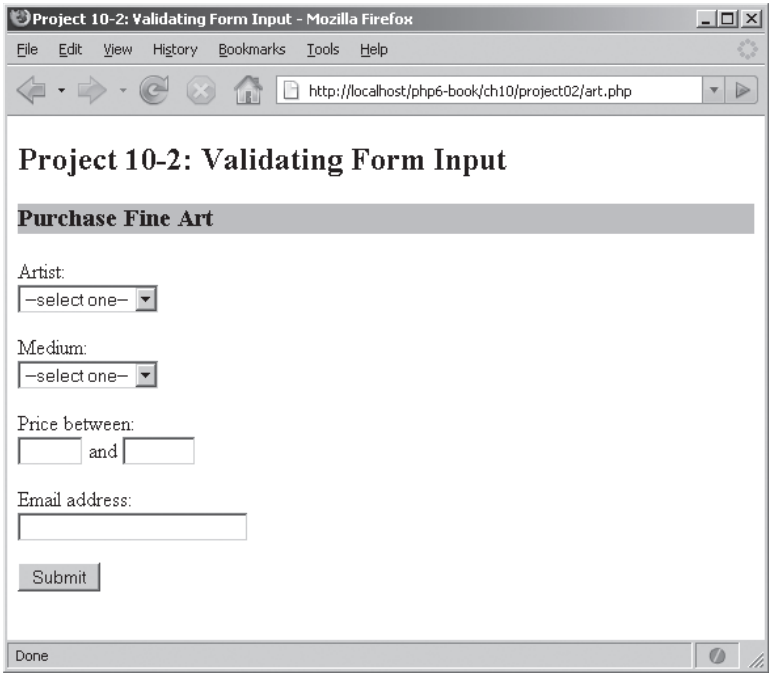


Figura 10-5 Formulario Web para que el usuario haga una orden de compra de arte fino

y ExcepCorreo para los errores en el envío del correo electrónico. Estas excepciones son simples extensiones del objeto genérico Exception, sin más adornos.

A continuación, se utiliza un bloque `try` para encapsular el proceso lógico del script. Primero, se verifica la consistencia de los campos del formulario; los errores en los campos de datos generan una excepción ExcepDatosEntrada o una ExcepLogica. Una vez que los datos han sido validados, se forman como un mensaje de correo electrónico, que es transmitido utilizando la función `mail()` de PHP. Si la función `mail()` regresa un valor falso, indica que el mensaje no fue enviado y surge una excepción ExcepCorreo; en caso contrario, aparece un mensaje de éxito.

Las excepciones generadas por el proceso lógico no son ignoradas; en cambio, cuatro bloques `catch` (uno para cada una de las excepciones definidas y uno genérico) atrapan estas excepciones, presentan al usuario la notificación apropiada del error y detienen el proceso del script. Cada una de estas excepciones es manejada de modo un poco diferente, para mostrar la manera en que se personaliza cada rutina de excepción: la excepción ExcepCorreo es enviada a un archivo con la fecha, hora y la dirección de correo electrónico del destinatario; por su parte, las excepciones ExcepDatosEntrada y ExcepLogica producen diferentes mensajes de error.

(*continúa*)

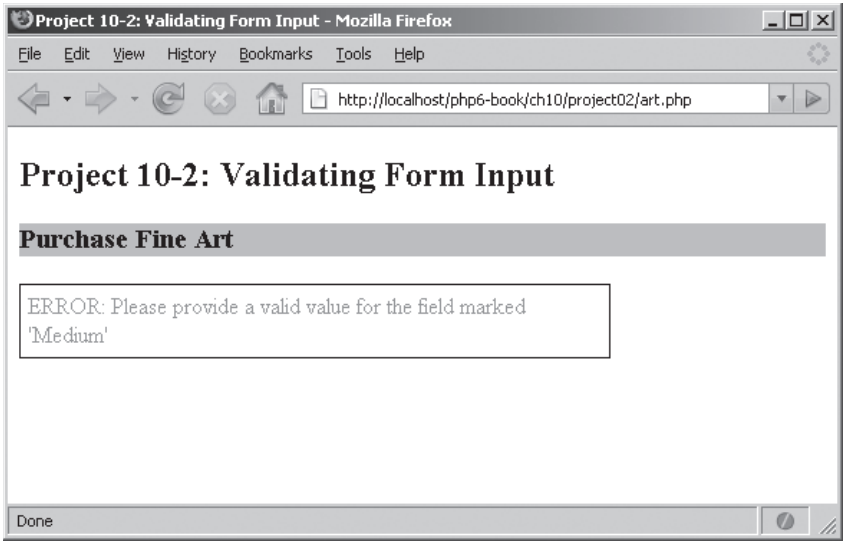


Figura 10-6 El resultado de atrapar una excepción ExcepDatosEntrada cuando se procesa el formulario Web

La figura 10-6 presenta el resultado que genera una excepción ExcepDatosEntrada.

La figura 10-7 presenta el resultado cuando falla la transmisión del correo electrónico y que genera una excepción ExcepCorreo.

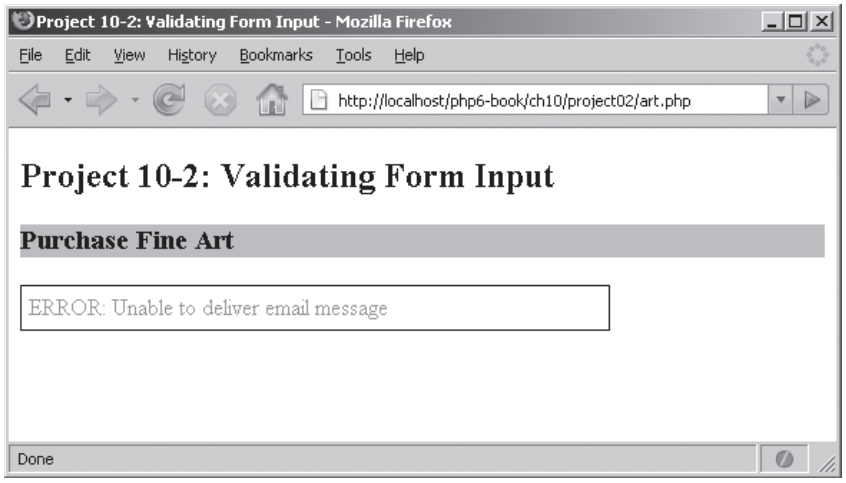


Figura 10-7 El resultado de atrapar una excepción ExcepCorreo cuando se procesa el formulario Web

Registro de errores

Además de atrapar y manejar los errores en tiempo de ejecución, por lo regular también es una buena idea mantener un registro semipermanente de los errores que generó la aplicación, con el fin de inspeccionarlos y depurarlos. Ahí es donde entra en acción la función `error_log()` de PHP: te permite enviar los errores a un archivo de disco o a una dirección de correo electrónico en el momento en que ocurren.

La función `error_log()` necesita un mínimo de dos argumentos: el mensaje de error que debe enviarse y un valor numérico entero que indique el destino. Este valor entero puede ser 0 (el archivo de registros del sistema), 1 (una dirección de correo electrónico) o 3 (un archivo de disco). Para una dirección de correo electrónico o un archivo de disco se debe especificar la ruta de acceso correspondiente como tercer argumento para `error_log()`.

He aquí un ejemplo que muestra su funcionamiento:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "musica");
if (mysqli_connect_error()) {
  error_log("ERROR: No se estableció la conexión. " . mysqli_connect_
error() . "\n", 3, 'debug.log');
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}

// intenta ejecutar query
// itera sobre colección de resultados
// muestra cada registro y sus campos
// datos de salida: "1:Aerosmith \n 2:Abba \n ..."
$sql = "SELECT artista_id, artista_nombre FROM artistas";
if ($result = $mysql ->query($sql)) {
  if ($result->num_rows > 0) {
    while($row = $result->fetch_array()) {
      echo $row['artista_id'] . ":" . $row['artista_nombre'] . "\n";
    }
    $result->close();
  } else {
    echo "No se encontró ningún registro que coincida con su búsqueda.";
  }
} else {
  error_log("ERROR: No fue posible ejecutar $sql. " . $mysqli->error . "\
n", 3, 'debug.log');
  echo "ERROR: No fue posible ejecutar $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Suponiendo que se utilice un nombre de usuario o una contraseña incorrecta, el script enviará el siguiente mensaje al archivo *debug.log*:

```
ERROR: No se estableció la conexión. Acceso denegado para usuario
'usuario'@'localhost' (uso de contraseña: YES)
```

Es fácil combinar esta función de creación de reportes de error con el manejador de errores personalizado para asegurarse de que los errores del script queden almacenados en un archivo. He aquí un ejemplo que lo demuestra:

```
<?php
// controlador de errores personalizado
function miControlador($type, $msg, $file, $line, $context) {
    error_log("[" . date("d-M-Y h:m:s", mktime()) . "] $msg en la línea
$line de $file\n", 3, 'depuración.log');
    return false;
}

// reporta todos los errores
error_reporting(E_ALL);

// establece controlador personalizado
set_error_handler("miControlador");

// muestra E_NOTICE (variable indefinida)
echo $unaVariable;

// muestra E_WARNING (archivo perdido)
include("cualquier.php");
?>
```

En este script, todas las notificaciones y advertencias generadas por el código serán enviadas primero en automático al archivo *debug.log* con un sello cronológico. Después de ello, el controlador personalizado regresa un valor falso; esto se hace deliberadamente para que el control regrese al controlador de errores por defecto de PHP y lo maneje de la manera tradicional.

Depurar errores

Con aplicaciones más complejas se hace necesario encapsular las tareas más utilizadas en componentes independientes (funciones y clases) e importarlas conforme sea necesario a tus scripts PHP. Así como este enfoque mejora sin lugar a dudas el mantenimiento de tu código, también puede convertirse en una carga cuando las cosas no funcionan como es debido. Por ejemplo, es posible que un error en un componente afecte la función correcta de otros componentes, y rastrear ese error hasta su lugar de origen es una experiencia larga y frustrante.

Existen varias herramientas para reducir la complejidad y laboriosidad de este proceso. La primera es proporcionada por PHP mismo, con la función `debug_print_backtrace()`. Esta función presenta una lista de todas las invocaciones de función que conducen a un error en particular, con lo que puedes identificar rápidamente la fuente del error. Para mostrarla en acción, revisa el siguiente script, que contiene errores deliberados:

```
<?php
// controlador de errores personalizado
// presenta un rastreo de invocaciones en caso de que ocurra un error
function miControlador($type, $msg) {
  debug_print_backtrace();
  exit();
}

class Hoja {
  public function_construct($num1, $num2) {
    $ex = new Ejemplo($num1, $num2);
  }
}

class Ejemplo {
  public function_construct($a, $b) {
    $this->a = $a+1;
    $this->b = $b-1;
    $this->run();
  }

  public function run() {
    return $this->a/$this->b;
  }
}

set_error_handler('miControlador');
$hoja = new Hoja(10,1);
echo 'Ejecución del código exitosa';
?>
```

Aquí, el script generará una advertencia de "división entre cero", no porque se trate de un error en la definición de la clase Hoja, sino porque el error es generado por el método `run()`, después de la invocación. La figura 10-8 muestra el mensaje de error que uno normalmente vería en este caso.

Depurar un error así en la práctica puede ser muy complicado, en especial cuando las definiciones de función y clase están contenidas en archivos separados. Sin embargo, como el controlador de errores en el script presenta un rastreo automático cuando ocurre el error, no es difícil identificar la función que lo causa. La figura 10-9 muestra el resultado modificado con rastreo automático.

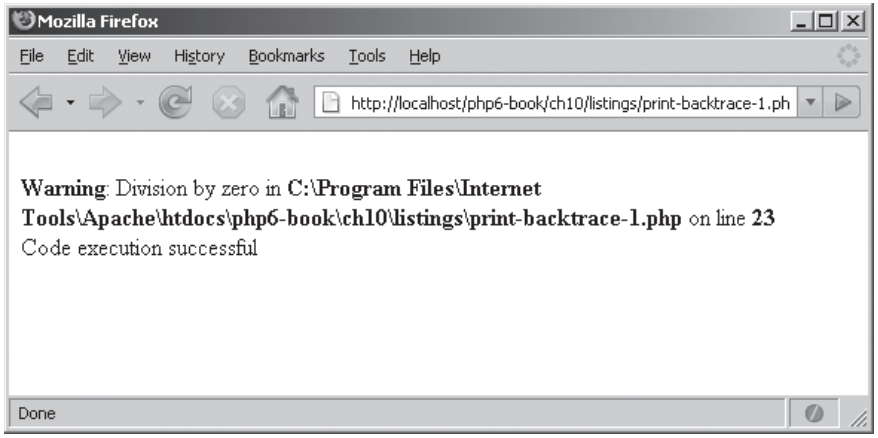


Figura 10-8 Página PHP de errores sin la información adicional de depuración

TIP

Para enriquecer el motor central de PHP con el fin de que añada automáticamente el rastreo de errores en todos los mensajes de error, intenta agregar la extensión gratuita Xdebug a tu configuración de PHP. Para más información e instrucciones de instalación visita www.xdebug.com/.

Un método aún más avanzado incluye el uso de la clase Debug de PHP, que se puede obtener gratuitamente en www.php-depuración.com/. Esta clase proporciona un conjunto de herramientas muy útil para rastrear la ejecución del script, observa y desecha variables, calcula tiempos de ejecución y realiza otras tareas de depuración comunes.

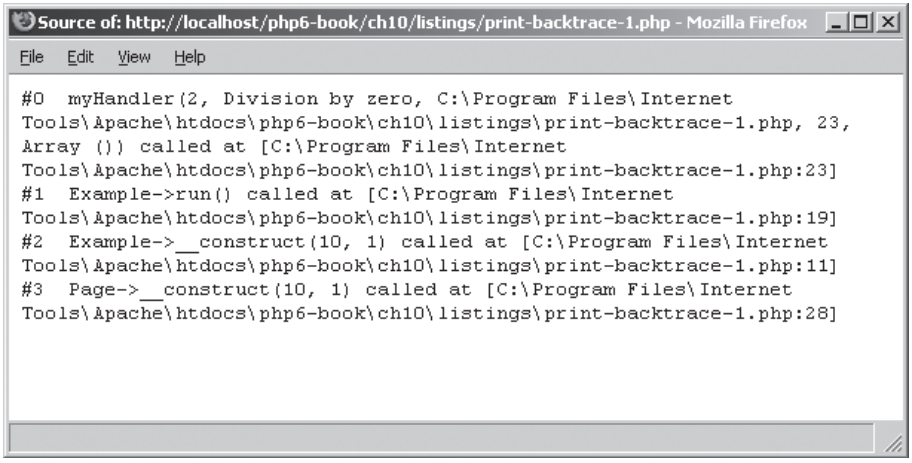


Figura 10-9 Página PHP de errores enriquecida con rastreo automático

Para utilizar esta clase, descárgala y coloca sus archivos en el directorio de tu aplicación. Después, revisa el siguiente código para utilizarla, como sigue:

```
<?php
// controlador de errores personalizado
// presenta un rastreo de invocaciones en caso de que ocurra un error
function miControlador($type, $msg, $file, $line, $context) {
  global $debug;
  $debug->error($msg);
  echo '
  <!DOCTYPE html PUBLIC "-// W3C//DTD XHTML 1.0 Transitional//EN"
     "DTD/xhtml1-transitional.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
      <title>Página de Errores</title>
      <link rel="stylesheet" type="text/css" href="./PHP_Debug-1.0.0/css/
html_table.css" />
    </head>
    <body>
  ';
  $debug->display();
  echo '
    </body>
  </html>
  ';
  exit();
}

// añade ganchos de depuración a las clases y funciones
class Hoja {
  public function _construct($num1, $num2) {
    global $debug;
    $debug->add('Entering Page::_construct');
    $debug->dump($this);
    $debug->dump(func_get_args(), 'method args');
    $ex = new Ejemplo($num1, $num2);
  }
}

class Ejemplo {
  public function_construct($a, $b) {
    global $debug;
    $debug->add('Entering Hoja::_construct');
    $debug->dump($this);
    $debug->dump(func_get_args(), 'method args');
    $this->a = $a+1;
    $this->b = $b-1;
```

```
    $this->run();
  }

  public function run() {
    global $debug;
    $debug->add('Entering Hoja::run');
    $debug->dump($this);
    $debug->dump(func_get_args(), 'method args');
    return $this->a/$this->b;
  }
}

// configura y lee archivos PHP_Debug
ini_set('incluir_ruta','./PHP_Debug-1.0.0/;');
include 'PHP_Debug-1.0.0/PHP/Debug.php;
include 'PHP_Debug-1.0.0/PHP/Debug/Renderer/HTML/TableConfig.php';
$options = array (
    'render_type'          => 'HTML',
    'render_mode'          => 'Table',
    'replace_errorhandler' => false,
);
$debug = new PHP_Debug($options);
set_error_handler('miControlador');

// comienza a ejecutar el código
$hoja = new Hoja(10,1);
echo 'Código ejecutado con éxito';
?>
```

La clase PHP_Debug proporciona un objeto con varias herramientas para ayudar a los desarrolladores a depurar sus scripts. Por ejemplo, el método `add()` del objeto puede utilizarse para rastrear la ejecución del script generando mensajes en ciertos puntos definidos por el usuario dentro del script, como pueden ser el comienzo de una función o de un bucle. De manera similar, el método `dump()` puede utilizarse para enviar el valor de una variable en determinado instante, mientras que el método `display()` puede usarse para enviar toda la depuración al dispositivo de salida. En la lista precedente, este método `display()` suele utilizarse como controlador de errores para que, en caso de que ocurra uno, genere de manera automática un rastreo nítido de todas las invocaciones que guiaron al error.

La figura 10-10 muestra los datos de salida del ejemplo anterior.

Como se aprecia en la figura 10-10, PHP_Debug puede proporcionar un medio más sofisticado para registrar todas las invocaciones de función que llevaron al error, en comparación con la función `depuración_print_backtrace()` de PHP.

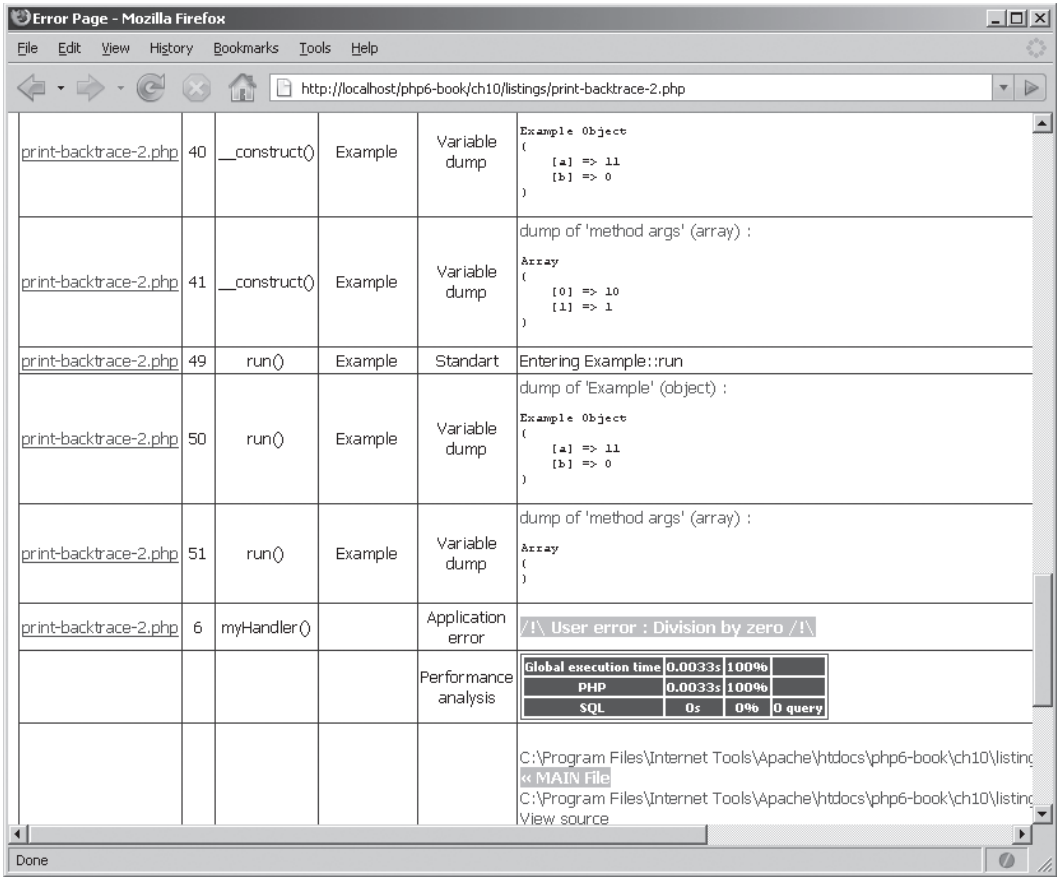


Figura 10-10 Rastreo generado por el paquete PHP_Debug

Resumen

Al finalizar este capítulo, debes tener una mejor idea de cómo funcionan las herramientas de depuración de PHP, y de la manera de personalizarlas para que se acoplen a las necesidades de una de tus aplicaciones en particular. Este capítulo abordó dos modelos: el antiguo, basado en errores, y el nuevo, basado en excepciones introducido en PHP 5. La facilidad de uso y la extensibilidad son puntos a favor del nuevo modelo en comparación con su antecesor, además de técnicas menos primitivas; sin embargo, como no cuenta con soporte completo en las bibliotecas principales de la aplicación, la mayoría de los programas desarrollados demandan una combinación juiciosa de ambos enfoques.

Además de enseñarte todo lo relacionado con los errores y las excepciones, este capítulo también te presentó algunas funciones de utilidad para el almacenamiento de errores y la depuración de los mismos, y presentó una herramienta de terceros, el paquete PHP_Debug, para obtener un rastreo efectivo cuando ocurre un error. Finalmente, dos proyectos (una base de datos para almacenar mensajes de error y un validador para datos de entrada en un formulario Web) pusieron en práctica la teoría enseñada en el capítulo.

Para conocer más acerca de los temas abordados en este capítulo, visita las siguientes ligas:

- Excepciones, en **www.php.net/exceptions**
- Funciones para el manejo de errores, en **www.php.net/errorfunc**
- La página oficial de PHP_Debug, en **www.php-debug.com**

Autoexamen Capítulo 10

1. Menciona dos beneficios de utilizar el modelo basado en excepciones.
2. ¿Qué tipos de errores de script no pueden atraparse con un controlador personalizado de errores?
3. ¿Qué es un espacio de memoria reservado? ¿Por qué es útil? Proporciona un ejemplo práctico que demuestre su utilidad.
4. ¿Cuál controlador de errores se activará después de la última línea de cada uno de los siguientes bloques de código al ejecutarse?

```
<?php
set_error_handler('controladorA');
restore_error_handler();
?>

<?php
set_error_handler('handlerA');
set_error_handler('handlerB');
set_error_handler('handlerA');
restore_error_handler();
restore_error_handler();
?>
```

5. Utilizando un controlador de errores personalizado, muestra cómo convertir notificaciones y advertencias PHP en excepciones.
6. Escribe un programa que automáticamente envíe al correo electrónico del administrador las excepciones no capturadas en un script.

Capítulo 11

Seguridad con PHP

Habilidades y conceptos clave

- Familiarizarse con aspectos de seguridad en las aplicaciones PHP
- Prevenir tipos comunes de ataques limpiando los datos de entrada y de salida
- Incrementar la seguridad de tus archivos de aplicación, bases de datos y sesiones
- Aprender a validar cadenas de texto, números y fechas
- Comenzar a utilizar expresiones regulares para la validación de datos de entrada
- Aprender sobre la configuración de seguridad de PHP

En los capítulos anteriores aprendiste mucho sobre traer y utilizar datos de fuentes externas en tus aplicaciones PHP. Has procesado con éxito datos enviados mediante un formulario Web, has conectado tus páginas Web a bases de datos para generar páginas dinámicas y has recuperado información codificada en XML.

Ahora bien, la interacción con todas estas fuentes de datos externas puede hacer que tus aplicaciones PHP sean más relevantes y útiles, pero hay algunos riesgos que debes controlar. Las aplicaciones PHP inseguras son vulnerables a los ataques de usuarios maliciosos, y los datos de entrada mal validados pueden causar cálculos erróneos y reportes equivocados. Este tipo de vulnerabilidades pueden ser causa de corrupciones significativas y pérdida de datos, lo mismo que provocar vergüenza a los desarrolladores orgullosos.

Ahí es donde entra este capítulo. En las siguientes páginas se te presentará una introducción a varias técnicas que pueden utilizarse para validar los datos de entrada para tu aplicación, haciéndola más robusta y segura. Además, se ofrecerán ejemplos prácticos de higiene en los datos de entrada y salida, se abordarán algunas de las más importantes variables de seguridad PHP en el archivo de configuración y se te sugerirán varios recursos en línea donde puedes aprender más sobre la seguridad para aplicaciones PHP.

Higiene en los datos de entrada y salida

Como desarrollador de aplicaciones Web, tendrás que aprender a vivir con un desafortunado hecho: siempre habrá allá afuera gente que se divierte encontrando agujeros en tu código y explotándolos para propósitos malignos.

Casi siempre los ataques consisten en enviar a tu aplicación datos de entrada inteligentemente disfrazados que la “engañan” para que haga algo que realmente no debe hacer. Un

ejemplo común de este tipo son los "ataques de inyección SQL", donde el atacante manipula desde afuera tu base de datos con una consulta SQL incrustada dentro de los datos de un formulario. Por ello, una de las tareas más importantes que debe realizar un desarrollador, antes de utilizar cualquier dato de salida enviado por un usuario, es higienizar los datos eliminando cualquier carácter especial o símbolo que pueda tener.

De manera similar, si tu aplicación va a utilizar datos provenientes de fuentes externas, siempre es necesario "limpiar" estos datos antes de mostrarlos al usuario. Una falla en este aspecto puede provocar que los atacantes incrusten contenido malicioso en tus páginas Web sin que lo notes. Un ejemplo común de este tipo es el "ataque de script de sitios cruzados", donde el atacante puede obtener acceso a datos sensibles de los usuarios al implantar de manera maliciosa código JavaScript o HTML como formulario en tus páginas Web. Con esto en mente, es siempre esencial higienizar por rutina los datos de salida antes de ponerlos a disposición de tus usuarios.

Por fortuna, PHP incluye varias funciones para ayudar a los desarrolladores en las tareas de saneamiento de datos de entrada y salida. He aquí una breve lista:

- La función `addslashes()` elimina caracteres especiales (como comillas dobles y diagonales invertidas) de los datos de entrada, de manera que sea seguro insertarlos en la base de datos. Como opción, utiliza el método `real_escape_string()` de MySQLi para sanear los datos de entrada antes de insertarlos en una base de datos MySQL, o bien la función `sqlite_escape_string()` para hacer lo mismo en bases de datos SQLite.
- La función `strip_tags()` permite a los desarrolladores eliminar todas las etiquetas HTML y PHP de una cadena de texto, regresando así sólo contenido ASCII. Esto puede ser útil para eliminar código potencialmente malicioso tanto de los datos de entrada de usuarios como de fuentes remotas.
- La función `htmlentities()` se encarga de reemplazar caracteres especiales como `"`, `&`, `<` y `>` por su correspondiente valor HTML. Al convertir estos caracteres especiales y evitar que sean interpretados como código HTML por el cliente, esta función es muy útil para "desarmar" los datos e incapacitarlos para afectar la apariencia y funcionalidad de la página Web.

He aquí un ejemplo donde se utiliza el método `real_scape_string()` y la función `strip_tags()` para incapacitar los datos de entrada antes de guardarlos en una base de datos MySQL:

```
<?php
// intenta conectarse con la base de datos
$mysqli = new mysqli("localhost", "user" "pass", "música");
if ($mysqli === false) {
  die ("ERROR: No se estableció la conexión. " . mysqli_connect_error());
}
```

```
// limpia valores de entrada
if (isset($_POST['artista']) && !empty($_POST['artista'])) {
  $artista = $mysqli->real_scape_string(htmlspecialchars($_
POST['artista']));
}

if (isset($_POST['país']) && !empty($_POST['país'])) {
  $país = $mysqli->real_scape_string(htmlspecialchars($_POST['país']));
}

// intenta ejecutar query
// añade un nuevo registro
// datos de salida: " Nuevo artista con id: 7 ha sido añadido."
$sql = "INSERT INTO artistas (artista_nombre, artista_país) VALUES
('$artista', '$país')";
if ($mysqli->query($sql)=== true) {
  echo 'Nuevo artista con id: ' . $mysqli->insert_id . 'ha sido
añadido.';
} else {
  echo "ERROR: No fue posible ejecutar query: $sql. " . $mysqli->error;
}

// cierra conexión
$mysqli->close();
?>
```

Y a continuación un ejemplo para higienizar los datos de entrada de un formulario con la función `htmlentities()`:

```
<?php
// define una matriz para la limpieza de datos
$limpia = array ();

// strip tags from POST input
if (isset($_POST['nombre']) && !empty($_POST['nombre'])) {
  $limpia['nombre'] = htmlentities($_POST['nombre']);
}

// código a procesar//
?>
```

Por último, un ejemplo para limpiar los datos de salida potencialmente peligrosos antes de presentarlos al usuario:

```
<?php
// define una matriz para la limpieza de datos
$limpia = array();

// obtiene datos remotos
```

```
$out = 'Je<script>document.location.href="http://un.sitio.malo.com/";
</script>la';

// convierte todos los caracteres especiales en entidades
// antes de utilizarlas
$limpia['out'] = htmlentities($out);

echo $out; // inseguro
echo $limpia['out']; // seguro
?>
```

Asegurar los datos

Además de limpiar los datos de entrada, también es importante que tu aplicación no permita que los usuarios vean o manipulen los archivos o bases de datos privadas sin que se lo permitas. Las siguientes secciones abordan algunas técnicas que puedes utilizar para proteger el acceso a los archivos de configuración y otras fuentes de datos.

Asegurar los archivos de configuración

A menos que configures explícitamente tu servidor Web para que impida el acceso a ciertos tipos de archivo, un usuario remoto puede acceder a cualquier documento localizado bajo el archivo raíz del servidor . Esto hace que las aplicaciones Web sean muy vulnerables a los accesos remotos sin autorización, puesto que por lo regular guardan sus archivos de configuración con el resto del código de aplicación bajo el archivo raíz del servidor.

Una manera sencilla de rellenar este hueco de seguridad es almacenar cualquier dato de configuración sensible fuera del archivo raíz del servidor Web, y leerlo en tu aplicación conforme sea necesario mediante invocaciones `require()` o `include()`. Dado que ambas funciones aceptan rutas de acceso del sistema de archivos del sistema (en vez de rutas HTTP), pueden importar archivos de directorios que no forman parte del archivo raíz del servidor Web; por lo mismo se hace más difícil que los atacantes tengan acceso a los datos de configuración.

He aquí un ejemplo de lo que se puede hacer:

```
<?php
// la raíz de documentos del servidor es: /var/www/html/
// el archivo de configuración está guardado en: /var/www/apps/conf/

// tu script puede leer esta ruta de acceso
include_once'/var/www/apps/conf/myapp.conf'; // sí funcionará

// un atacante que sigue la misma ruta de acceso con HTTP no lo
conseguirá
file_get_contents ('http://localhost/../apps/conf/myapp.conf '); // no
funcionará
?>
```

NOTA

Para que esto funcione, el directorio que contiene el archivo de configuración debe tener sus bits de permisos configurados de tal manera que el usuario propietario de los procesos del servidor Web pueda leer y modificar sus archivos. Esto es muy importante en los sistemas de desarrollo *NIX.

Asegurar el acceso a la base de datos

Una razón común para no aplicar seguridad al acceso de las bases de datos es porque se trata de una tarea "difícil" y "complicada". Esto no es completamente cierto. En casi todos los casos son sólo unos sencillos pasos que debes seguir para hacer más difícil que los atacantes tengan acceso a tu base de datos, con lo que se reduce drásticamente el riesgo de que le suceda algo malo a tu información. Dado que las bases de datos más utilizadas con PHP son MySQL, las siguientes tres sugerencias están directamente relacionadas con este motor de base de datos; sin embargo, también pueden aplicarse a cualquier otro RDBMS.

- **Dar a los usuarios sólo el nivel de acceso que necesitan** La mayor parte de las bases de datos, incluida MySQL, permiten mantener un control preciso sobre los niveles de acceso garantizados para usuarios individuales que utilizan la base. Hay una buena razón para usar este sistema de privilegios y permitir que los usuarios sólo tengan el acceso que necesitan: si se les concede permiso abierto a toda la base de datos, una sola cuenta en riesgo puede implicar pérdida o robo de importantes datos. MySQL ofrece los comandos `GRANT` y `REVOKE` para controlar los niveles de acceso para usuarios; diferentes comandos realizan la misma acción en otras bases de datos.

  El método recomendado aquí es definir un conjunto de usuarios individual para cada aplicación PHP, y permitirles el acceso únicamente a la base de datos que utiliza dicha aplicación. Esto significa que aunque la aplicación corra riesgo y un usuario malintencionado robe los permisos, la extensión del daño que puede provocar será limitada y no afectará a otras bases de datos en el mismo servidor.

- **Utilizar claves de acceso seguras** Cuando se instala por primera vez MySQL, el acceso al servidor de base de datos queda restringido únicamente al administrador (`'root'`). Por defecto, esta cuenta se inicializa con una clave de acceso en blanco, permitiendo acceso a cualquiera. Resulta sorprendente la frecuencia con la que los desarrolladores novatos suelen omitir este agujero en la seguridad; ellos permanecen ignorantes del peligro que implica seguir trabajando con la configuración por defecto. En MySQL, la contraseña se cambia con la herramienta *mysqladmin*; el proceso puede ser diferente para otras bases de datos.

  Para mejorar la seguridad, entonces, es una buena idea resetear la contraseña del administrador del servidor durante la instalación y luego distribuirla sólo entre un pequeño grupo

de usuarios que realmente necesiten conocerla. Una contraseña de administrador que es conocida por muchos usuarios es demasiado insegura; el viejo adagio "el pez por su propia boca muere" sigue siendo cierta desde los tiempos de nuestros abuelos.

- **Deshabilitar el acceso remoto** La configuración más común para una plataforma de desarrollo LAMP tiene el servidor de base de datos y el Web hospedados en la misma computadora física. En este caso, es posible reducir el riesgo de un ataque externo de manera importante al permitir sólo acceso "local" al servidor de base de datos y bloquear todas las conexiones externas. Con MySQL, esto se realiza utilizando la opción `–skip_networking` al iniciar el servidor; el proceso es diferente en otras bases de datos.

Asegurar las sesiones

En el capítulo 9 viste cómo las herramientas de manejo de sesión integradas a PHP pueden utilizarse para proteger páginas Web, y restringir el acceso a los usuarios que firmaron su entrada correctamente al sistema. Sin embargo, este sistema no es totalmente seguro: siempre resulta posible que un usuario malintencionado "robe" una sesión obteniendo acceso al identificador único de sesión y que lo utilice para recrear la misma.

La guía de seguridad de PHP, escrita por Chris Shiflett y otros, ubicada en www.phpsec.org/projects/guide/, recomienda varios métodos para impedir que los usuarios con malas intenciones roben las sesiones. Una de estas recomendaciones sugiere el uso de contraseñas adicionales para cada cliente con el objetivo de verificar su identidad. En esencia, esta técnica consiste en registrar varios atributos del cliente la primera vez que ingresa a una página manejada por sesión; por ejemplo, puede registrar el encabezado `'User_Agent'` para identificar el nombre del explorador del cliente, para luego verificar estos mismos atributos la próxima vez que visite la página. Si no hay coincidencia, significaría que la sesión del cliente ha sido robada y el acceso a la misma será denegado.

He aquí un ejemplo de cómo podría utilizarse esta técnica. El primer paso implica registrar el contenido del encabezado `'User_Agent'` del cliente como una variable de sesión, cuando la sesión se inicializa por primera vez:

```
<?php
// inicia sesión
// registra el nombre del cliente remoto como variable de sesión
session_start();
$_SESSION['verif_remote_agent'] = $_SERVER['HTTP_USER_AGENT'];
echo 'Sesión iniciada. Su ID de sesión es: ' . session_id();

// Código a procesarse //
?>
```

Ahora, en todos los accesos posteriores, además de verificar la presencia de una sesión válida, el script debe verificar también el encabezado `'User_Agent'` enviado por el cliente.

```
<?php
// verifica sesión
// también verifica el nombre del cliente remoto
session_start();
if (!isset($_SESSION['verif_remote_agent']) || $_SESSION['verif_remote_
agent'] != $_SERVER['HTTP_USER_AGENT']) {
   die('La verificación de sesión falló.');
} else {
  echo 'Sesión verificada.';
}

// Código a procesarse //
?>
```

Este método, aunque sencillo y muy efectivo, no resulta completamente seguro: es posible que un atacante dedicado le dé la vuelta creando un encabezado `'User_Agent'`. Sin embargo, la verificación adicional incorpora un poco más de seguridad que antes no se tenía, porque el atacante tendría que adivinar primero el encabezado correcto. Así, se cuenta con un poco más de seguridad que antes.

Los detalles de las diferentes técnicas sugeridas en la guía de seguridad de PHP para hacer las sesiones más seguras rebasan los alcances de este capítulo; sin embargo, encontrarás muchos ejemplos bien documentados en www.phpsec.org/projects/guide/.

Validar los datos de entrada del usuario

Los formularios no son el único medio por el que un script PHP puede recibir datos de entrada; sin embargo, ése es el *medio más común*. Antes de utilizar estos datos, es necesario verificar su validez, para cortar los datos incorrectos o incompletos. Esta sección aborda varias técnicas que puedes utilizar para validar los datos de entrada del usuario, recibidos en un formulario Web o de cualquier otra manera.

Trabajar con campos obligatorios

Uno de los errores más comunes del programador novato consiste en olvidar la verificación de los valores en los campos requeridos dentro de un formulario Web. Esto puede dar como resultado una base de datos con numerosos registros vacíos, y uno de ellos puede, a su vez, afectar la exactitud requerida para los cálculos.

Una manera sencilla para verificar si todos los campos requeridos en un formulario han sido llenados consiste en revisar cada clave correspondiente en `$_POST` o `$_GET` con la función PHP `empty()`. Ésta verifica si la variable no está vacía y si contiene un valor diferente a cero. He aquí un ejemplo:

```
<?php
// define una matriz de datos válidos
```

```
$valid = array();
// verifica el nombre de usuario
if (!empty($_POST['nombredeusuario'])) {
  $valid['nombredeusuario'] = trim($_POST['nombredeusuario']);
} else {
  die ('ERROR: Nombre de usuario ausente.');
}

// verifica contraseña
if (!empty($_POST['clavedeacceso'])) {
  $valid['clavedeacceso'] = trim($_POST['clavedeacceso']);
} else {
  die ('ERROR: Contraseña ausente.');
}

// Código a procesarse //
?>
```

En este listado, el script verifica primero si se enviaron los datos correspondientes al nombre del usuario y la contraseña. Si ambos resultan verdaderos, entonces el script continúa su ejecución; si cualquiera de ellos regresa un valor falso, el script termina de inmediato, sin intentar siquiera realizar el procesamiento de los datos del formulario.

Advierte también el uso de la matriz `$valid` en el ejemplo, que se utiliza para almacenar datos que han pasado la validación. Ésta es una buena práctica de programación para asegurarte de que no estés utilizando información que no haya sido validada con anticipación. Al asegurar que tu código sólo utilice datos provenientes de `$valid` se reduce el riesgo de insertar datos no válidos para tu base o tus cálculos.

Debemos hacer notar que la función `empty()` sólo se puede utilizar con variables de formulario en que el cero se considera un valor no válido. Esto se debe a que `empty()` regresa un valor falso si la variable que transmites contiene un valor `NULL`, una cadena de texto vacía (`'  '`) o un cero como valor. Si el cero es considerado como valor válido en tu variable de formulario, reemplaza `empty()` por una combinación de funciones `isset()` y `trim()`, como en el siguiente ejemplo:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica incremento de oferta
if (isset($_POST['incremento']) && trim($_POST['incremento']) != '') {
  $valid['incremento'] = trim($_POST['incremento']);
} else {
  die ('ERROR: Valor de incremento ausente.');
}

// Código a procesarse //
?>
```

La verificación de errores aquí es simple y lógica: la función `trim()` se utiliza para recortar espacios vacíos de la variable del campo, que es comparada después con una cadena vacía. Si la comparación resulta verdadera, significa que el campo fue enviado sin información y el script deja de ejecutarse y envía un mensaje de error.

A partir de estos ejemplos, debe quedar claro que una simple verificación condicional es todo lo que se requiere para asegurar que los campos requeridos de tu formulario nunca estén vacíos. No aplicar esta validación significa que los datos de entrada enviados por el usuario serán procesados sin comprobar que todos los valores requeridos estén presentes. Esto puede guiar a situaciones de riesgo: por ejemplo, el usuario podría registrarse con una contraseña en blanco, situación que puede traer consecuencias serias para la seguridad general de la aplicación.

Pregunta al experto

P: Ya valido los datos de entrada de mis formularios con JavaScript. ¿Por qué necesitaría validarlos también con PHP?

R: Es una práctica común utilizar lenguajes de script del lado del cliente, como JavaScript y VBScript, para validar los datos de entrada del lado del cliente. Sin embargo, este tipo de validación no es completamente segura. Dos casos sencillos lo remarcan:

- El código fuente de las páginas Web puede verse en casi todos los exploradores. Es posible que el usuario guarde el formulario Web, desactive la validación del lado del cliente editando el código fuente del formulario y la vuelva a enviar al servidor con valores ilegales.
- Si el usuario desactiva las funciones de JavaScript en su explorador, no se ejecutará el código del lado del cliente. Las rutinas de validación para el formulario serán omitidas por completo, con lo que de nuevo se abren las puertas para el ingreso de valores ilegales en el sistema.

La validación de entrada basada en PHP resuelve ambos aspectos de seguridad, porque el código PHP se ejecuta en el servidor y, por tanto, no puede modificarse ni deshabilitarse del sistema del cliente.

Trabajar con números

Como has visto, cuando defines una nueva tabla en una base de datos, también es necesario definir el tipo de datos que se ingresarán en cada campo. Sin embargo, diferentes sistemas de bases de datos difieren en el rigor aplicable en cada tipo de dato. Por ejemplo, SQLite permite el ingreso de cadenas de texto en los campos marcados como numéricos (`NUMERIC`), mientras que MySQL "corrige" automáticamente estos valores convirtiéndolos en 0 antes de insertarlos en un campo `INT` o `FLOAT`. Dadas estas diferencias, por lo general es una buena

idea forzar la verificación del tipo de dato desde la aplicación PHP, para evitar que a tu base de datos lleguen valores incorrectos o capturados de manera equívoca.

Una manera fácil de verificar si una variable contiene un dato numérico es mediante la función de PHP `is_numeric()`, que regresa un valor verdadero cuando se invoca con números como argumento. El siguiente ejemplo muestra su funcionamiento:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica si el campo edad es un número
if (is_numeric(trim($_POST['edad']))) {
  $valid['edad'] = trim($_POST['edad']);
} else {
  die ('ERROR: Edad no es un número.');
}

// Código a procesarse //
?>
```

Sin embargo, la función `is_numeric()` no distingue entre valores enteros y de punto flotante. Si necesitas este nivel de validación, una opción consiste en convertir primero la variable en un entero o valor de punto flotante y luego probarla con la función `strval()`. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica si la edad es un valor entero
if (strval($_POST['edad']) == strval((int)$_POST['edad'])) {
  $valid['edad'] = trim($_POST['edad']);
} else {
  die ('ERROR: Edad no es un valor entero.');
}

// verifica si el precio es un valor de punto flotante
if (strval($_POST['precio']) == strval((float)$_POST['precio'])) {
  $valid['precio'] = trim($_POST['precio']);
} else {
  die ('ERROR: Precio no es un valor decimal.');
}

// Código a procesar //
?>
```

Aquí la lógica es muy simple: si la variable contiene un entero, el valor de la cadena regresado por `strval()` después de convertirlo en entero será idéntico al valor original de la variable.

Por otra parte, si la variable contiene un valor que no sea numérico, el valor original de la variable no coincidirá con el valor obtenido después de la conversión.

Para una prueba aún más estrecha, utiliza la función `ctype_digit()` de PHP. Esta función verifica cada carácter del valor que le es transmitido, y regresa un valor verdadero sólo si cada carácter es un valor entre 0 y 9. He aquí un ejemplo:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica si el campo edad es un número
if (ctype_digit($_POST['edad'])) {
  $valid['edad'] = trim($_POST['edad']);
} else {
  die ('ERROR: Edad no es un número.');
}

// Código a procesarse //
?>
```

PRECAUCIÓN

Advierte que el rigor de `ctype_digit()` puede ser contraproducente en algunos casos: la función regresará un valor falso cuando se transmitan valores decimales, porque el punto decimal no es un dígito entre 0 y 9.

En algunos casos tal vez desees reforzar un rango de valores numéricos que acepte tu aplicación. Verificar esto es tan sencillo como utilizar operadores de comparación PHP, como se muestra a continuación:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica si el valor es un entero entre 1 y 31
if ((strval($_POST['day']) == strval((int)$_POST['day'])) &&
($_POST['day'] >= 1 && $_POST['day'] <= 31)) {
  $valid['day'] = trim($_POST['day']);
} else {
  die ('ERROR: Edad no es un número.');
}

// Código a procesarse //
?>
```

Trabajar con cadenas de texto

Muchas bases de datos (incluida MySQL) truncarán de manera automática los valores de cadena de caracteres, si éstos exceden la longitud específica correspondiente al campo. Esto es inquietante porque significa que los datos de entrada proporcionados por el usuario pueden ser fácil y silenciosamente corrompidos sin que aparezca ninguna notificación al respecto. Por tanto, es una buena idea realizar una validación a nivel de aplicación para las cadenas de texto, para alertar a los usuarios en caso de que el texto rebase la longitud permitida y facultarlos para modificar la cadena.

Un buen lugar para comenzar con este tipo de validación es la función `strlen()`, que regresa la longitud de la cadena. Esto resulta útil para asegurar que los datos del formulario no excedan una longitud específica. Examina el siguiente ejemplo, que lo ilustra:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica el nombre de usuario
if (!empty($_POST['nombredeusuario'])) {
  $nombredeusuario = trim($_POST['nombredeusuario']);
} else {
  die ('ERROR: Nombre de usuario ausente.');
}

// verifica la longitud del nombre de usuario
if (strlen($nombredeusuario) <= 25) {
  $valid['nombredeusuario'] = $nombredeusuario;
} else {
  die ('ERROR: Nombre de usuario demasiado largo.');
}

// Código a procesarse //
?>
```

Si necesitas estar completamente seguro de que los datos de un campo son sólo caracteres alfabéticos (por ejemplo, en los casos de nombres y apellidos), PHP ofrece la función `ctype_alpha()`. Al igual que la función `ctype_digit()` abordada en la sección anterior, esta función regresa un valor verdadero sólo si cada carácter dentro de la cadena es alfabético. He aquí un ejemplo:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica el nombre de pila
if (isset($_POST['nombre']) && ctype_alpha($_POST['nombre'])) {
```

```
  $valid['nombre'] = trim($_POST['nombre']);
} else {
  die ('ERROR: Nombre ausente o no válido.');
}

// Código a procesarse //
?>
```

TIP

PHP también ofrece la función `ctype_alnum()` para caracteres alfanuméricos, la función `ctype_space()` para espacios en blanco y la función `ctype_print()` para caracteres imprimibles. Para obtener la lista completa visita www.php.net/ctype.

Comparar patrones

Para una validación de cadenas de texto más compleja, PHP da soporte a *expresiones regulares*, una herramienta muy poderosa para validar patrones de cadenas de texto. Comúnmente asociada con la plataforma *NIX, una expresión regular te permite definir patrones utilizando un conjunto especial de *metacaracteres*. Estos patrones pueden compararse con el texto existente en un archivo, con los datos insertados en una aplicación o con los valores enviados en un formulario Web. Dependiendo de la coincidencia entre los patrones, los datos pueden ser considerados válidos o no.

Siguiendo los estándares de Perl, una expresión regular se encierra entre diagonales y por lo regular tiene esta apariencia:

```
/fo+/
```

El símbolo de adición (+) en esta expresión es un metacarácter: significa "coincide con una o más existencias del carácter anterior". En el contexto del ejemplo anterior, la expresión regular se traduce como "un patrón que contenga el carácter *f* seguido por una o más apariciones del carácter *o*". Por tanto, coincidirá con las palabras "fotografía", "formidable" y "fogata", pero no coincidirá con "frío" ni con "fatal".

Similares a + son los metacaracteres `*` y `?`. Son utilizados para coincidir con cero o más existencias de los caracteres anteriores y cero o una existencia de los caracteres precedentes, respectivamente. De esta manera, la expresión regular `/ma*/` coincidirá con "mamá", "manía" y "menor", mientras que la expresión `/com?/` coincidirá con "coincidencia", "consciente", "colateral", pero no coincidirá con "ciencia" ni con "cama".

También puedes especificar un rango con el número de coincidencias. Por ejemplo, la expresión regular `/jim{2,6}/` coincidiría con "jimmy" y con "jimmmmmmy", pero no con "jim". Los números encerrados entre llaves representan los valores menor y mayor del rango de la coincidencia; puedes dejar fuera el límite superior para un rango abierto.

Metacarácter	Lo que significa
^	Inicio de una cadena de texto
$	Final de una cadena de texto
.	Cualquier carácter, excepto uno de nueva línea
\s	Un solo espacio en blanco
\S	Un solo carácter que no sea un espacio en blanco
\d	Un dígito entre 0 y 9
\w	Un carácter alfabético o numérico, o subrayado
[A-Z]	Un carácter alfabético en mayúsculas
[a-z]	Un carácter alfabético en minúsculas
[0-9]	Un dígito entre 0 y 9
|	Operador lógico OR (O)
(?=	Prueba condicional positiva
(?!	Prueba condicional negativa

Tabla 11-1 Metacaracteres de expresiones regulares

La tabla 11-1 muestra una breve lista de metacaracteres útiles.

Para ver el funcionamiento de éstos, imagina un formulario Web que requiere que el usuario introduzca su nombre de usuario, contraseña y número de seguro social. Al validar estos datos de entrada, el desarrollador necesita reforzar las siguientes restricciones:

- El nombre de usuario debe tener entre tres y ocho caracteres de longitud y contener sólo caracteres alfabéticos.
- La contraseña debe tener entre cinco y ocho caracteres de longitud y contener al menos un número.
- El número de seguro social debe contener nueve dígitos, y un guión después del tercer y quinto dígito.

Funciones como `strlen()` y `ctype_alnum()` son demasiado sencillas para este tipo de validación. En lugar de ellas, un mejor método consiste en definir patrones para cada valor de entrada, y comparar el dato de entrada contra el patrón para decidir si es válido o no. He aquí un ejemplo de la manera de hacerlo con expresiones regulares:

```
<?php
// define una matriz de datos válidos
```

```
$valid = array();
// verifica nombre de usuario
if (isset($_POST['nombredeusuario']) && preg_match('/^([a-zA-Z]){3,8}$/',
$_POST['nombredeusuario'])) {
  $valid['nombredeusuario'] = trim($_POST['nombredeusuario']);
} else {
  die ('ERROR: Nombre de usuario ausente o no válido.');
}

// verifica contraseña
if (isset($_POST['clavedeacceso']) && preg_match('/^(?=.*\d).{5,8}$/',
$_POST['clavedeacceso'])) {
  $valid['clavedeacceso'] = trim($_POST['clavedeacceso']);
} else {
  die ('ERROR: Contraseña ausente o no válida.');
}

// verifica número de seguro social
if (isset($_POST['nss']) && preg_match('/^([0-9]){3}-([0-9]){2}-
[0-9]){4}$/', $_POST['nss'])) {
  $valid['nss'] = trim($_POST['nss']);
} else {
  die ('ERROR: NSS ausente o no válido.');
}

// Código a procesarse //
?>
```

Este código utiliza la función PHP `preg_match()` para probar si los datos de entrada se acoplan al patrón definido por la misma. Esta función `preg_match()` requiere dos argumentos obligatorios: un patrón y los valores que se compararán contra este patrón. Regresa un valor verdadero si se encuentra una coincidencia y falso en caso contrario.

Regresando a las expresiones regulares, no debe ser muy difícil decodificarlos después de revisar el material de la tabla 11-1. Especifican el rango de caracteres permitido para cada valor insertado y las longitudes mínimas y máximas para cada subconjunto de caracteres. Se les considerará válidos sólo si los datos de entrada coinciden con el patrón especificado.

He aquí otro ejemplo; éste valida números telefónicos internacionales:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica el número telefónico
if (isset($_POST['tel']) && preg_match("/^(\+|00[1-9]{8,14}$/",
$_POST['tel'])) {
  $valid['tel'] = trim($_POST['tel']);
} else {
  die ('ERROR: Número telefónico ausente o no válido.');
}
?>
```

Si juegas un poco con este código, verás que acepta los números *+441865123456* y *0091112345678*, aunque cada uno tiene un formato diferente. Esto se debe a que la expresión regular utiliza el metacarácter |, que funciona como el operador lógico `OR` (`O`) y hace posible crear un patrón que da soporte a opciones internamente. Por supuesto, puedes hacer el patrón más riguroso o más flexible, dependiendo de los requerimientos específicos de tu aplicación.

De esta manera, las expresiones regulares proporcionan una herramienta poderosa y flexible para validar los datos de entrada de acuerdo con una serie de reglas personalizadas... aunque cueste un poco acostumbrarse a la sintaxis.

Validar direcciones de correo electrónico y URL

Una tarea común cuando se trabaja con datos de entrada proporcionados por el usuario incluye verificar las direcciones de correo electrónico y los URL, para asegurar que tengan el formato correcto. Existen varias maneras de realizar esto; tal vez el método más común consista en utilizar expresiones regulares, como en el siguiente ejemplo:

```
<?php
// función para validar
// una dirección de correo electrónico
function validaemail($str) {
    return preg_match("/^([a-z0-9_-])+([\.a-z0-9_-])*@([a-z0-9-])+(\
.[a-z0-9-]+)*\.([a-z]{2,6})$/", strtolower($str));
}

// verifica dirección de correo electrónico
// datos de salida: 'válido'
echo validaemail("joe@dominio.com") ? "válido" : "no válido";

// verifica dirección de correo electrónico
// datos de salida: 'no válido'
echo validaemail("joe@dominio.") ? "válido" : "no válido";
?>
```

No es difícil encontrar una expresión regular para validar direcciones de correo electrónico; se puede encontrar una gran cantidad de opciones en línea, desde muy estrictas hasta más relajadas. El código anterior utiliza uno de los patrones más estrictos, que restringe el rango de los caracteres tanto para el nombre de usuario como para el dominio y requiere una longitud de caracteres para el nivel superior de dominio entre dos y seis caracteres.

También es posible escribir una función similar para validar URL. He aquí un ejemplo:

```
<?php
// función para validar URL
function validaUrl($str) {
    return preg_match("/^(http|https|ftp): \/\/([a-z0-9]([a-z0-9_-]*[a-
z0-9])?\.)+[a-z]{2,6}\/?([a-z0-9\?\._-~&#=+%]*)?/", strtolower($str));
}
```

```
// verifica URL
// datos de salida: "válido"
echo validaUrl("http://www.ejemplo.com/html/index.php") ? "válido" : "no
válido";

// datos de salida: "no válido"
echo validaUrl("http://ejemplo.com") ? "válido" : "no válido";
?>
```

Los URL tienen diferentes formas y tamaños, como las direcciones de correo electrónico, y puedes seleccionar el grado de rigidez con el que los vas a validar. La expresión regular utilizada en el ejemplo anterior restringe los protocolos a HTTP, HTTPS y FTP; requiere que el nivel superior del dominio tenga una longitud entre dos y seis caracteres, y da soporte a rutas de acceso que contengan nombres de archivo y anclas.

Como opción, quizás una manera más sencilla de completar la misma tarea consiste en utilizar la función PHP `filter_var()`, que proporciona reglas de validación integradas para tipos comunes de datos de entrada, incluidas direcciones de correo electrónico y URL. He aquí una versión modificada del ejemplo anterior para validar una dirección de correo electrónico:

```
<?php
// función para validar
// una dirección de correo electrónico
function validaemail($str){
    return filter_var($str, FILTER_VALIDATE_EMAIL);
}

// verifica la dirección de correo electrónico
// datos de salida: 'válido'
echo validaemail("joe@dominio.com") ? "válido" : "no válido";

// verifica la dirección de correo electrónico
// datos de salida: 'no válido'
echo validaemail("joe@dominio.") ? "válido" : "no válido";
?>
```

Aquí, la función `filter_var()` prueba la variable enviada para ver si se trata de una dirección de correo electrónico válida, y regresa un valor verdadero o falso según el caso. La constante `FILTER_VALIDATE_EMAIL` le indica a `filter_var()` que verifique la variable contra el patrón que se espera para una dirección de correo electrónico. De manera similar, existe la constante `FILTER_VALIDATE_URL`, que puede utilizarse para probar la validez de los URL (aunque utiliza una prueba menos rigurosa que las expresiones regulares mostradas antes):

```
<?php
// función para validar una URL
function validaUrl($str) {
    return filter_var($str, FILTER_VALIDATE_URL);
```

```
}
// verifica URL
// datos de salida: 'válido'
echo validaUrl("http://www.ejemplo.com/html/index.php") ? "válido" : "no
válido";

// datos de salida: 'no válido'
echo validaUrl("http:/ejemplo.com") ? "válido" : "no válido";
?>
```

Trabajar con fechas

Validar fechas es otro aspecto importante para el procesamiento de los datos de entrada. Sin la validación apropiada es muy fácil que el usuario introduzca una fecha no válida como 29 de febrero de 2009 o 31 de junio de 2008. Por ello, es importante asegurar que los valores de fecha proporcionados por el usuario sean genuinos antes de utilizarlos en los cálculos del programa.

En PHP, esta tarea es muy sencilla en comparación con otros programas, porque tiene la función `checkdate()`, que acepta tres argumentos: mes, día y año, y regresa un valor booleano que indica si la fecha es válida o no. El siguiente ejemplo lo ilustra:

```
<?php
// define una matriz de datos válidos
$valid = array();

// verifica el día
if (!empty($_POST['día']) && ctype_digit($_POST['día'])) {
  $valid['día'] = trim($_POST['día']);
} else {
  die ('ERROR: Día ausente.');
}

// verifica el mes
if (!empty($_POST['mes']) && ctype_digit($_POST['mes'])) {
  $valid['mes'] = trim($_POST['mes']);
} else {
  die ('ERROR: Mes ausente.');
}

// verifica el año
if (!empty($_POST['año']) && ctype_digit($_POST['año'])) {
  $valid['año'] = trim($_POST['año']);
} else {
  die ('ERROR: Año ausente.');
}
// verifica la validez de la fecha
if (!checkdate($valid['mes'], $valid['día'], $valid['año'])){
  die ('ERROR: Fecha no válida.');
```

```
}

// Código a procesarse//
?>
```

PRECAUCIÓN

Si estás almacenando fechas en los campos `DATE`, `DATETIME` o `TIMESTAMP` de MySQL, ten presente que esta base de datos *no* realiza ninguna verificación rigurosa de fecha por sí misma. Por lo tanto, la responsabilidad de verificar las fechas antes de guardarlas en una tabla de MySQL reside por completo en el desarrollador de la aplicación. Lo más que hará MySQL, en caso de encontrar una fecha *no* válida, será reemplazarla con una serie de ceros... que no es la mejor solución. Lee más sobre el manejo de fechas y horas por parte de MySQL en http://dev.mysql.com/doc/mysql/en/datetime.html.

Prueba esto 11-1 Validar datos de entrada de un formulario

Ahora que ya conoces los aspectos básicos sobre el saneamiento de los datos de entrada y la validación de los mismos, apliquemos estos conocimientos en un proyecto práctico. El siguiente ejemplo presenta un formulario Web que solicita al usuario el ingreso de varios detalles de un libro: título, autor, número ISBN y precio. Después valida estos datos utilizando una combinación de las técnicas abordadas en las secciones anteriores; una vez validados los datos, los guarda en una base de datos de SQLite.

Para comenzar, crea una base de datos de SQLite y una tabla para almacenar los registros ingresados por el usuario:

```
shell> sqlite libros.db
SQLite version 3.3.17
Enter ".help" for instructions
sqlite> CREATE TABLE libros (
...>  id INTEGER PRIMARY KEY,
...>  título TEXT,
...>  autor TEXT,
...>  isbn INTEGER,
...>  precio REAL
...>);
```

A continuación escribe el código PHP para validar los datos ingresados por el usuario mediante un formulario Web, y guárdalos en la base de datos. He aquí el script (*libros.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 11-1: Validar datos de entrada de un formulario</title>
     <style type="text/css">
```

```
    div.error {
      color:red;
      font-weight: bolder;
    }
    div.correcto {
      color:green;
      font-weight: bolder;
    }
    </style>
  </head>
  <body>
    <h2>Proyecto 11-1: Validar datos de entrada de un formulario</h2>
    <h3 style="background-color: silver">Escriba los detalles del libro</h3>
<?php
    // muestra mensaje de error en la validación de los datos de entrada
    function getInputError($key, $errArray) {
      if (in_array($key, $errArray)) {
        return "<div class=\"error\">ERROR: Datos no válidos para el
campo '$key'</div>";
      } else {
        return false;
      }
    }

    $inputErrors = array();
    $submitted = false;

    // si el formulario ya fue enviado
    // valida los datos de entrada
    if (isset($_POST['submit'])) {
      $submitted = true;
      $valid = array();

      // valida título
      if (!empty($_POST['título'])) {
        $valid['título'] = htmlentities(trim($_POST['título']));
      } else {
        $inputErrors[] = 'título';
      }

      // valida nombre del autor
      if (!empty($_POST['autor']) && preg_match("/^[a-zA-Z\s.\-]+$/",
$_POST['autor'])) {
        $valid['autor'] = htmlentities(trim($_POST['autor']));
      } else {
        $inputErrors[] = 'autor';
      }

      // valida ISBN
```

(*continúa*)

```
      if (!empty($_POST['isbn']) && preg_match('/^(97(8|9))?\d{9}
(\d|X)$/', $_POST['isbn'])) {
        $valid['isbn'] = htmlentities(trim($_POST['isbn']));
      } else {
        $inputErrors[] = 'isbn';
      }

      // valida precio
      if (!empty($_POST['precio']) && is_numeric($_POST['precio']) &&
$_POST['precio'] > 0)
{
        $valid['precio'] = htmlentities(trim($_POST['precio']));
      } else {
        $inputErrors[] = 'precio';
      }
    }

    // si el formulario no ha sido enviado
    // o si existe un error de validación
    // vuelve a mostrar el formulario
    if (($submitted == true && count($inputErrors) > 0) || $submitted ==
false) {
?>
    <form method="post" action="libros.php">
      Título: <br />
      <input type="text" size="25" name="título"
         value="<?php echo isset($_POST['título']) ? $_POST['título'] :
'';?>" /><br />
      <?php echo getInputError('título', $inputErrors); ?>
      <p>
      Autor: <br />
      <input type="text" size="25" name="autor"
         value="<?php echo isset($_POST['autor']) ? $_POST['autor'] :
'';?>" /><br />
      <?php echo getInputError('autor', $inputErrors); ?>
      <p>
      ISBN: <br />
      <input type="text" size="25" name="isbn"
         value="<?php echo isset($_POST['isbn']) ? $_POST['isbn'] :
'';?>" /><br />
      <?php echo getInputError('isbn', $inputErrors); ?>
      <p>
      Precio: <br />
      <input type="text" size="6" name="precio"
         value="<?php echo isset($_POST['precio']) ? $_POST['precio'] :
'';?>" /><br />
      <?php echo getInputError('precio', $inputErrors); ?>
      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
```

```
<?php
    // si el formulario fue enviado sin errores
    // inserta el contenido en la base de datos
    } else {
      // abre la base de datos SQLite
      try {
        $pdo = new PDO('sqlite:libros.db');
        $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
      } catch (PDOException $e) {
        die("ERROR: No se estableció la conexión: " . $e->getMessage());
      }

      // crea un query de inserción
      try {
        $título = $pdo->quote($valid['título']);
        $autor = $pdo->quote($valid['autor']);
        $isbn = $pdo->quote($valid['isbn']);
        $precio = $pdo->quote($valid['precio']);
        $sql = "INSERT INTO libros (título, autor, isbn, precio) VALUES
($título, $autor, $isbn, $precio)";
        $ret = $pdo->exec($sql);
        echo '<div class="correcto">ÉXITO: Registro guardado;</div>';
      } catch (Exception $e) {
        echo '<div class="error">ERROR: ' . $e->getMessage() . '</div>';
      }

      // cierra conexión
      unset($pdo);
    }
?>
  </body>
</html>
```

La figura 11-1 muestra la apariencia del formulario Web.

Cuando se envía este formulario, el script comienza a trabajar validando los datos proporcionados por el usuario. Además de verificar que todos los campos tengan su respectivo valor, el script también utiliza expresiones regulares para probar el nombre del autor, el número de ISBN, y la función `is_numeric()` asegura que el valor insertado en el campo precio sea un número. Los campos que no tienen validación son marcados, añadiéndolos a la matriz `$inputErrors`. Los campos válidos son limpiados pasándolos por la función `htmlentities()` y luego añadiéndolos a la matriz `$valid`.

Una vez que se han completado las fases de validación y limpia, el script comienza a verificar si ha ocurrido algún error de validación. Suponiendo que no haya errores, se abre la conexión PDO hacia la base de datos SQLite, y los valores de la matriz `$valid` se integran en una consulta `INSERT` y se guardan en la base de datos. Sin embargo, si ocurre uno o más errores, el formulario se vuelve a desplegar y muestra un mensaje de error para que el usuario corrija los valores equivocados. Advierte la función definida por el usuario `getInput`

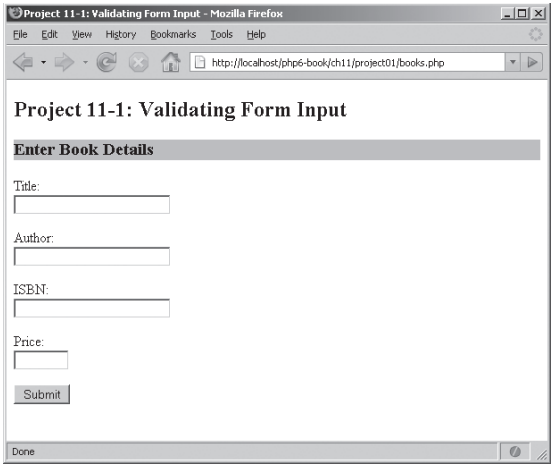


Figura 11-1 Formulario Web para insertar detalles de libros

`Error()`, que verifica y muestra el estatus de error de cada valor de entrada, proporcionando así una manera conveniente de notificar al usuario sobre todos los errores cometidos al mismo tiempo en lugar de mostrarlos uno tras otro después de cada corrección.

La figura 11-2 muestra la salida cuando existe error en alguno de los campos del formulario. La figura 11-3 muestra la salida cuando los datos de entrada son validados y guardados con éxito.

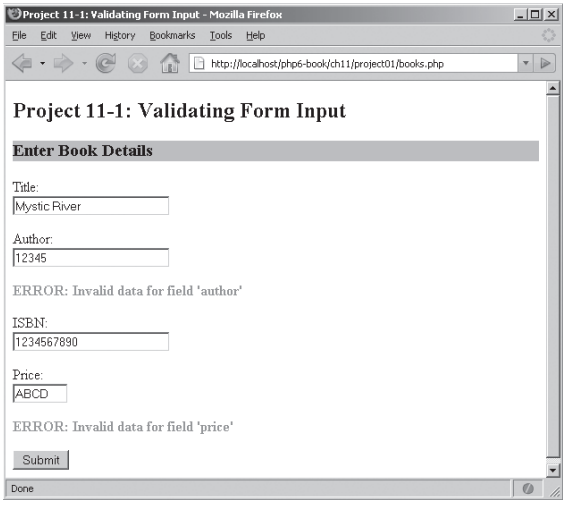


Figura 11-2 La salida cuando los datos del formulario tienen errores

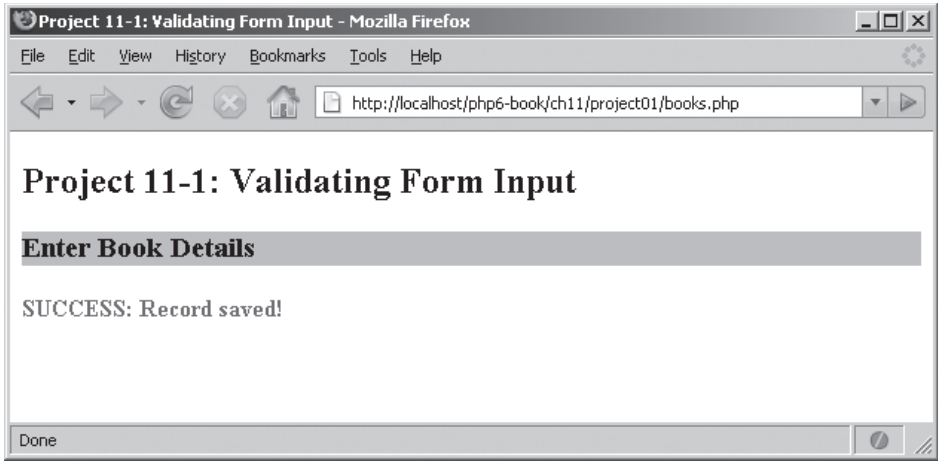


Figura 11-3 La salida cuando los datos del formulario son validados con éxito

Configurar la seguridad con PHP

Además de las técnicas presentadas en secciones anteriores, también existen varias directivas de configuración de PHP que puedes establecer con el fin de reducir el riesgo de que los atacantes obtengan acceso no autorizado a tu aplicación. Todas estas directivas pueden establecerse en el archivo de configuración de PHP, *php.ini*, o durante el periodo de ejecución con la función `ini_set()` de PHP. A continuación una breve lista:

- **`'disable_functions'`** La directiva `'disable_functions'` permite que los desarrolladores desactiven ciertas funciones integradas de PHP por razones de seguridad. Ejemplos de estas funciones son las que permiten la ejecución remota de comandos como `exec()` y `passthru()`, o bien funciones que muestran información interna de PHP o el servidor Web como `phpinfo()`.
- **`'disable_classes'`** Al igual que `'disable_functions'`, la directiva `'disable_classes'` permite que los desarrolladores desactiven ciertas clases PHP que impliquen cierto riesgo.
- **`'allow_url_fopen'`** La directiva `'allow_url_fopen'` determina si un script PHP puede leer datos de un URL remoto como si fueran archivos, con las funciones de archivo como `file_get_contents()`, `include()` o `fopen()`. A menos que tu aplicación necesite obtener datos de un URL remoto a través de HTTP o FTP, esta función debe quedar deshabilitada.

- **`'open_basedir'`** La directiva `'open_basedir'` permite a los desarrolladores restringir todas las operaciones de archivo que tienen lugar dentro del script PHP a un directorio en particular y sus directorios secundarios. Cuando esta directiva está activa, un script PHP no podrá "ver" fuera del directorio de nivel superior mencionado en la directiva. Es útil para restringir la aplicación a un árbol de directorios definido, y reducir así la posibilidad de acceder a archivos de sistema sensibles, o manipularlos.
- **`'error_reporting'`** Ya viste la función `error_reporting()` en el capítulo anterior. Esta directiva `'error_reporting'` realiza la misma tarea: permite que el desarrollador controle el nivel de reporte de errores. La guía de seguridad de PHP recomienda que en la mayor parte de los casos sea configurada como `E_ALL`, de manera que todos los errores (notificaciones y advertencias) sean reportados al desarrollador.
- **`'display_errors'`** La directiva `'display_errors'` controla la posibilidad de que los mensajes de error generados por el script sean presentados o no en pantalla. El manual de PHP recomienda activar esta directiva en ambientes de desarrollo, pero deshabilitarla en ambientes de producción, porque los atacantes pueden utilizar la información del diagnóstico que muestra un mensaje de error para localizar y explotar vulnerabilidades en el código PHP.
- **`'log_errors'`** El hecho de que no se muestren los errores, no significa que deban ser completamente ignorados. La directiva `'log_errors'` especifica que los errores que ocurren en un script deben escribirse en un archivo de ingreso para su posterior análisis. Casi siempre, esta directiva debe estar activada, en especial si `'display_errors'` está deshabilitada, de manera que exista un registro de los errores generados por la aplicación.
- **`'expose_php'`** La directiva `'expose_php'` determina si PHP añade información sobre sí mismo al contexto del servidor Web. El manual PHP recomienda que se deshabilite esta directiva, para evitar que los atacantes obtengan información adicional sobre las capacidades del servidor.
- **`'max_input_time'`** La directiva `'max_input_time'` determina la cantidad máxima de tiempo que tiene un script PHP para recibir o interpretar datos de entrada, incluida la información transmitida por GET y POST. Un límite de este tiempo reduce el tiempo del que dispone un atacante para intentar la construcción y transmisión interactiva de una requisición POST o GET.
- **`'session.name'`** La directiva `'session.name'` controla el nombre de la cookie de sesión utilizado por PHP para rastrear la sesión del usuario. Por defecto, esta cookie recibe el nombre de `PHPSESSID`. Es buena idea cambiar este nombre, de nuevo con el fin de hacer más difícil que los atacantes identifiquen y vean su contenido.

La tabla 11-2 muestra una lista de dónde pueden establecerse estas directivas.

Directiva	Puede ser establecida en *php.ini*	Puede ser establecida en tiempo de ejecución ini_set( )
`'disable_functions'`	Sí	No
`'disable_classes'`	Sí	No
`'allow_url_fopen'`	Sí	Sí
`'open_basedir'`	Sí	Sí
`'error_reporting'`	Sí	Sí
`'display_errors'`	Sí	Sí
`'log_errors'`	Sí	Sí
`'expose_php'`	Sí	No
`'max_input_time'`	Sí	No
`'session.name'`	Sí	Sí

Tabla 11-2 Directivas de seguridad de PHP

NOTA

Es necesario reiniciar el servidor Web para activar los cambios hechos en el archivo de configuración PHP *php.ini*.

Resumen

Este capítulo se concentró específicamente en la seguridad, presentando varias técnicas que puedes usar para reducir el riesgo de daño (ya sea intencional o accidental) a tu aplicación PHP. Te presentó las bases del saneamiento de datos de entrada y salida, explicó la manera de limpiar los datos de entrada para las bases de datos y de salida para terceros; te ofreció consejos para tomar medidas de seguridad en tus archivos de aplicación, bases de datos y sesiones. También te ofreció un curso relámpago sobre la validación de datos de entrada, mostrándote la manera de utilizar expresiones regulares, funciones de tipo de datos y pruebas condicionales con el fin de validar la información proporcionada por el usuario antes de utilizarla para realizar cálculos o insertarla en tu base de datos.

Este capítulo cubrió mucho terreno, pero es sólo la punta del *iceberg*: la seguridad en PHP es un tema muy amplio y construir una aplicación robusta, resistente a ataques, requiere tanto conocimiento como experiencia. Por fortuna, no es difícil adquirir experiencia en este tema; existen numerosos recursos disponibles en línea para ayudarte a aprender más sobre potenciales ataques y a cerrar huecos de seguridad en el código de tus aplicaciones. Te invitamos a visitar las siguientes direcciones para aprender más sobre los temas tratados en este capítulo:

- Una panorámica sobre temas de seguridad en PHP, en **www.php.net/security**
- La Guía de Seguridad PHP, en **www.phpsec.org/projects/guide/**

- Artículos y discusiones sobre la seguridad en PHP por PHP Security Consortium, en **www.phpsec.org/**
- Funciones de expresiones regulares, en **www.php.net/pcre**
- Funciones de tipos de caracteres, en **www.php.net/ctype**
- Discusiones sobre ataques de sitios cruzados, en **http://en.wikipedia.org/wiki/Cross-site_scripting**
- Una discusión sobre ataques de inyección SQL, en **http://en.wikipedia.org/wiki/SQL_injection**
- Ejemplos de ataque de sitios cruzados, en **http://ha.ckers.org/xss.html**
- Ejemplos de expresiones regulares, en **www.regexlib.com**
- Tutoriales sobre expresiones regulares, en **www.melonfire.com/community/columns/trog/article.php?id=2** y **www.regular-expressions.info/tutorial.html**
- Tutoriales sobre validación de datos del lado del cliente en **www.sitepoint.com/article/client-side-form-validation** y **http://home.cogeco.ca/~ve3ll/jstutor5.htm**

Autoexamen Capítulo 11

1. Menciona y explica dos ataques abordados en este capítulo. También explica cómo defender tu aplicación contra los mismos.
2. Menciona y ejemplifica dos funciones para limpiar los datos de salida de una aplicación.
3. Demuestra el uso de una expresión regular para validar códigos postales de Estados Unidos, con el formato *ddddd-dddd*.
4. ¿Por qué debe deshabilitarse el despliegue de errores en los ambientes de producción?
5. Explica lo que hacen las siguientes funciones:
   - `ctype_alnum()`
   - `addslashes()`
   - `filter_var()`
   - `htmlentities()`
   - `sqlite_escape_string()`
   - `preg_match()`
   - `strval()`

Capítulo 12

Extender PHP

Habilidades y conceptos clave

- Aprender sobre los depósitos de código PEAR y PECL
- Entender cómo instalar un paquete PEAR y uno PECL
- Comunicarse con un servidor POP3 utilizando el paquete PEAR
- Crear dinámicamente un archivo ZIP utilizando el paquete PECL

Como un lenguaje de código libre, PHP tiene el soporte de miles de desarrolladores de todo el mundo. El soporte de esta comunidad, junto con la facilidad de uso del lenguaje, ha producido cientos de aplicaciones auxiliares y extensiones que pueden ser utilizadas para añadir nuevas capacidades al motor original de PHP. Estos auxiliares y extensiones conforman una base robusta y estable para el código que es invaluable para los desarrolladores, porque les permite crear rápidamente aplicaciones Web de alta calidad y eficiencia sin necesidad de "escribir mucho código".

Dos de los más extensos depósitos en línea para estos complementos son el depósito de extensiones y aplicaciones PHP (PEAR, PHP Extension and Application Repository), y la biblioteca comunitaria de extensiones PHP (PHP Extension Community Library). Este capítulo presenta una introducción a ambos; utiliza ejemplos prácticos para demostrar cómo pueden hacer más simple y efectivo el desarrollo de aplicaciones PHP.

Utilizar PEAR

PEAR es el depósito de extensiones y aplicaciones PHP, disponible en la dirección Web http://www.pear.php.net/. Su propósito es proporcionar a los desarrolladores una biblioteca de clases PHP reutilizables (también llamadas *paquetes*), que pueden integrarse fácilmente en cualquier aplicación PHP y que siguen el estilo estándar de codificación, así como la estructura de archivos. Los paquetes de PEAR son distribuidos como archivos TAR comprimidos y pueden instalarse en cualquier sistema de desarrollo PHP utilizando el instalador PEAR (incluido con todas las distribuciones de PHP).

Los paquetes PEAR abarcan una diversa amalgama de categorías. Algunas de ellas son:

- Autentificación del usuario (Auth, Auth_HTTP).
- Integración de bases de datos (MDB, DB_DataObject, DB_QueryTool, Query2XML y Structures_DataGrid).
- Procesamiento de formularios (HTML_QuickForm, Validate y Text_CAPTCHA).
- Protocolos de red (Net_SMTP, Net_POP3, NET_LDAP y Net_FTP).

- Formatos de archivo (File_PDF, Archive_TAR y Spreadsheet_Excel_Writer).
- Localización de aplicaciones (I18N).
- Comparaciones, ingreso y prueba de unidades (Estudio comparativo, Log, PHPUnit).

Instalar paquetes PEAR

PHP incluye un instalador automático para los paquetes PEAR. Este instalador tiene la capacidad de conectarse automáticamente con el servidor central PEAR, descargar los paquetes requeridos e instalarlos en tu ambiente de desarrollo PHP.

NOTA

Los usuarios de Windows primero deben configurar manualmente el instalador PEAR, ejecutando el archivo por lotes *go-pear.bat*, localizado en el directorio de instalación de PHP. Este archivo configurará el instalador de PEAR, generará los registros necesarios y colocará los archivos de instalación en su ubicación correcta dentro del sistema. Para mayores detalles, revisa las notas de instalación para Windows en el manual de PEAR, en http://pear.php.net/manual/en/installation.getting.php.

Para instalar un paquete de PEAR utilizando el instalador, simplemente escribe el siguiente comando en la consola:

```
shell> pear install nombre-del-paquete
```

El instalador de PEAR se conectará con el servidor de paquetes PEAR, descargará el paquete solicitado y lo instalará en la ubicación adecuada dentro del sistema. La figura 12-1 muestra un ejemplo de la instalación del paquete Net_POP3.

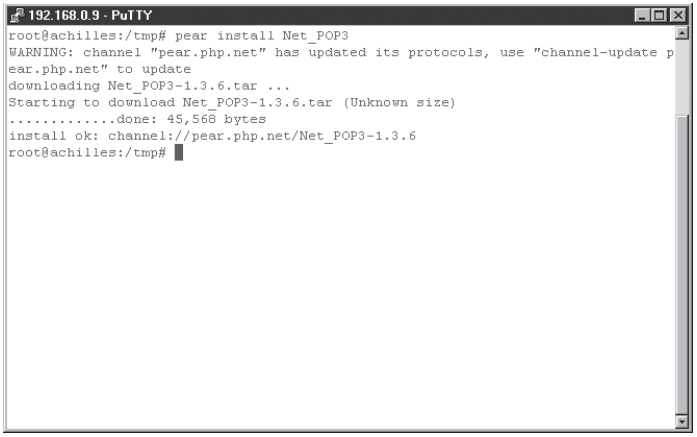


Figura 12-1 Instalación de un paquete de PEAR en UNIX

TIP

Si no puedes hacer que el instalador PEAR funcione correctamente, intenta añadir la opción –v a la línea de comando para la instalación, para que muestre información adicional de depuración.

Prueba esto 12-1 Acceder a buzones electrónicos POP3 con PEAR

Para demostrar la utilidad de PEAR en la vida real, hagamos una aplicación sencilla: un lector de buzones electrónicos, que acepte la identificación del usuario y lo conecte con su servidor de correo electrónico para recuperar los datos de su buzón. Para hacerlo en PHP, un desarrollador necesitaría recompilar PHP con soporte para las extensiones IMAP o programar una conexión que se comunicara directamente con el servidor de correo electrónico, enviar manualmente una solicitud e interpretar las respuestas. La primera opción consume demasiado tiempo (y por lo regular no es práctica en servidores compartidos o de producción), mientras que la segunda ocupa mucho tiempo y recursos del sistema.

Sin embargo, PEAR ofrece una tercera opción más sencilla: el paquete Net_POP3 en http://pear.php.net/package/Net_POP3, que proporciona una API ya hecha para conectarse a un servidor de correo que cumpla con POP3 e interactuar con él. El uso de este paquete ofrece dos importantes beneficios: en primer lugar, puede aplicarse de inmediato a un script PHP, sin necesidad de hacer cambios en el servidor; y en segundo lugar, proporciona una API de alto nivel que ya tiene integrada toda la funcionalidad requerida, incluido el manejo de errores, con lo que se reduce de manera importante el total del tiempo de desarrollo requerido para el proyecto.

He aquí un ejemplo del código (*pop3.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
   <title>Proyecto 12-1: Accesar buzones electrónicos POP3 con PEAR</title>
  </head>
  <body>
    <h2>Proyecto 12-1: Accesar buzones electrónicos POP3 con PEAR</h2>
    <?php
    // si el formulario ya ha sido enviado
    if (isset($_POST['submit'])) {

      // crea excepciones
      class InputException extends Exception {}
      class ConnException extends Exception {}

      // obtiene los datos de entrada del formulario
      $host = $_POST['host'];
```

```
$puerto = $_POST['puerto'];
$usuario = $_POST['usuario'];
$clave = $_POST['clave'];

try {
// valida los datos de entrada del formulario
if (empty($host)) {
  throw new InputException('Nombre del host perdido');
}
if (empty($puerto)) {
  throw new InputException('Puerto perdido');
}
if (empty($usuario)) {
  throw new InputException('Nombre de usuario perdido');
}
if (empty($clave)) {
  throw new InputException('Contraseña perdida');
}

// crea objeto
require_once 'Net/POP3.php';
$pop3 =& new Net_POP3();

// establece conexión con el host
if (PEAR::isError($ret = $pop3->connect($host, $puerto))) {
  throw new ConnException($ret->getMessage());
}

// inicio de sesión
if (PEAR::isError($ret = $pop3->login($usuario, $clave, 'USER'))) {
  throw new ConnException($ret->getMessage());
}

// obtiene la cantidad de mensajes y el tamaño del buzón electrónico
echo $pop3->numMsg() . ' mensajes en el buzón, ' . $pop3->getSize()
. 'bytes <p/>';

// obtiene encabezados de los mensajes más recientes
if ($pop3->numMsg() > 0) {
  $msgData = $pop3->getParsedHeaders($pop3->numMsg());
  echo 'Mensajes más recientes de ' . htmlentities($msgData['From']
) . ',subject\'' , htmlentities($msgData['Subject']) . '\'';
}

// desconecta
$pop3->disconnect();

} catch (InputException $e) {
  die ('Error en la validación de datos de entrada: ' . $e->getMessage());
} catch (ConnException $e) {
  die ('Error en la conexión: El servidor dice' . $e->getMessage());
} catch (Exception $e) {
  die ('ERROR: ' . $e->getMessage());
```

(*continúa*)

```
      }
    } else {
    ?>
    <form method="post" action="pop3.php">
      Nombre del servidor: <br />
      <input type="text" size="20" name="host" />
      <p>
      Puerto del servidor: <br />
      <input type="text" size="4" name="puerto" value="110" />
      <p>
      Nombre de usuario: <br />
      <input type="text" size="20" name="usuario" />
      <p>
      Contraseña: <br />
      <input type="password" size="10" name="clave" />
      <p>
      <input type="submit" name="submit" value="Enviar" />
    </form>
    <?php
    }
    ?>
  </body>
</html>
```

Este script comienza generando un sencillo formulario Web para ingresar la información sobre el servidor de correo electrónico del usuario (figura 12-2). Una vez que el formulario

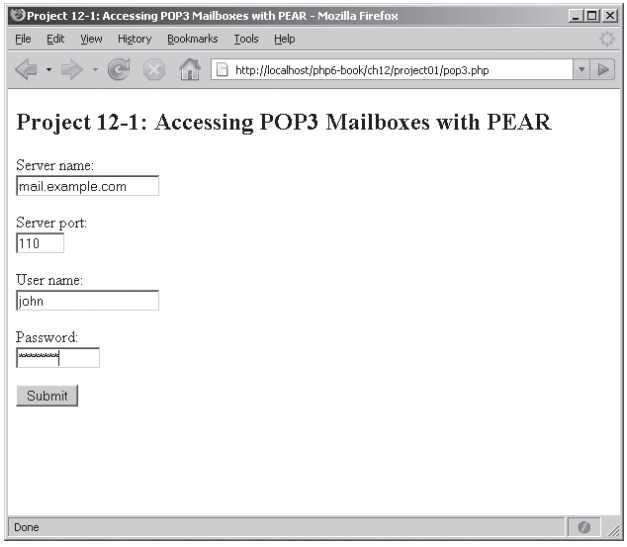


Figura 12-2 Formulario Web para ingresar los datos del servidor de correo electrónico

es enviado, los datos ingresados en él son validados y en caso de que cualquiera de ellos no sea válido, se lanzan las respectivas excepciones. A continuación se carga la clase Net_POP3 y se inicializa una instancia de la misma. La clase expone el método `connect()`; cuando pasa el nombre del servidor POP3 y el puerto como argumentos, este método intenta abrir una conexión al servidor POP3 especificado. Cuando se establece la conexión, el método `login()` de la misma clase es utilizado para ingresar al servidor y acceder al buzón electrónico del usuario, utilizando los datos de nombre de usuario y contraseña proporcionados por el formulario. Después de ingresar con éxito, se dispone de cierta cantidad de métodos útiles para interactuar con el buzón, por ejemplo: el método `numMsg()` regresa la cantidad de mensajes existentes en el buzón, mientras que el método `getSize()` regresa el tamaño del buzón en bytes. Cuando es posible, también se recuperan el tema y el remitente del mensaje más reciente utilizando el método `getParsedHeaders()`, que regresa una matriz con los encabezados del mensaje. Una vez que esta información ha sido recolectada y presentada, el método `disconnect()` se utiliza para dar por terminada la conexión con el servidor.

La figura 12-3 muestra el resultado de una conexión correcta.

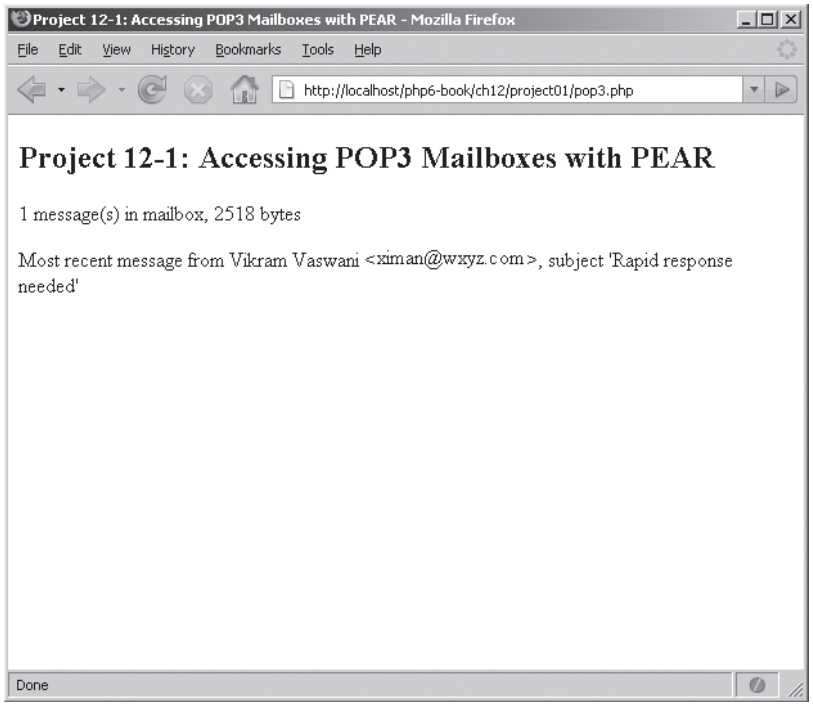


Figura 12-3 Información recuperada de un servidor POP3 con Net_POP3

Utilizar PECL

PECL es la biblioteca comunitaria de extensiones PHP, disponible en el sitio Web http://pecl.php.net/. PECL busca expandir las capacidades de PHP a través de módulos de lenguaje de bajo nivel, escritos en lenguaje de programación C; casi todos ellos deben integrarse directamente en el motor PHP (por lo general compilándolos). Las extensiones PECL se distribuyen como archivos comprimidos TAR y pueden instalarse en el sistema de desarrollo ya sea a través de un proceso manual de compilación e instalación o con el instalador PECL (incluido con cada distribución de PHP).

Instalar extensiones PECL

El procedimiento para instalar las extensiones PECL es diferente en Windows y UNIX. Para bajar, compilar e instalar extensiones en UNIX, todo en uno, se ejecuta el siguiente comando en la consola:

```
shell> pecl install nombre-de-la-extensión
```

El instalador PECL se conectará ahora con el servidor PECL, bajará el código fuente, lo compilará y lo instalará en la ubicación apropiada de tu sistema. La figura 12-4 muestra un ejemplo de instalación de las extensiones Zip, disponibles en http://pecl.php.net/package/zip.

```
192.168.0.9 - PuTTY
-build-root/install-zip-1.8.10
166335    4 drwxr-xr-x   3 root     root         4096 Mar 24 22:05 /var/tmp/pear
-build-root/install-zip-1.8.10/usr
166336    4 drwxr-xr-x   3 root     root         4096 Mar 24 22:05 /var/tmp/pear
-build-root/install-zip-1.8.10/usr/local
166337    4 drwxr-xr-x   3 root     root         4096 Mar 24 22:05 /var/tmp/pear
-build-root/install-zip-1.8.10/usr/local/lib
166338    4 drwxr-xr-x   3 root     root         4096 Mar 24 22:05 /var/tmp/pear
-build-root/install-zip-1.8.10/usr/local/lib/php
166339    4 drwxr-xr-x   3 root     root         4096 Mar 24 22:05 /var/tmp/pear
-build-root/install-zip-1.8.10/usr/local/lib/php/extensions
166340    4 drwxr-xr-x   2 root     root         4096 Mar 24 22:05 /var/tmp/pear
-build-root/install-zip-1.8.10/usr/local/lib/php/extensions/no-debug-non-zts-200
60613
166341  292 -rwxr-xr-x   1 root     root       291000 Mar 24 22:05 /var/tmp/pear
-build-root/install-zip-1.8.10/usr/local/lib/php/extensions/no-debug-non-zts-200
60613/zip.so

Build process completed successfully
Installing '/usr/local/lib/php/extensions/no-debug-non-zts-20060613/zip.so'
install ok: channel://pecl.php.net/zip-1.8.10
configuration option "php_ini" is not set to php.ini location
You should add "extension=zip.so" to php.ini
root@achilles:/tmp#
```

Figura 12-4 Instalación de las extensiones PECL en UNIX

Como opción, es posible bajar el código fuente y compilarlo manualmente en un módulo PHP:

```
shell# cd zip-1.8.10
shell# phpize
shell# ./configure
shell# make
shell# make install
```

Este proceso debe generar un módulo PHP cargable llamado *zip.so* y copiarlo en el directorio de extensiones PHP. Ahora debe habilitarse en el archivo de configuración *php.ini*.

Para los usuarios de Windows es más sencillo: sólo necesitan descargar una extensión PECL precompilada, copiarla al directorio de extensiones PHP y luego activar la extensión en el archivo de configuración *php.ini*. Las extensiones PECL precompiladas para Windows están disponibles en http://pecl4win.php.net/.

Una vez que las extensiones han sido correctamente instaladas y activadas, reinicia el servidor Web y verifica los datos de salida del comando `phpinfo()`. Si las extensiones han sido instaladas correctamente, `phpinfo()` las mostrará como se aprecia en la figura 12-5.

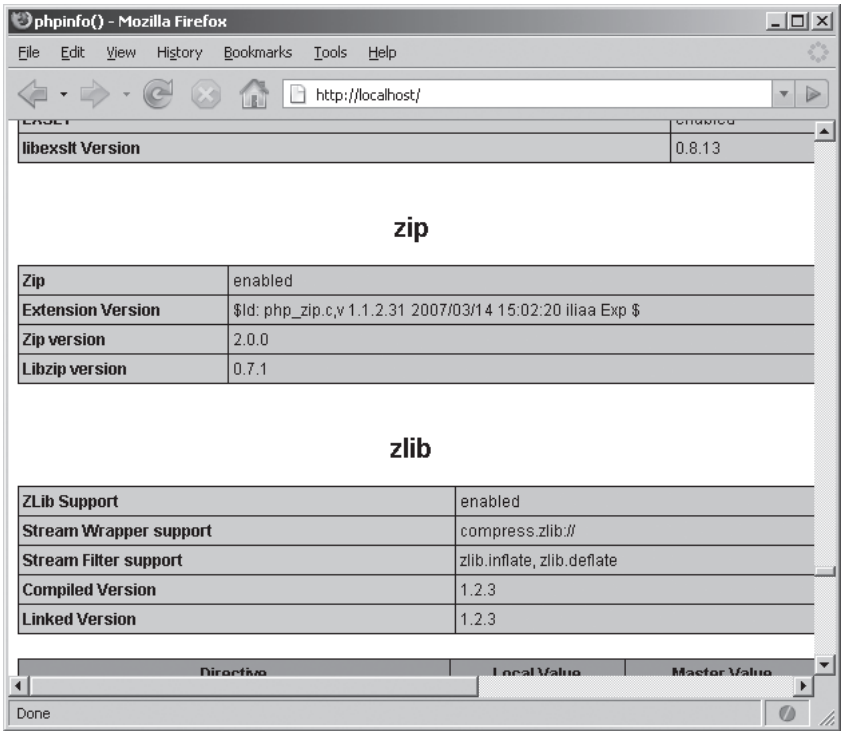


Figura 12-5 Los datos de salida del comando phpinfo(), mostrando las extensiones PECL

TIP

Puedes encontrar más información sobre la instalación de las extensiones PECL en el manual de PHP, en www.php.net/manual/install.pecl.php

Prueba esto 12-2 Crear archivos Zip con PECL

Pongamos otro ejemplo: crear un archivo comprimido Zip. Esto no es algo que normalmente puedas hacer con PHP, porque la construcción original de PHP no incluye soporte para archivos Zip. PECL viene al rescate con su extensión Zip, que proporciona una API completa para leer y escribir archivos Zip.

He aquí el ejemplo en acción (*zip.php*):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Proyecto 12-2: Crear archivos Zip con PECL</title>
  </head>
  <body>
    <h2>Proyecto 12-2: Crear archivos Zip con PECL</h2>
    <?php
      // incrementa el tiempo de ejecución del script
      ini_set('max_execution_time', 300);

      // crea un objeto
      $zip = new ZipArchive();

      // abre archivo
      if ($zip->open('mi-archivo.zip', ZIPARCHIVE::CREATE) !== TRUE) {
          die ("ERROR: No fue posible abrir archivo.");
      }

      // inicializa un reiterador
      // pasa el directorio que debe ser procesado
      $iterator = new RecursiveIteratorIterator(new RecursiveDirectory
Iterator("app/"));

      // hace reiteraciones sobre el directorio
      // añade cada archivo encontrado a archive
      foreach($iterator as $key->$value) {
        $zip->addFile(realpath($key), $key) or die ("ERROR: No fue
posible añadir archivo: $key");
        echo "Añadiendo archivo $key…<br />";
      }

      // cierra y guarda archivo
```

```
      $zip->close();
      echo "Archivo creado con éxito.";
    ?>
  </body>
</html>
```

Este código muestra cómo se puede utilizar la extensión Zip de PECL para añadir soporte Zip a PHP. El script comienza creando una instancia del objeto ZipArchive; este objeto sirve de punto de entrada para todas las funciones de la extensión Zip. A continuación, el método `open()` del mismo objeto se utiliza para crear un nuevo archivo, y el método `addFile()` se usa después, en combinación con un RecursiveDirectoryIterator, para iterar sobre el directorio *app/* y sus directorios secundarios, añadiendo todos los que encuentra al archivo Zip. Una vez que todos los archivos se han añadido, el método `close()` del mismo objeto se encarga de realizar la compresión y escribir el archivo final en disco.

La figura 12-6 muestra el resultado.

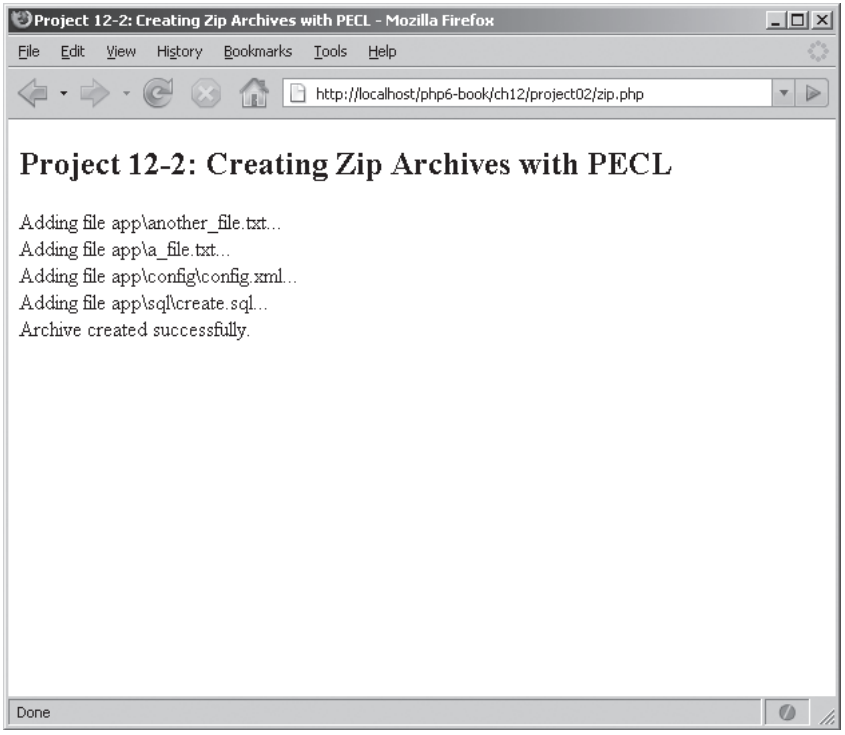


Figura 12-6 Creación dinámica de un archivo Zip con PHP

Resumen

Aunque ya has llegado al capítulo final de este libro, debe quedarte claro que tu viaje por PHP está muy lejos de haber concluido..., esto es sólo el principio. Tanto PEAR como PECL son depósitos muy populares que albergan programas auxiliares de alta calidad y estables para PHP y su contenido crece constantemente. Este capítulo mostró sólo dos de los cientos de programas auxiliares disponibles: una biblioteca cliente POP3 y un módulo para archivos comprimidos en formato Zip.

Si alguna vez te encuentras con algún problema de desarrollo difícil, ocupa unos momentos buscando en estos depósitos; existe una gran probabilidad de que encuentres la solución ya hecha en alguno de ellos, lo que te ahorrará muchas horas de trabajo. Mientras tanto, los siguientes vínculos te ayudarán a aprender más sobre los temas tratados en este capítulo:

- El sitio Web PEAR, en **http://pear.php.net**
- El sitio Web PECL, en **http://pecl.php.net**
- Notas de instalación de PEAR, en **http://pear.php.net/manual/en/installation.php**
- Notas de instalación de PECL, en **http://php.net/manual/install.pecl.php**
- Documentación para el paquete Net_POP3, en **http://pear.php.net/manual/en/package.networking.net-pop3.php**
- Documentación para las extensiones PECL Zip, en **www.php.net/zip**

Has llegado al final del libro. Espero que lo hayas disfrutado y encontrado útil, y que ahora tengas las bases necesarias para salir y crear tus propias aplicaciones PHP de alta calidad. ¡Buena suerte y feliz codificación!

Autoexamen Capítulo 12

1. ¿Cuál es la diferencia entre PEAR y PECL?
2. Describe los pasos para instalar una extensión PECL en el ambiente de desarrollo Windows.
3. Revisa la documentación utilizada para el paquete Net_POP3 utilizado en este capítulo y reescribe el código para que muestre el contenido completo del mensaje más reciente.
4. Descarga e instala la extensión ID3 de PECL para trabajar con archivos MP3, y luego escribe una aplicación PHP para procesar un directorio de archivos MP3, imprimiendo el título de la canción y el artista que la interpreta. (Pista: el paquete PECL está disponible en http:// pecl.php.net/package/id3, y la documentación está disponible en www.php.net/id3.)

Parte IV

Apéndices

Apéndice A

Instalar y configurar los programas requeridos

Habilidades y conceptos clave

- Aprender a obtener e instalar el software MySQL, SQLite, PHP y Apache de Internet
- Realizar pruebas básicas para asegurar que la aplicación funcione correctamente
- Descubrir la manera de activar automáticamente todos los componentes requeridos cuando arranque el sistema
- Dar los pasos básicos para salvaguardar la seguridad de tu instalación de MySQL

En este libro has aprendido sobre el lenguaje de programación PHP y cómo puede ser utilizado para construir aplicaciones Web sofisticadas. Algunos de los ejemplos de este libro también incluyen el uso de PHP con componentes de terceros, como XML, MySQL y SQLite. Este apéndice te enseña a instalar y configurar estos componentes en tu estación de trabajo, además de la manera de crear un ambiente de desarrollo que puede ser utilizado para ejecutar el código presentado en este libro.

PRECAUCIÓN

La intención de este apéndice es proporcionar un panorama general sobre el proceso de instalación y configuración de MySQL, SQLite, PHP y Apache bajo UNIX y Windows. *No intenta ser* un sustituto para la documentación de instalación que acompaña cada uno de los paquetes mencionados. Si encuentras dificultades para instalar los paquetes que aquí se mencionan, visita su respectivo sitio Web o busca información en Web sobre el manejo de errores y su posible corrección (algunos sitios se mencionan al final de este apéndice).

Obtener el software

El primer paso consiste en asegurar que tienes todos los programas necesarios. He aquí la lista:

- **PHP** PHP proporciona un conjunto de herramientas para desarrollo de aplicaciones para Web y de consola. Puede ser descargado de www.php.net/. En el mismo sitio encontrarás la versión en código fuente como la binaria para las plataformas Windows, UNIX y Mac OS X. Los usuarios de UNIX deben descargar la versión en código fuente más reciente, mientras que los de Windows deben descargar la última versión en formato binario. En el momento en que este libro se imprimió, la versión más reciente de PHP era la 5.3.0 alpha 1.

- **Apache** Apache es un servidor Web rico en características que trabaja bien con PHP. Puede ser descargado gratuitamente en http://httpd.apache.org/ como código fuente y en formato binario, para diferentes plataformas. Los usuarios de UNIX deben descargar el código fuente más reciente, mientras que los de Windows deben descargar el instalador binario adecuado para la versión de Windows que manejen. En el momento en que este libro se imprimió, la versión más reciente del servidor Apache era la 2.2.9.
- **MySQL** El servidor de base de datos MySQL es un sistema para almacenamiento y recuperación de datos robusto y escalable. Está disponible en código fuente y versión binaria en www.mysql.com/. Las versiones binarias están disponibles para Linux, Solaris, FreeBSD, Mac OS X, Windows, HP-UX, IBM AIX, SCO OpenUNIX y SGI Irix, mientras que el código está disponible para las plataformas Windows y UNIX. La versión binaria es la más recomendable por dos razones: es más fácil de instalar y el equipo de desarrolladores de MySQL la ha optimizado para las diferentes plataformas. En el momento en que este libro se imprimió, la versión más reciente de la base de datos MySQL era la 5.0.67.
- **SQLite** SQLite es una base de datos independiente significativamente más pequeña que MySQL. Está disponible tanto en código fuente como en formato binario en www.sqlite.org/, para las plataformas Windows, UNIX y Mac OS X. Los usuarios de Windows y UNIX deben descargar la versión binaria (una liga hacia la descarga está disponible en la página Web compañera de este libro). En el momento en que este libro se imprimió, la versión más reciente de la base de datos SQLite era la 3.6.1. Sin embargo, como SQLite 3.x soporta PHP 5.3, todavía es una versión experimental en el momento de la publicación de este libro, por lo que en los ejemplos se utilizó SQLite 2.x.

Además de estos cuatro componentes básicos, los usuarios de UNIX pueden requerir algunas bibliotecas de soporte. He aquí la lista:

- La biblioteca `libxml2`, disponible en www.xmlsoft.org/
- La biblioteca `zlib`, disponible en www.gzip.org/zlib/

Por último, los usuarios de ambas plataformas necesitarán una herramienta de descompresión capaz de manejar archivos TAR (archivos Tape) y GZ (GNU Zip). En UNIX, las utilidades *tar* y *gzip* son apropiadas y suelen incluirse en el sistema operativo. En Windows, una buena herramienta para descomprimir es WinZip, disponible en www.winzip.com/.

NOTA

Los ejemplos de este libro han sido desarrollados y probados en SQLite 2.8.17, MySQL 5.0.67, con Apache 2.2.9 y PHP 5.2.5 y 5.3.0 alpha1.

Instalar y configurar los programas

Una vez que se han obtenido los programas necesarios, el paso siguiente consiste en instalar las diferentes piezas y hacer que se comuniquen entre sí. La siguiente sección delinea los pasos a seguir para las plataformas Windows y UNIX.

NOTA

Si estás utilizando una computadora Apple, encontrarás las instrucciones para instalar PHP en Mac OS X en el manual PHP: www.php.net/manual/en/install.macosx.php

Instalar en UNIX

El proceso de instalación en UNIX incluye diferentes pasos: instalar MySQL a partir del formato binario; compilar e instalar PHP del código fuente, y compilar y configurar Apache para que maneje correctamente las solicitudes de páginas Web PHP. Estos pasos son descritos con gran detalle en las siguientes subsecciones.

Instalar MySQL

Para instalar MySQL a partir del formato binario, sigue estos pasos:

1. Asegúrate de haber ingresado al sistema como usuario "root".

```
[user@host]# su - root
```

2. Extrae el contenido del archivo binario de MySQL en el directorio apropiado de tu sistema, por ejemplo: */usr/local/*.

```
[root@host]# cd/usr/local
[root@host]# tar -xzvf /tmp/mysql-5.0.67-linuxi686.tar.gz
```

   Los archivos de MySQL deben extraerse en un directorio cuyo nombre esté de acuerdo con el formato *mysql-versión-so-arquitectura*, por ejemplo: *mysql-5.0.67-linux-i686*.

3. Para facilitar el uso, establece un nombre corto para el directorio creado en el paso anterior, creando un vínculo suave llamado *mysql* que apunte a este directorio en la misma ubicación.

```
[root@host]# ln -s mysql-5.0.67-linux-i686 mysql
```

4. Por razones de seguridad, nunca debe ejecutarse el servidor de base de datos MySQL como superusuario. Por lo tanto, es necesario crear un usuario especial "mysql" y un grupo para este propósito. Realiza esta tarea con los comandos `groupadd` y `useradd`, y luego cambia la propiedad del directorio de instalación de MySQL a los usuarios y grupos recién creados:

```
[root@host]# groupadd mysql
[root@host]# useradd -g mysql mysql
[root@host]# chown -R mysql /usr/local/mysql
[root@host]# chgprp -R mysql /usr/local/mysql
```

5. Inicializa las tablas de MySQL con el script de inicialización *mysql_install_db* que se incluye con el paquete de distribución.

```
[root@host]# /usr/local/mysql/scripts/mysql_install_db --user=mysql
```

La figura A-1 muestra lo que verás cuando aplicas el comando.

Como lo sugieren los datos de salida anteriores, este script de inicialización prepara e instala las diferentes tablas de la base de datos MySQL, y también establece los permisos por defecto para el acceso a la base.

6. Altera la propiedad de los binarios de MySQL para que el propietario sea "root":

```
[root@host]# chown -R root /usr/local/mysql
```

y asegúrate de que el usuario "mysql" creado en el paso 4 tenga permisos de lectura y escritura al directorio de datos MySQL.

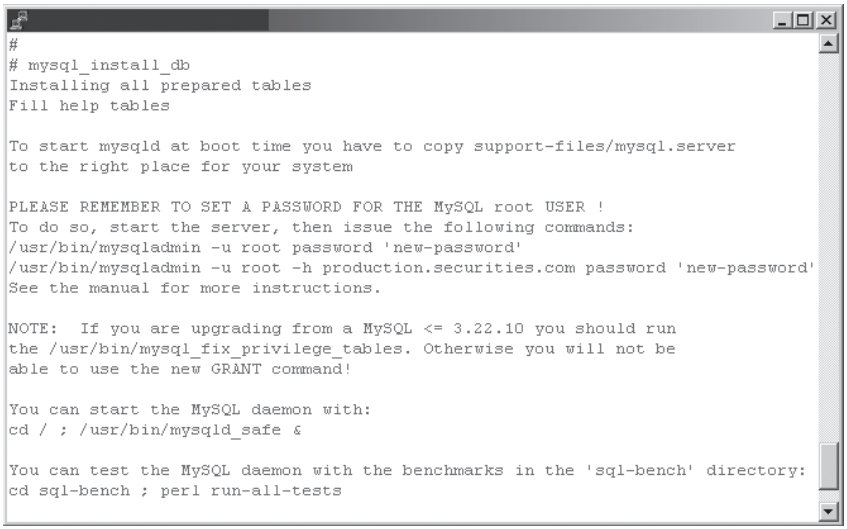
```
[root@host]# chown -R mysql /usr/local/mysql/data
```

7. Arranca el servidor MySQL ejecutando manualmente el script *mysqld_safe*.

```
[root@host]# /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

Ahora MySQL debe iniciar de manera normal.

Una vez que la instalación se ha completado correctamente y el servidor ha arrancado, pasa a la sección "Probar MySQL" para verificar que funciona como debe.



```
#
# mysql_install_db
Installing all prepared tables
Fill help tables

To start mysqld at boot time you have to copy support-files/mysql.server
to the right place for your system

PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !
To do so, start the server, then issue the following commands:
/usr/bin/mysqladmin -u root password 'new-password'
/usr/bin/mysqladmin -u root -h production.securities.com password 'new-password'
See the manual for more instructions.

NOTE:  If you are upgrading from a MySQL <= 3.22.10 you should run
the /usr/bin/mysql_fix_privilege_tables. Otherwise you will not be
able to use the new GRANT command!

You can start the MySQL daemon with:
cd / ; /usr/bin/mysqld_safe &

You can test the MySQL daemon with the benchmarks in the 'sql-bench' directory:
cd sql-bench ; perl run-all-tests
```

Figura A-1 Los datos de salida del script *mysql_install_db*

Instalar Apache y PHP

PHP puede integrarse con el servidor Web Apache de dos maneras: como un módulo dinámico cargado dentro del servidor Web en tiempo de ejecución, o como un módulo estático que está integrado al código fuente de Apache en tiempo de construcción. Cada opción tiene ventajas y desventajas:

- Instalar PHP como un módulo dinámico facilita la actualización del motor PHP más adelante, porque sólo necesita volver a compilar el módulo PHP y no el resto del servidor Web Apache. Por otra parte, con un módulo cargado dinámicamente, el rendimiento tiende a ser más lento en comparación con un módulo estático, que está más integrado al servidor.
- Instalar PHP como un módulo estático mejora el rendimiento, porque dicho módulo está compilado directamente en el código fuente del servidor Web. Sin embargo, esta integración cercana tiene una importante desventaja: si decides actualizar el motor PHP, necesitarás reintegrar el nuevo módulo PHP al código fuente de Apache y volver a compilar el servidor Web.

Esta sección muestra cómo compilar PHP como módulo dinámico que se carga en el servidor Apache en tiempo de ejecución.

1. Asegúrate de haber ingresado al sistema como usuario "root".

```
[user@host]# su - root
```

2. Extrae el contenido del archivo fuente de Apache en el directorio temporal de tu sistema.

```
[root@host]# cd/tmp
[root@host]# tar -xzvf /tmp/httpd-2.2.9.tar.gz
```

3. Para permitir que PHP se cargue dinámicamente, el servidor Apache debe ser compilado con soporte para compartir objetos dinámicamente (DSO). Este soporte se activa con la opción `--enable-so`, transmitida al script de configuración *configure* del servidor Apache, como se muestra a continuación:

```
[root@host]# cd /tmp/httpd-2.2.9
[root@host]# ./configure --prefix=/usr/local/apache --enable-so
```

   Aparecerán algunas pantallas con datos de salida (la figura A-2 muestra un ejemplo), mientras que el script *configure* establece las variables necesarias para el proceso de compilación.

4. Ahora, compila el servidor utilizando `make`, e instálalo en tu sistema utilizando `make install`.

```
[root@host]# make
[root@host]# make install
```

   La figura A-3 muestra lo que probablemente verás durante el proceso de compilación. Ahora, Apache ya debe estar instalado en */usr/local/apache/*.

```
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
Applying APR hints file rules for i686-pc-linux-gnu
  setting CPPFLAGS to "-DLINUX=2"
  adding "-D_REENTRANT" to CPPFLAGS
  adding "-D_GNU_SOURCE" to CPPFLAGS
(Default will be unix)
checking whether make sets $(MAKE)... yes
checking how to run the C preprocessor... gcc -E
checking for gawk... gawk
checking whether ln -s works... yes
checking for ranlib... ranlib
checking for a BSD-compatible install... /usr/bin/install -c
checking for rm... rm
checking for as... as
checking for cpp... cpp
checking for ar... ar
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for ANSI C header files...
```

Figura A-2 Configuración del árbol fuente Apache

```
pthread   -DHAVE_CONFIG_H -DLINUX=2 -D_REENTRANT -D_GNU_SOURCE -D_LARGEFILE64_SO
URCE   -I./include -I/tmp/httpd-2.2.9/srclib/apr/include/arch/unix -I./include/a
rch/unix -I/tmp/httpd-2.2.9/srclib/apr/include/arch/unix -I/tmp/httpd-2.2.9/srcl
ib/apr/include  -o strings/apr_cpystrn.lo -c strings/apr_cpystrn.c && touch stri
ngs/apr_cpystrn.lo
/bin/sh /tmp/httpd-2.2.9/srclib/apr/libtool --silent --mode=compile gcc -g -O2 -
pthread   -DHAVE_CONFIG_H -DLINUX=2 -D_REENTRANT -D_GNU_SOURCE -D_LARGEFILE64_SO
URCE   -I./include -I/tmp/httpd-2.2.9/srclib/apr/include/arch/unix -I./include/a
rch/unix -I/tmp/httpd-2.2.9/srclib/apr/include/arch/unix -I/tmp/httpd-2.2.9/srcl
ib/apr/include  -o strings/apr_fnmatch.lo -c strings/apr_fnmatch.c && touch stri
ngs/apr_fnmatch.lo
/bin/sh /tmp/httpd-2.2.9/srclib/apr/libtool --silent --mode=compile gcc -g -O2 -
pthread   -DHAVE_CONFIG_H -DLINUX=2 -D_REENTRANT -D_GNU_SOURCE -D_LARGEFILE64_SO
URCE   -I./include -I/tmp/httpd-2.2.9/srclib/apr/include/arch/unix -I./include/a
rch/unix -I/tmp/httpd-2.2.9/srclib/apr/include/arch/unix -I/tmp/httpd-2.2.9/srcl
ib/apr/include  -o strings/apr_snprintf.lo -c strings/apr_snprintf.c && touch st
rings/apr_snprintf.lo
/bin/sh /tmp/httpd-2.2.9/srclib/apr/libtool --silent --mode=compile gcc -g -O2 -
pthread   -DHAVE_CONFIG_H -DLINUX=2 -D_REENTRANT -D_GNU_SOURCE -D_LARGEFILE64_SO
URCE   -I./include -I/tmp/httpd-2.2.9/srclib/apr/include/arch/unix -I./include/a
rch/unix -I/tmp/httpd-2.2.9/srclib/apr/include/arch/unix -I/tmp/httpd-2.2.9/srcl
ib/apr/include  -o strings/apr_strings.lo -c strings/apr_strings.c && touch stri
ngs/apr_strings.lo
```

Figura A-3 Compilación de Apache

5. Ahora compila e instala PHP. Comienza por extraer el contenido del archivo fuente de PHP en el directorio temporal de tu sistema.

```
[root@host]# cd /tmp
[root@host]# tar -xzvf /tmp/php-5.3.0.tar.gz
```

6. Este paso es el más importante en el proceso de instalación de PHP. Implica enviar argumentos al script *configure* para configurar el módulo PHP. Estos parámetros en líneas de comando especifican las extensiones PHP que serán activadas, y también le indican a PHP dónde encontrar las bibliotecas de soporte necesarias para esas extensiones.

```
[root@host]# cd /tmp/php-5.3.0
[root@host]# ./configure -prefix=/usr/local/php --with-apx2=/usr/
local/apache/bin/apxs --with-zlib --with-mysqli=mysqlnd --with-pdo-
mysql=mysqlnd
```

He aquí una breve explicación de lo que hace cada uno de estos argumentos:

- El argumento `--with-apxs2` le indica a PHP dónde encontrar el script APXS (APache eXtenSion) de Apache. Este script simplifica la tarea de construir e instalar los módulos descargables de Apache.
- El argumento `--with-zlib` le indica a PHP que active las características de compresión (Zip), que son utilizadas por diferentes servicios PHP.
- El argumento `--with-mysqli` activa la extensión PHP MySQLi y le indica a PHP que utilice el Controlador Nativo MySQL (mysqlnd).
- El argumento --`with-pdo-mysql` activa el controlador MySQL PDO y le indica a PHP que utilice el Controlador Nativo MySQL (mysqlnd).

La figura A-4 muestra lo que verás durante el proceso de configuración.

TIP

El proceso de configuración de PHP es muy complejo, te permite controlar muchos aspectos del comportamiento de PHP. Para ver una lista completa de las opciones disponibles, usa el comando `configure --help`, y visita la página www.php.net/manual/en/configure.php para explicaciones detalladas de lo que hace cada una de estas opciones.

7. A continuación, compila e instala PHP utilizando `make` y `make install`:

```
[root@host]# make
[root@host]# make install
```

La figura A-5 muestra lo que probablemente verás durante el proceso de instalación. Ahora, PHP debe estar instalado en */usr/local/php/*.

```
checking for strftime... yes
checking for strptime... yes
checking for strstr... yes
checking for strtok_r... yes
checking for symlink... yes
checking for tempnam... yes
checking for tzset... yes
checking for unlockpt... yes
checking for unsetenv... yes
checking for usleep... yes
checking for nanosleep... yes
checking for utime... yes
checking for vsnprintf... yes
checking for getaddrinfo... yes
checking for strlcat... no
checking for strlcpy... no
checking for getopt... yes
checking whether utime accepts a null argument... yes
checking for working alloca.h... (cached) yes
checking for alloca... yes
checking for declared timezone... yes
checking for type of reentrant time-related functions... POSIX
checking for readdir_r... yes
checking for type of readdir_r...
```

Figura A-4 Configuración del árbol fuente de PHP

```
  -I/usr/include -g -O2  -prefer-non-pic -c /tmp/php-5.3.0/ext/pcre/pcrelib/pcre
_study.c -o ext/pcre/pcrelib/pcre_study.lo
/bin/sh /tmp/php-5.3.0/libtool --silent --preserve-dup-deps --mode=compile gcc -
I/tmp/php-5.3.0/ext/pcre/pcrelib -Iext/pcre/ -I/tmp/php-5.3.0/ext/pcre/ -DPHP_AT
OM_INC -I/tmp/php-5.3.0/include -I/tmp/php-5.3.0/main -I/tmp/php-5.3.0 -I/tmp/ph
p-5.3.0/ext/ereg/regex -I/usr/include/libxml2 -I/tmp/php-5.3.0/ext/date/lib -I/t
mp/php-5.3.0/ext/sqlite3/libsqlite -I/tmp/php-5.3.0/TSRM -I/tmp/php-5.3.0/Zend
  -I/usr/include -g -O2  -prefer-non-pic -c /tmp/php-5.3.0/ext/pcre/pcrelib/pcre
_tables.c -o ext/pcre/pcrelib/pcre_tables.lo
/bin/sh /tmp/php-5.3.0/libtool --silent --preserve-dup-deps --mode=compile gcc -
I/tmp/php-5.3.0/ext/pcre/pcrelib -Iext/pcre/ -I/tmp/php-5.3.0/ext/pcre/ -DPHP_AT
OM_INC -I/tmp/php-5.3.0/include -I/tmp/php-5.3.0/main -I/tmp/php-5.3.0 -I/tmp/ph
p-5.3.0/ext/ereg/regex -I/usr/include/libxml2 -I/tmp/php-5.3.0/ext/date/lib -I/t
mp/php-5.3.0/ext/sqlite3/libsqlite -I/tmp/php-5.3.0/TSRM -I/tmp/php-5.3.0/Zend
  -I/usr/include -g -O2  -prefer-non-pic -c /tmp/php-5.3.0/ext/pcre/pcrelib/pcre
_try_flipped.c -o ext/pcre/pcrelib/pcre_try_flipped.lo
/bin/sh /tmp/php-5.3.0/libtool --silent --preserve-dup-deps --mode=compile gcc -
I/tmp/php-5.3.0/ext/pcre/pcrelib -Iext/pcre/ -I/tmp/php-5.3.0/ext/pcre/ -DPHP_AT
OM_INC -I/tmp/php-5.3.0/include -I/tmp/php-5.3.0/main -I/tmp/php-5.3.0 -I/tmp/ph
p-5.3.0/ext/ereg/regex -I/usr/include/libxml2 -I/tmp/php-5.3.0/ext/date/lib -I/t
mp/php-5.3.0/ext/sqlite3/libsqlite -I/tmp/php-5.3.0/TSRM -I/tmp/php-5.3.0/Zend
  -I/usr/include -g -O2  -prefer-non-pic -c /tmp/php-5.3.0/ext/pcre/pcrelib/pcre
_valid_utf8.c -o ext/pcre/pcrelib/pcre_valid_utf8.lo
```

Figura A-5 Compilación de PHP

8. El paso final en el proceso de instalación consiste en configurar Apache para que reconozca correctamente las solicitudes de páginas PHP. Esto se logra abriendo el archivo de configuración de Apache, *httpd.conf* (puede localizarse en el subdirectorio *conf/* en el directorio de instalación de Apache), con un editor de textos y añadiendo la siguiente línea:

   ```
   AddType application/x-httpd-php .php
   ```

   Guarda los cambios en el archivo. Además, verifica que la siguiente línea aparezca en algún lugar del archivo:

   ```
   LoadModule php5_module libexec/libphp5.so
   ```

9. Arranca el servidor Apache ejecutando manualmente el script *apachectl*.

   ```
   [root@host]# /usr/local/apache/bin/apachectl start
   ```

   Apache debe ejecutarse de manera normal.

Una vez que la instalación se ha completado correctamente y que el servidor ha iniciado, ve a la sección titulada "Probar PHP" para comprobar que todo esté funcionando como es debido.

Pregunta al experto

P: ¿Por qué debo activar manualmente el controlador PDO MySQL, pero no el PDO SQLite en la configuración de PHP?

R: En PHP 5, tanto la extensión SQLite como el controlador PDO SQLite se activan por defecto. Por lo tanto, no hay necesidad de activar manualmente el controlador PDO SQLite durante el proceso de configuración. Sin embargo, todos los demás controladores PDO, incluidos aquellos para MySQL, PostgreSQL y ODBC, requieren activación manual.

Instalar SQLite

Para instalar SQLite a partir de la versión binaria, sigue estos pasos:

1. Asegúrate de haber ingresado al sistema como usuario "root".

   ```
   [user@host]# su - root
   ```

2. Desempaqueta los archivos SQLite binarios en el directorio apropiado de tu sistema (por ejemplo, */usr/local/bin*) y haz el binario ejecutable (figura A-6).

   ```
   [user@host]# cd /usr/local/bin
   [user@host]# gunzip sqlite2-2.8.17.bin.gz
   [user@host]# chmod +x sqlite2-2.8.17.bin
   [user@host]# ln -s sqlite2-2.8.17.bin sqlite
   ```

```
# chmod +x sqlite2-2.8.17.bin
# ln -s sqlite2-2.8.17.bin sqlite
#
```

Figura A-6 Instalación de SQLite

SQLite está instalado y listo para utilizarse. Para comprobarlo, ve a la sección titulada "Probar SQLite".

Instalar en Windows

Compilar aplicaciones en Windows es un proceso desafiante, sobre todo para desarrolladores novatos. Con esto en mente, es recomendable que los usuarios de Windows se concentren en instalar y configurar las versiones binarias precompiladas de MySQL, SQLite, PHP y Apache, en vez de intentar compilarlas desde el código fuente. Estas versiones se descargan desde los sitios Web mencionados en la sección anterior y deben instalarse en el orden que se presenta en las siguientes subsecciones.

Instalar MySQL

La versión binaria de MySQL para Windows incluye un instalador automático, que te permite tener funcionando la base de datos en tu sistema en pocos minutos.

1. Ingresa como administrador (si estás utilizando Windows NT/2000/XP/Vista) y desempaqueta el archivo binario en un directorio temporal de tu sistema.
2. Haz doble clic en el archivo *setup.exe* para iniciar el proceso de instalación. Debes ver una pantalla de bienvenida (figura A-7).
3. Selecciona el tipo de instalación requerido (figura A-8).

   Lo más frecuente es que sea una instalación Typical; sin embargo, si no te agradan las opciones por omisión, o si tienes poco espacio en disco, selecciona la opción Custom y decide qué componentes del paquete deben instalarse.
4. MySQL debe empezar a instalarse en tu sistema (figura A-9).
5. Una vez que se complete la instalación, debes ver una notificación. En este punto tendrás la opción de lanzar el asistente de configuración para el servidor MySQL (MySQL Server Instance Config Wizard), para completar la configuración del programa. Selecciona esta opción y tendrás que ver la pantalla de bienvenida correspondiente (figura A-10).

Figura A-7 Inicio de la instalación de MySQL en Windows

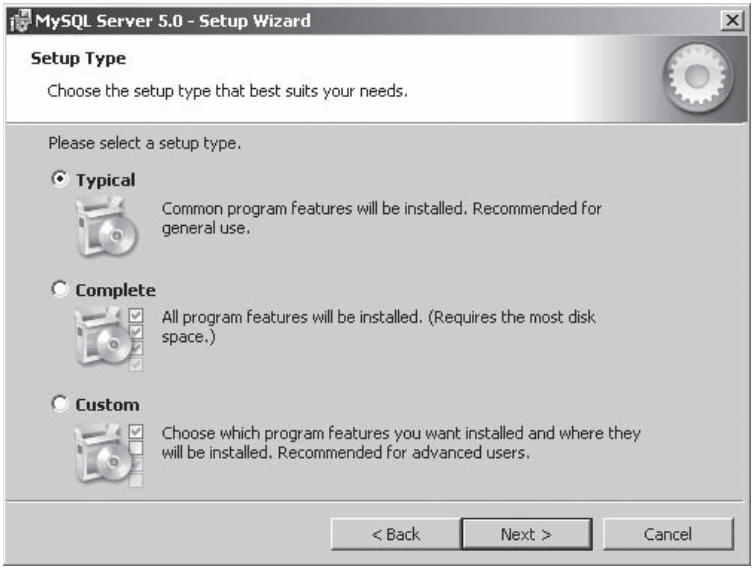


Figura A-8 Selección del tipo de instalación de MySQL

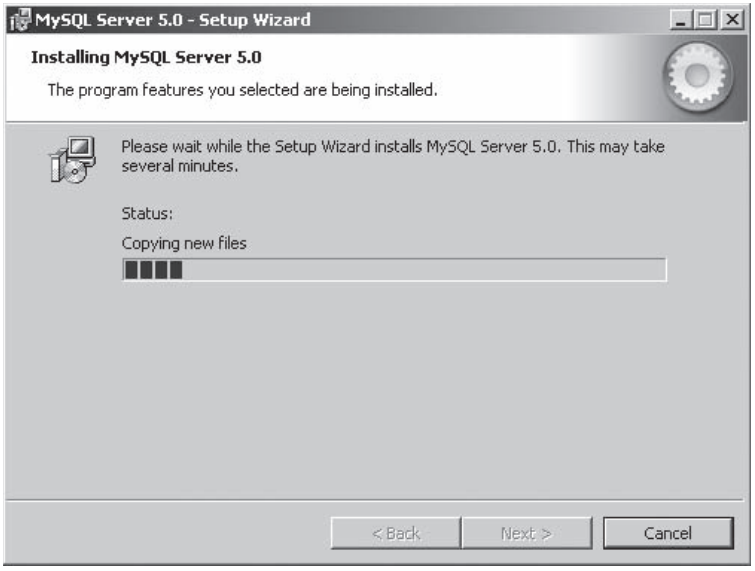


Figura A-9 Progreso de la instalación de MySQL

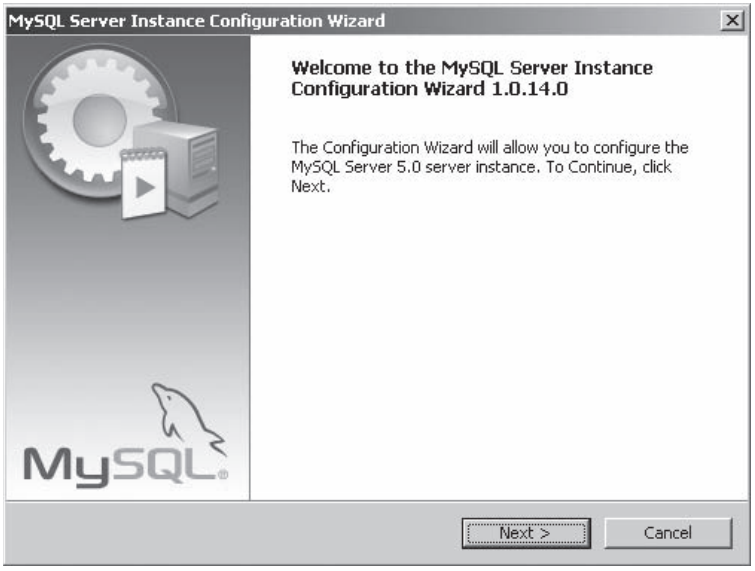


Figura A-10 Inicia la configuración de MySQL en Windows

6. Selecciona el tipo de configuración (figura A-11). En casi todos los casos, la configuración estándar (Standard Configuration) será suficiente.
7. Instala MySQL como un servicio de Windows; de esta manera iniciará y se detendrá automáticamente junto con Windows (figura A-12).
8. Ingresa la clave de acceso para la cuenta del administrador de MySQL ("root") (figura A-13).
9. El servidor ya estará configurado con las opciones especificadas e iniciará automáticamente. Se te presentará una notificación de la instalación correcta una vez que se hayan completado todas las tareas requeridas (figura A-14).

Ahora puedes proceder a probar el servidor como se describe en la sección "Probando MySQL", para asegurar que todo trabaja como debe.

Instalar Apache

Una vez que MySQL está instalado, el siguiente paso es instalar el servidor Web Apache. En Windows, se trata de un proceso de colocar el cursor y hacer clic, similar al utilizado en la instalación de MySQL.

1. Haz doble clic en el instalador Apache para comenzar el proceso de instalación. Debes ver una pantalla de bienvenida (figura A-15).

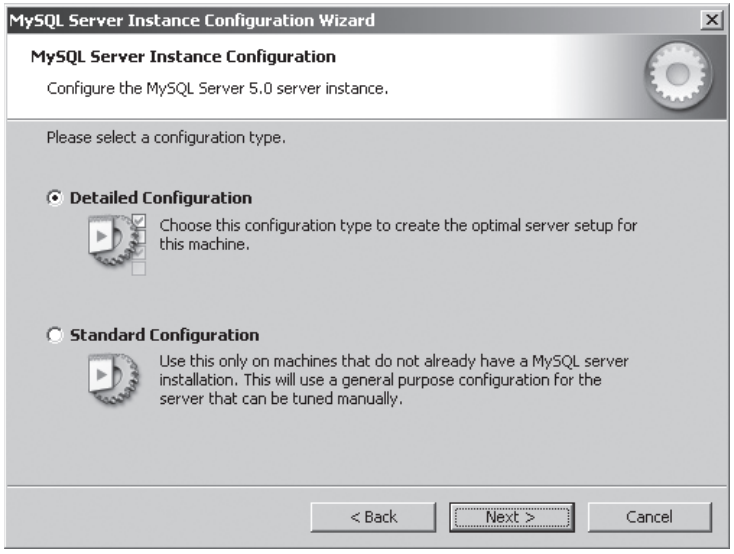


Figura A-11 Seleccionar el tipo de configuración

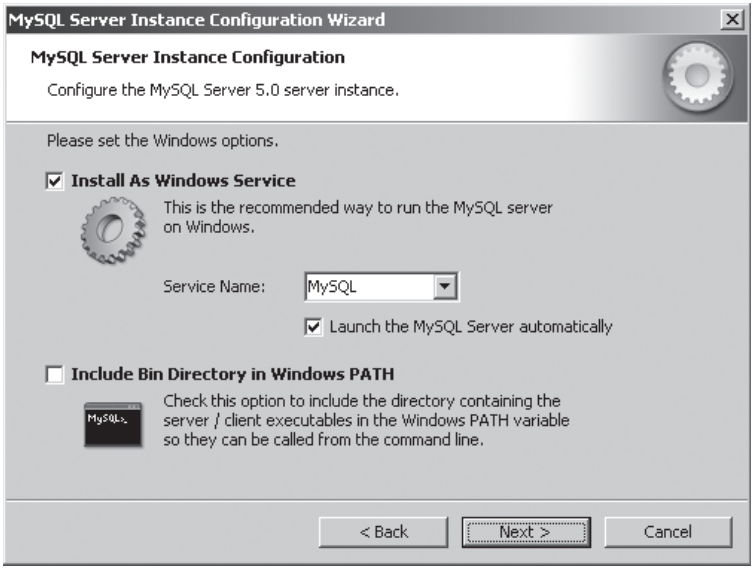


Figura A-12 Establecer el servicio MySQL

Figura A-13 Establecer la contraseña del administrador

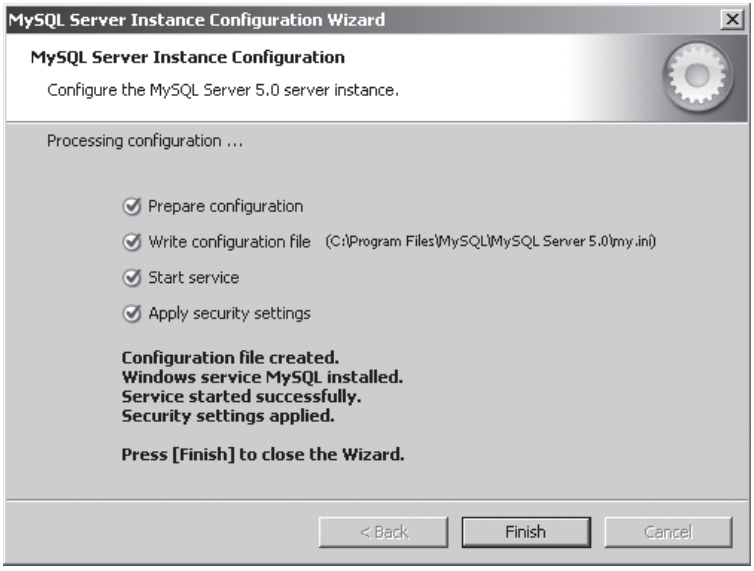


Figura A-14 La configuración de MySQL completada correctamente

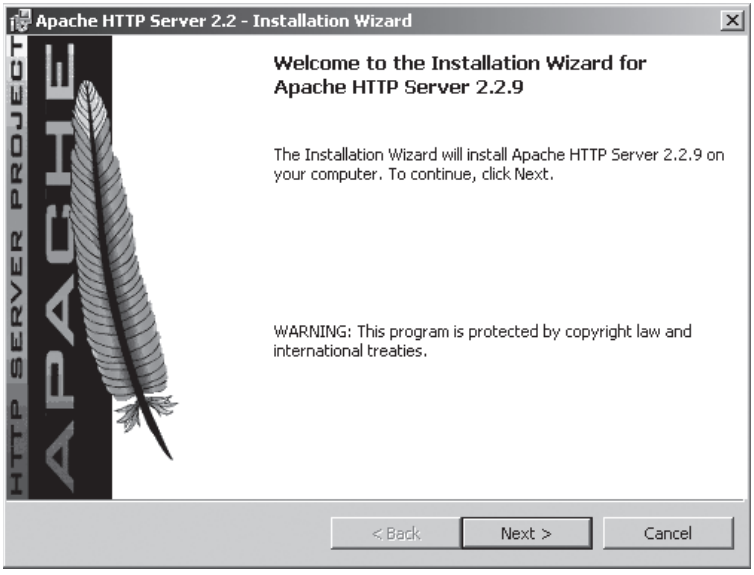


Figura A-15 Principia la instalación de Apache para Windows

2. Lee el acuerdo de licencia y acepta los términos para seguir adelante.
3. Lee la información descriptiva y procede a ingresar la información básica sobre el servidor y la dirección de correo electrónico que será mostrada en las páginas de error (figura A-16).
4. Selecciona el tipo de instalación requerida (figura A-17).

   Puedes seleccionar la opción Custom para decidir qué componentes del paquete deben instalarse.
5. Selecciona la ubicación donde debe instalarse Apache, por ejemplo: *c:\archivos de programa\grupo apache* (figura A-18).
6. Apache debe quedar instalado en la ubicación especificada (figura A-19). El proceso de instalación toma algunos minutos para completarse, así que es buena idea que vayas por una taza de café.
7. Una vez que la instalación esté completada, el instalador Apache mostrará una notificación de éxito e iniciará el servidor Web.

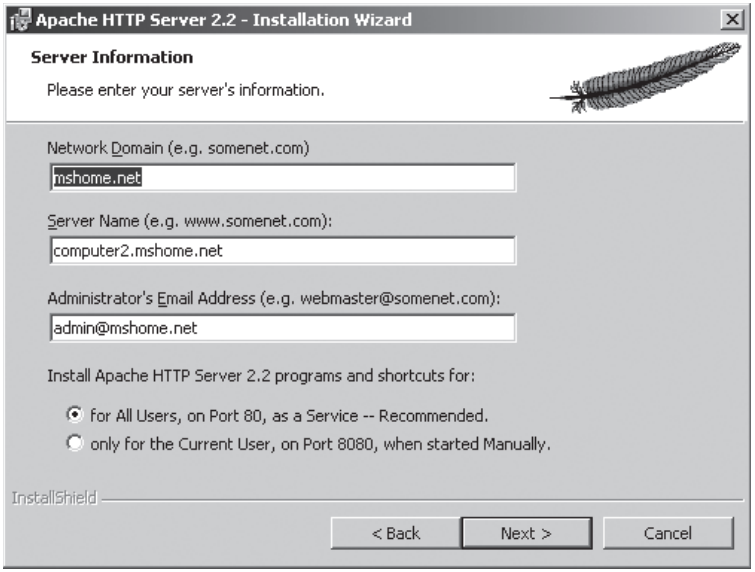


Figura A-16 Ingreso de la información del servidor Apache

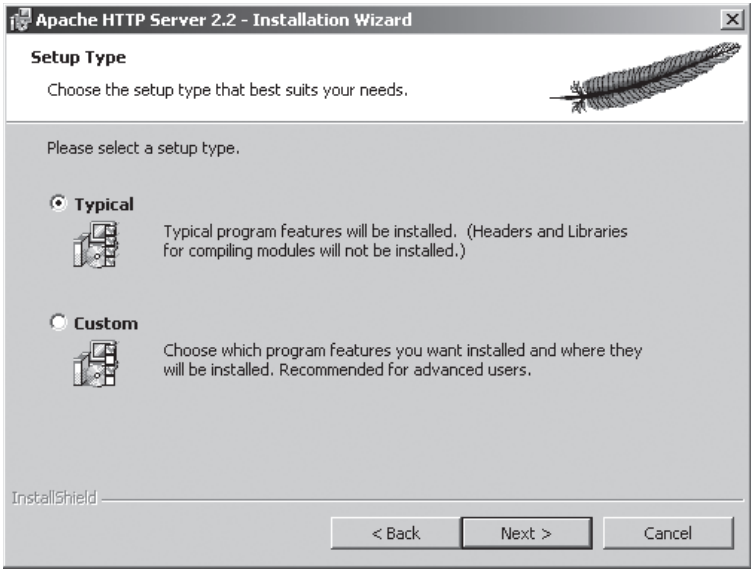


Figura A-17 Selección del tipo de instalación de Apache

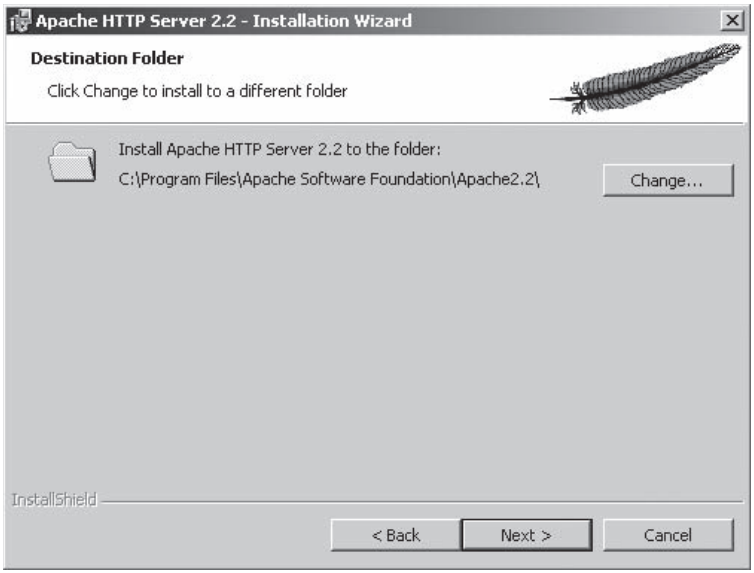


Figura A-18 Selección de la ubicación para instalar Apache

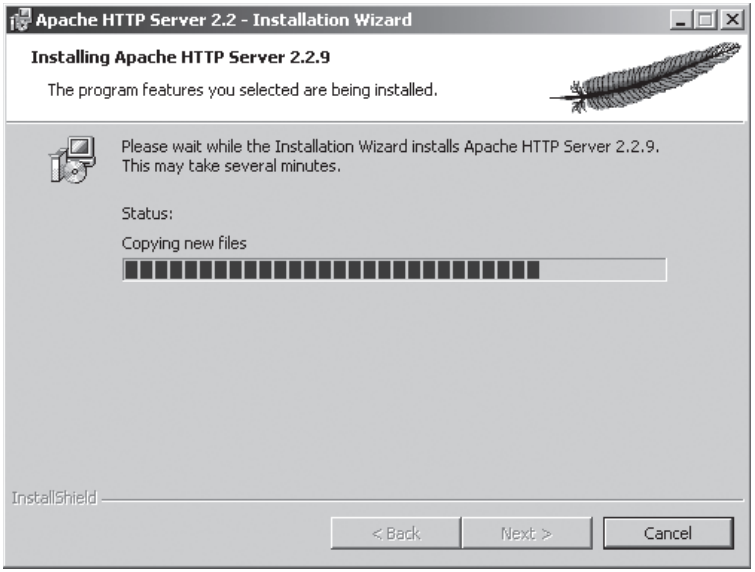


Figura A-19 Instalación de Apache en progreso

Instalar PHP

Existen dos versiones del archivo binario para instalar PHP en Windows: un archivo Zip que contiene todas las extensiones PHP y requiere instalación manual y la versión automática del instalador para Windows que sólo contiene el archivo binario PHP sin extensiones extra. Esta sección describe el proceso de instalación para el archivo Zip PHP.

1. Ingresa como administrador (si estás utilizando Windows NT/2000/XP/Vista) y desempaqueta el archivo binario en un directorio de tu sistema, por ejemplo: *c:\php*. Después de la extracción, este directorio debe parecerse al que se presenta en la figura A-20.
2. A continuación, renombra el archivo *php.ini-recommended* en el directorio de instalación PHP y nómbralo *php.ini*. Este archivo contiene los valores de configuración para PHP, que pueden ser utilizados para modificar la manera en que funciona. Lee los comentarios dentro del archivo para conocer más sobre las configuraciones disponibles.
3. Dentro del archivo *php.ini* localiza la línea

   ```
   extension_dir = "./"
   ```

   y modifícala para que se lea así:

   ```
   extension_dir = "c:\php\ext\"
   ```

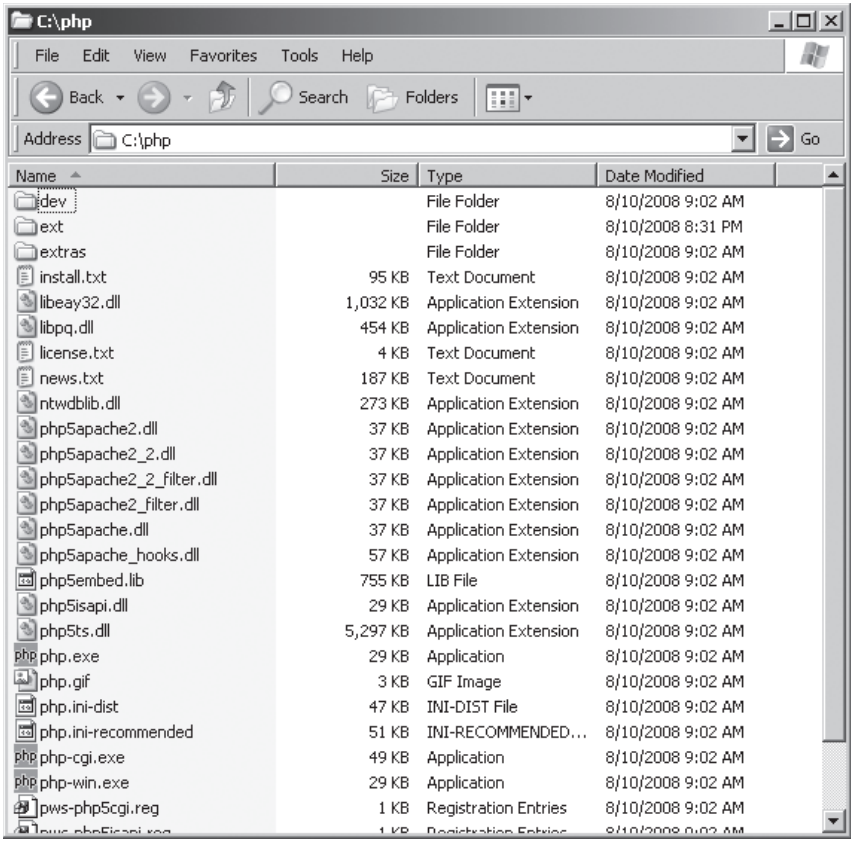


Figura A-20 La estructura de directorio creada al desempaquetar la versión binaria de instalación para Windows

Esto le indica a PHP dónde se localizan las extensiones incluidas en el paquete. Recuerda reemplazar la ruta de acceso "`c:\php\`" con la ubicación de tu instalación PHP.

4. A continuación busca las siguientes líneas y elimina el punto y coma al principio de cada una de ellas (si está presente) de manera que se lean así:

```
extension=php_pdo.dll
extension=php_sqlite.dll
extension=php_mysqli.dll
extension=php_pdo_sqlite.dll
extension=php_pdo_mysqli.dll
```

Esto se encarga de activar las extensiones de PHP para MySQLi, SQLite y PDO.

5. Abre el archivo de configuración de Apache, *httpd.conf* (que puede localizarse en el subdirectorio *conf/* del directorio de instalación de Apache) en un editor de textos y añádele las siguientes líneas:

```
AddType application/x-httpd-php .php
LoadModule php5_module "c:\php\php5apache2_2.dll"
SetEnv PHPRC C:\php\
```

   Estas líneas le indican a Apache cómo manejar los scripts PHP y dónde encontrar el archivo de configuración *php.ini*. Recuerda reemplazar la ruta de acceso *c:\php* con la ubicación de tu instalación PHP.

6. Cuando el servidor Apache está instalado, se añade de manera automática al menú Inicio. Utiliza este grupo del menú para detener y reiniciar el servidor, como se muestra en la figura A-21.

Ahora PHP está instalado y configurado para funcionar con Apache. Para comprobarlo, ve a la sección "Probar PHP".

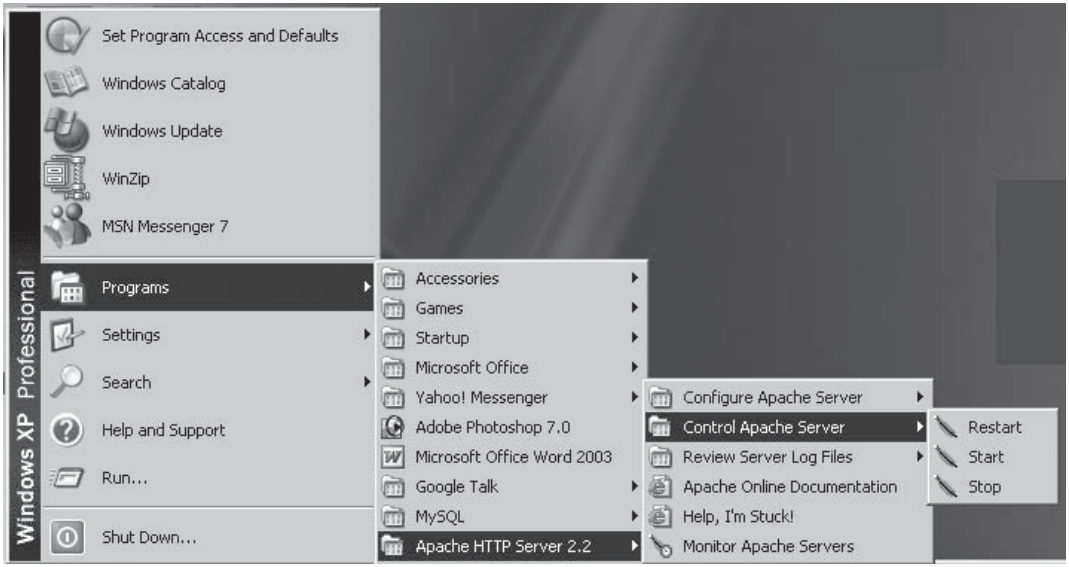


Figura A-21 Controles del servidor Apache en Windows

Instalar SQLite

Dado que SQLite es un archivo ejecutable independiente, la instalación en Windows se realiza en un abrir y cerrar de ojos: simplemente ingresa como administrador (si estás utilizando Windows NT/2000/XP/Vista) y desempaqueta el archivo binario en un directorio temporal de tu sistema. Para mayor facilidad renombra el archivo binario *sqlite2.exe* por *sqlite.exe*.

La versión binaria de SQLite ya está instalada y lista para funcionar. Para probarla, ve a la sección "Probar SQLite".

Probar el software

Una vez que el software ha sido instalado con éxito y que los diferentes servidores están en funcionamiento, puedes verificar que todo funcione como es debido mediante unas cuantas pruebas sencillas.

Probar MySQL

Primero, arranca el programa cliente de línea de comandos de MySQL, cambiando al subdirectorio *bin/* de tu directorio de instalación MySQL y escribiendo el siguiente comando:

```
prompt# mysql -u root
```

Se te debe recompensar con un mensaje como el que aparece a continuación:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 26288
Server version: 5.0.51a-community MySQL Community Edition (GPL)
Type 'help' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

En este punto, ya estás conectado al servidor MySQL y puedes comenzar a ejecutar comandos SQL o consultas para verificar que el servidor está funcionando apropiadamente. A continuación presentamos algunos ejemplos con sus respectivos datos de salida:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.13 sec)
mysql> USE mysql;
```

```
Database changed
mysql> SHOW TABLES;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| func                      |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| proc                      |
| procs_priv                |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
17 rows in set (0.01 sec)
mysql> SELECT COUNT(*) FROM user;
+----------+
| count(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)
```

Si ves datos de entrada similares a éstos, tu instalación de MySQL está funcionando correctamente. Sal del cliente de líneas de comando escribiendo el siguiente comando, y regresarás a la línea del principio:

```
mysql> exit
```

Si no ves datos de salida semejantes a los que se presentan arriba, o si tu SQL lanza mensajes de advertencia o errores, revisa el procedimiento de instalación en la sección anterior, al igual que los documentos que acompañan a la versión de MySQL que instalaste para saber qué anda mal.

Probar PHP

Una vez que instalaste con éxito PHP como módulo de Apache, debes probarlo para asegurarte de que el servidor Web reconoce los scripts PHP y los maneja correctamente.

Para realizar esta prueba crea un script PHP en cualquier editor de texto que contenga las siguientes líneas:

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

Guarda este archivo como *prueba.php* en la raíz de los documentos de tu servidor Web (el subdirectorio *htdocs/* de tu directorio de instalación de Apache), y dirige tu explorador Web a http://localhost/prueba.php. Debes ver una página con la información de la construcción de PHP, como se muestra en la figura A-22.

Pon atención a la lista de extensiones para asegurarte de que estén activas las correspondientes a SimpleXML, MySQLi y PDO. Si no están activas, revisa el procedimiento de instalación, así como la documentación que acompaña al programa, para saber qué salió mal.

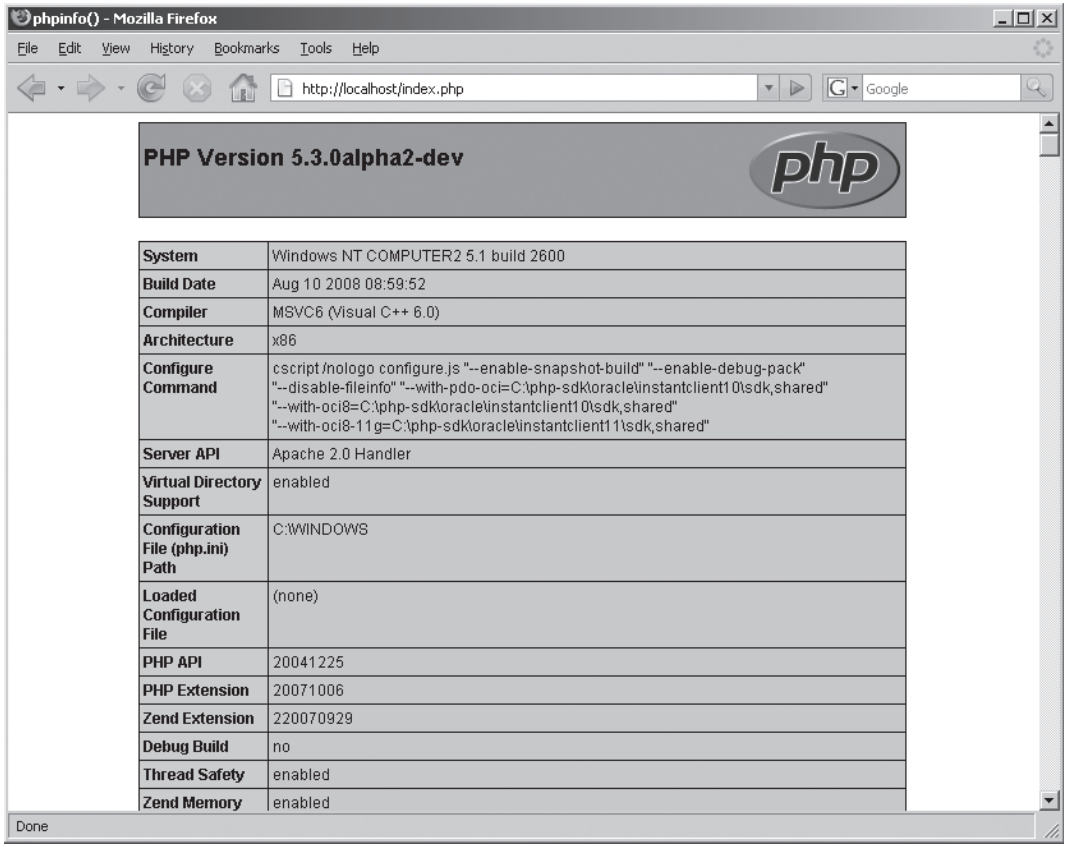


Figura A-22 Datos de salida del comando `phpinfo()`

Probar SQLite

Una vez que has instalado SQLite, puedes probarlo al cambiar al directorio donde fue instalado y ejecutando el binario SQLite en la línea de comando, como se muestra aquí:

```
prompt# sqlite
```

Debes recibir una recompensa con el mensaje que se muestra a continuación:

```
SQLite versión 2.8.17
Enter ".help" for instructions
sqlite>
```

En este punto, puedes ejecutar comandos SQL o comandos internos SQLite para probar si las cosas están funcionando como es debido. He aquí algunos ejemplos:

```
sqlite> .show
     echo: off
  explain: off
  headers: off
     mode: list
nullvalue: ""
   output: stdout
separator: "|"
    width:
sqlite> CREATE TABLE prueba (
   ...> fld1 INTEGER PRIMARY KEY,
   ...> fld2 TEXT
   ...> );
sqlite> INSERT INTO prueba (fld2) VALUES ('Hola');
sqlite> SELECT * FROM prueba;
1|Hola
```

Si ves datos de salida parecidos a los anteriores, tu instalación de SQLite está funcionando correctamente. Sal del cliente de línea de comandos escribiendo el siguiente comando, y regresarás a la primera instancia.

```
sqlite> .quit
```

Realizar pasos posteriores a la instalación

Una vez que las pruebas fueron completadas, es posible que quieras realizar las siguientes dos tareas.

Establecer la contraseña de superusuario en MySQL

En UNIX, la primera vez que se instala MySQL, el acceso al servidor de base de datos está restringido al administrador de la misma, también conocido como "root". Por defecto, este usuario se inicializa con una contraseña en blanco, lo que se considera en general como una mala práctica. Por lo tanto, debes remediarlo lo antes posible estableciendo una contraseña para este usuario a través de la utilidad *mysqladmin* incluida en el paquete, con la siguiente sintaxis en UNIX:

```
[root@host]# /usr/local/mysql/bin/mysqladmin -u root password 'nueva-
contraseña'
```

En Windows, puedes utilizar el MySQL Server Instance Config Wizard, que te permite establecer o restablecer la contraseña para el administrador de la base de datos (consulta la sección "Instalar en Windows" para conocer más detalles).

El cambio de la contraseña se aplica de inmediato, sin necesidad de reiniciar el servidor.

Pregunta al experto

P: ¿Cambiar la contraseña de "root" de MySQL en UNIX afecta a la misma cuenta en todo el sistema operativo?

R: No. El usuario "root" de MySQL no es el mismo que el superusuario del sistema operativo UNIX. Así que modificar uno no afecta al otro.

Configurar MySQL y Apache para comenzar automáticamente

En UNIX, los servidores MySQL y Apache tienen scripts para arrancar y terminar su ejecución. Estos scripts se localizan dentro de la jerarquía de instalación de cada programa. He aquí un ejemplo para utilizar el script de control del servidor MySQL:

```
[root@host]# /usr/local/mysql/support-files/mysql.server start
[root@host]# /usr/local/mysql/support-files/mysql.server stop
```

Y aquí un ejemplo de cómo utilizar el script de control de Apache:

```
[root@host]# /usr/local/apache/bin/apachectl start
[root@host]# /usr/local/apache/bin/apachectl stop
```

- Para que MySQL y Apache arranquen automáticamente en el momento en que inicia UNIX, simplemente invoca su respectivo script de control con los parámetros apropiados desde tus scripts de arranque y detención en la jerarquía `/etc/rc.d/*`.
- Para arrancar MySQL y Apache automáticamente en Windows, añade un vínculo de los archivos binarios *mysqld.exe* y *apache.exe* al grupo Inicio. También puedes iniciar automáticamente MySQL instalándolo como servicio de Windows (ver la sección titulada "Instalar en Windows" para instrucciones).

Resumen

Como aplicaciones de código libre, MyQSL, SQLite, Apache y PHP están disponibles para una amplia gama de plataformas y arquitecturas, tanto en formato binario como en código fuente. Este capítulo muestra el proceso de instalación y configuración de los componentes de software necesarios para crear un ambiente de desarrollo en las dos plataformas más comunes, UNIX y Windows. También te enseñó a configurar tu computadora para lanzar estos componentes automáticamente cada vez que se arranca el sistema, y te ofreció algunos consejos para la seguridad básica de MySQL.

Para saber más sobre el proceso de instalación descrito en este capítulo, o para conocer detalles sobre manejo y asesoría en la detección y solución de problemas, puedes visitar las siguientes páginas:

- Notas de instalación de MySQL, en **http://dev.mysql.com/doc/refman/5.0/en/installing-binary.html**
- Lineamientos generales para compilar Apache en UNIX, en **http://httpd.apache.org/docs/2.2/install.html**
- Notas específicas para Windows para la instalación del archivo binario de Apache, en **http://httpd.apache.org/docs/2.2/platform/windows.html**
- Instrucciones de instalación de PHP para Windows, en **www.php.net/manual/en/install.windows.php**
- Instrucciones de instalación de PHP para UNIX, en **www.php.net/manual/en/install.unix.php**
- Instrucciones de instalación de PHP para Mac OS X, en **www.php.net/manual/en/install.macosx.php**
- Preguntas más frecuentes de SQLite, en **www.sqlite.org/faq.html**

Apéndice B

Respuestas a los autoexámenes

Capítulo 1: Introducción a PHP

1. Los cuatro componentes del conjunto LAMP son el sistema operativo Linux, el servidor Web Apache, el servidor de base de datos MySQL y el lenguaje de programación PHP.
2. PHP es superior a los lenguajes del lado del cliente como JavaScript porque todo el código de PHP se ejecuta del lado del servidor, y por lo tanto no tiene los riesgos de ejecutarse del lado del cliente ni está supeditado a la configuración del explorador cliente.
3. La declaración `echo` produce una o más líneas como datos de salida.
4. El analizador sintáctico PHP ignora espacios en blanco o líneas vacías en el script PHP.
5. Un punto y coma. Sin embargo, este punto y coma terminador puede ser omitido en la última línea del script PHP.
6. Una secuencia de escape es un conjunto de caracteres que es reemplazado por un solo carácter "especial". Por ejemplo, `\t` reemplaza el carácter tabulador. Otras secuencias de escape muy utilizadas son `\n` (salto de línea) y `\r` (retorno de carro).
7. A. `Hoy amaneció`
   `soleado y brillante`

   B. `Lo nuestro no es preguntarnos por qué;Lo nuestro es hacer o morir`
8. A. La sintaxis del bloque comentado es incorrecta.

   B. No hay error.

   C. A la declaración `echo` le falta la comilla de cierre.

Capítulo 2: Utilizar variables y operadores

1. La función `gettype()`
2. `$24` y `$^b` son nombres de variables no válidos. Los nombres de variables no pueden iniciar con un carácter numérico; tampoco pueden tener signos de puntuación ni símbolos especiales.
3.
```
<?php
define ('SABOR', 'fresa');
?>
```
4.
```
<?php
$x = 6;
$x += 3;
?>
```

5. Responder verdadero o falso:

A. Verdadero

B. Falso

C. Falso

D. Falso

E. Falso

F. Verdadero

G. Verdadero

H. Falso

6. `$x = 179; ABC = 90`

7. Como ambas variables son de tipo entero y ambas contienen el mismo valor, la comparación regresa un valor 1 (verdadero).

8. La constante `NUM` es una cadena de texto (advierte las comillas sencillas), la cual es asignada después a la variable `$a`. Por lo tanto, la variable `$a` es también una cadena de texto (string).

9. El código de formulario relevante es:

```
<form method="post" action="convertir.php">
      Tasa de cambio: <br />
      <input type="text" name="tasa" /> <p />
      Cantidad en dólares: <br />
      <input type="text" name="dólares" /> <p />
      <input type="submit" />
</form>
```

Y el código PHP para el procesamiento es

```
<?php
$dólares = $_POST['dólares'];
$tasa =  $_POST['tasa'];
$euros = $dólares * $tasa;
echo "$dólares USD es equivalente a: $euros EUR";
?>
```

10. El código relevante del formulario es:

```
<form method="post" action="convertir.php">
      Temperatura en Celsius: <br />
      <input type="text" name="c" /> <p />
      <input type="submit" />
</form>
```

Y el código PHP es

```
<?php
$c = $_POST['c'];
```

```
$f = (9/5) * $c + 32;
echo "$c grados Celsius es equivalente a: $f grados Fahrenheit";
?>
```

11. El código relevante es:

```
<form method="post" action="muestra.php">
      Texto de entrada: <br />
      <input type="text" name="desc" /> <p />
      Contraseña: <br />
      <input type="password" name="clave" /> <p />
      Área de texto: <br />
      <textarea cols="10" rows="5" name="comentario"></textarea> <p />
      Lista de selección: <br />
      <select name="color">
        <option>rojo</option>
        <option>amarillo</option>
        <option>verde</option>
      </select><p />
      Botones de opción: <br />
      <input type="radio" name="sexo" value="masculino">Masculino</input>
      <input type="radio" name="sexo" value="femenino">Femenino</input>
<p />
      Casillas de verificación: <br />
      <input type="checkbox" name="tel">Teléfono</input>
      <input type="checkbox" name="fax">Fax</input> <p />
      <input type="hidden" name="referencia" value="XYZ" />
      <input type="submit" />
</form>
```

Y el código de procesamiento PHP es

```
<?php
var_dump($_POST);
?>
```

Capítulo 3: Controlar el flujo del programa

1. Una declaración `if-else` permite definir acciones para dos posibilidades: una condición verdadera y una condición falsa. Una declaración `if-elseif-else` permite definir acciones para varias posibilidades.

2. La declaración condicional relevante es

```
<?php
if ($ciudad == 'Minneapolis') {
  echo 'Bienvenidos a Minneapolis';
}
?>
```

3. Un bucle `while` verifica la expresión condicional al principio de cada iteración, mientras que el bucle `do-while` verifica la expresión condicional al final de cada iteración. De esta manera, si la expresión condicional se evalúa como falsa en la primera iteración, el bucle `while` nunca se ejecutará, pero el `do-while` se ejecutará una vez.

El siguiente código ilustra la diferencia:

Código	Datos de salida

`<?php`
`$a = 10;`
`while ($a < 10) {`
`echo 'Hola';`
`}`
`?>`
	Ningún dato de salida

`<?php`
`$a = 10;`
`do {`
`echo 'Hola';`
`} while ($a < 10);`
`?>`
	`'Hola'`

4. El código PHP relevante es:

```
<?php
$num = 1;
while ($num <= 10) {
echo "8 por $num es: " . ($num*8) . "<br/>";
$num++;
}
?>
```

5. El código PHP relevante es:

```
<?php
for ($num=1; $num<=10; $num++) {
echo "8 por $num es: " . ($num*8) . "<br/>";
}
?>
```

6. La función que coincide con cada tarea se muestra en la siguiente tabla:

Tarea	Función
Decodifica entidades HTML	`html_entity_decode()`
Pone en mayúsculas una cadena de texto	`strtoupper()`
Redondea hacia abajo un número	`floor()`
Elimina los espacios en blanco en una cadena de texto	`trim()`
Genera un número aleatorio	`rand()`

Tarea	Función
Invierte una cadena de texto	`strrev()`
Cuenta palabras en una cadena de texto	`str_word_count()`
Cuenta caracteres en una cadena de texto	`strlen()`
Termina el proceso del script con un mensaje personalizado	`die()`
Compara dos cadenas de texto	`strcmp()`
Calcula el exponente de un número	`exp()`
Convierte un número decimal a su valor hexadecimal	`dechex()`

7. Los datos de salida son `7402.404200`.

8. El código PHP requerido es

```
<?php
$str = 'Marco tuvo un día agradable';
$newStr = substr($str, 0,9) . substr($str,12,3) .
$str{13} . $str{2} . strtolower($str{0}) .
$str{7} . substr($str,-1,1) . substr($str,-3,3);
echo $newStr;
?>
```

Capítulo 4: Trabajar con matrices

1. Los dos tipos de matrices de PHP son las indexadas numéricamente y las indexadas por cadenas de texto (también conocidas como matrices asociativas). Con las primeras, los números se utilizan para identificar elementos de la matriz; con los segundos, se utilizan etiquetas de texto únicas (claves).

2. La función que coincide con cada tarea se muestra en la siguiente lista:

Tarea	Función
Elimina elementos duplicados de una matriz	`array_unique()`
Añade un elemento al inicio de una matriz	`array_unshift()`
Dispone una matriz en orden inverso	`rsort()`
Cuenta el número de elementos en una matriz	`count()`
Busca un valor en una matriz	`in_array()`
Muestra el contenido de un arreglo	`print_r()` o `var_dump()`
Revuelve el contenido de una matriz	`shuffle()`
Combina dos matrices en una	`array_merge()`
Encuentra los elementos comunes entre dos matrices	`array_intersect()`
Convierte una cadena de caracteres en una matriz	`explode()`
Extrae un segmento de la matriz	`array_slice()`

3. Los datos de salida serían:

```
Array
(
    [0] => f
    [1] => g
    [2] => i
)
```

4. El código PHP relevante es

```
<?php
$días = array(
  'Lunes',
  'Martes',
  'Miércoles',
  'Jueves',
  'Viernes',
  'Sábado',
  'Domingo'
);
foreach ($días as $día) {
  echo "$día \r\n";
}
?>
```

5. El código relevante es

```
<?php
$arr = array(23,45,2,67,17,12,5,68,14,78,192,4);
foreach ($arr as $n) {
  echo ($n < 15) ? "$n " : null;
}
?>
```

6. El código relevante es

```
<?php
$arr = array(23,45,'manzana','fig',17,12,5,'fig',14,2,78,192,45);
if ($arr == array_unique($arr)) {
  echo 'La matriz sólo tiene valores únicos';
} else {
  echo 'La matriz tiene algunos valores duplicados';
}
?>
```

Capítulo 5: Usar funciones y clases

1. Funciones

- Permiten crear módulos y reciclarlos.
- Reducen la duplicación del código de programación.

- Permiten cambiar el código desde un solo lugar central.
- Simplifican la depuración.

2. Un argumento es un dato de entrada de una función, mientras que un valor regresado es el dato de salida de una función.

3. El código PHP relevante es

```
<?php
function traeDistancia($velocidad, $tiempo) {
  return $velocidad * $tiempo;
}

// diferencia de tiempo = 4.5 hrs.
// tiempo total en el aire = 13-5 - 4.5 = 9
echo 'La distancia entre Bombay y Londres es: ' .
traeDistancia(910, 9) . ' km.';
?>
```

4. El código PHP relevante es

```
<?php
// definición de función
// obtiene MCM de un conjunto arbitrario de números usando MCD
function traeConjDeMCM() {
  $nums = func_get_args();
  if (count($nums)< 2) {
    die ('ERROR: Debe proporcionar por lo menos dos valores');
  }

  $mcm = 1;
  for ($x=0; $x<count($nums); $x++) {
    $mcm = traeMCM($mcm, $nums[$x]);
  }
  return $mcm;
}

// datos de salida: 160
echo traeConjDeMCM(4,8,16,32,10);
?>
```

5. Un constructor es un método de una clase que se ejecuta automáticamente cuando se crea una instancia de esa clase.

6. Los métodos privados de una clase sólo son visibles dentro de la clase y no son accesibles para las clases secundarias ni para las instancias de éstas. Si se trata de acceder a un método privado de una clase principal desde la clase secundaria se generará un error fatal.

7. La función `get_class()` regresa el nombre de la clase desde la cual fue inicializada, en este caso la clase Nene.

8. La definición relevante de clase es

```
<?php
class SeleccionarFecha {
  public $dia;
  public $mes;
  public $ano;

  public function _construct() {
    // crea lista de días
    $this->dia = new Select;
    $this->day ->setName('dd');
    $this->day ->setLabel('Día');
    foreach (range(1,31) as $d) {
      $this->day->setOption(new Option($d,$d));
    }
    // crea lista de meses
    $this->mes = new Select;
    $this->mes-> setName('mm');
    $this->mes-> setLabel('Mes');
    foreach (range(1,12) as $m) {
      $mon = date('F', mktime(0,0,0,$m));
      $this->mes-> = setOption(new Option($mon,$mon));
    }
    // crea lista de años
    $this->ano = new Select;
    $this->ano-> setName('yy');
    $this->day-> setLabel('Año');
    foreach (range(1950,2050) as $y) {
      $this->ano-> setOption(new Option($y,$y));
    }
  }
  public function render() {
    $this->dia->render();
    echo '<br />';
    $this->mes->render();
    echo '<br />';
    $this->ano->render();
  }
}
?>
```

Y un ejemplo utilizable es

```
<form method="post" action="fecha.php">
  <?php
  $datebox = new SeleccionarFecha;
  $datebox->render();
  ?>
  <p />
```

```
  <input type="submit" name="submit" value="Enviar" />
</form>
```

9. Monday 17 November 2008 12:15 pm

Capítulo 6: Trabajar con archivos y directorios

1. La función que coincide con cada tarea se muestra en la siguiente lista:

Tarea	Función
Obtiene la ruta de acceso absoluta de un archivo	`realpath()`
Borra un archivo	`unlink()`
Recupera el tamaño de un archivo	`filesize()`
Lee el contenido de un archivo en una matriz	`file()`
Verifica si el archivo tiene permisos de escritura	`is_writable`
Verifica si existe el archivo	`file_exists()`
Lee el contenido de un directorio en una matriz	`scandir()`
Escribe una cadena de texto en un archivo	`file_put_contents()`
Crea un nuevo directorio	`mkdir()`
Borra un directorio	`rmdir()`
Crea un apuntador a un directorio	`opendir()`
Verifica si la entrada a un directorio es un archivo	`is_file`

2. La función `flock()` se utiliza para evitar el procesamiento múltiple y simultáneo de un archivo, por lo que reduce la posibilidad de corrupción de datos.

3. El código PHP relevante es

```
<?php
echo count(file('ejemplo.txt')) . ' línea(s) en el archivo.';
?>
```

4. El código PHP relevante es

```
<?php
file_put_contents('invertido.txt', strrev(file_get_contents('ejemplo.
txt')))
?>
```

5. El código PHP relevante es

```
<?php
// crea un apuntador a directorio
$dp = opendir('.') or die ('ERROR: No es posible abrir el directorio');

// lee el contenido del directorio
// muestra los nombres de archivos encontrados
```

```
while ($archivo = readdir($dp)) {
  $datosArchivo = pathinfo($archivo);
  if ($datosArchivo ['extension'] == 'txt') {
    rename($archivo, $datosArchivo['filename'] . '.xtx') or die
('ERROR: No es posible cambiar el nombre del archivo ' . $archivo);
  }
}

// destruye el apuntador
closedir($dp);
?>
```

6. El código PHP relevante es

```
<?php
// definición de función
// copiar directorio y su contenido
function copiaArbol($origen, $destino) {
  if (file_exists($origen)) {
    // crea puntero hacia fuente
    $dp = opendir($origen) or die ('ERROR: No es posible abrir el
directorio');

    // si el directorio destino no existe
    // créalo
    if (!file_exists($destino)) {
      mkdir($destino) or die ('ERROR: No es posible crear el
directorio' . $destino);
    }

    // lee el contenido fuente
    // si es un archivo, copia al destino
    // si es un directorio, hace invocaciones recursivas
    while ($archivo = readdir($dp)) {
      if($archivo != '.' && $archivo != '..') {
        if(is_file("$origen/$archivo")) {
          copy("$origen/$archivo", "$destino/$archivo")
           or die('ERROR: No es posible copiar archivo ' . "$origen/
$archivo");
        } else if (is_dir("$origen/$archivo")) {
          copiaArbol("$origen/$archivo", "$destino/$archivo");
        }
      }
    }

    // cierra apuntador a origen
    closedir($dp);
  }
}
?>
```

Y he aquí un ejemplo utilizable:

```
<?php
if (file_exists('midir')) {
  copiaArbol('midir', '../../../micopia');
  echo 'Archivo(s) copiado(s).';
}
?>
```

7. El código PHP relevante es

```
<?php
// crea un puntero al directorio
$dp = opendir('.') or die ('ERROR: No es posible abrir el directorio');

// lee el contenido del directorio
// añade archivos al arreglo con mtime
while ($archivo = readdir($dp)) {
  if ($archivo != '.' && $archivo != '..') {
    $archivoList[$archivo] = filemtime($archivo);
  }
}

// destruye el puntero de directorio
closedir($dp);

// invierte el orden por mtime
// presenta la lista
arsort($archivoList);
foreach ($archivoList as $archivo => $mtime) {
  echo $archivo . '' . date ('d-M-y H:i', $mtime) . "\n";
}
?>
```

Capítulo 7: Trabajar con bases de datos y SQL

1. Las respuestas son

A. Falso. SQL permite que las tablas se fusionen en cualquiera de sus campos.

B. Falso. PHP ha incluido soporte para MySQL durante años.

C. Verdadero.

D. Falso. Las llaves primarias identifican de manera única a los registros de la tabla y no pueden estar vacías.

E. Verdadero.

F. Falso. Las declaraciones preparadas pueden ser utilizadas para cualquier operación SQL, incluyendo `SELECT`, `UPDATE` y `DELETE`.

2. Los comandos SQL relevantes son

A. DROP DATABASE

B. UPDATE

C. DELETE

D. CREATE TABLE

E. USE

3. La normalización de una base de datos es el proceso de identificar y eliminar redundancias y dependencias indeseables entre las tablas de una base de datos. Esto asegura que cierta pieza de información aparezca una sola vez en la base de datos, para evitar errores de referencias cruzadas y para realizar cambios fácilmente.

4. La ventaja de utilizar una biblioteca de abstracción, como PDO, en una base de datos, es que utiliza funciones genéricas que se traducen internamente a la API nativa de la base de datos; esto hace posible cambiar entre diferentes bases de datos con un impacto limitado en el código de aplicación. La desventaja es que la traducción interna de invocaciones API requiere ciclos de procesamiento adicionales, por lo que puede ser ineficiente.

5. El código PHP relevante es

```
<?php
// intenta establecer conexión con la base de datos
try {
   $pdo = new PDO('mysql:dbname=music;host=localhost', 'usuario',
'contra');
} catch (PDOException $e) {
   die ("ERROR: No se estableció la conexión: " . $e->getMessage());
}

// si el formulario no ha sido enviado
// muestra el formulario
if (!isset($_POST['submit'])) {

  // obtiene artistas
  $artistas = array();
  $sql = "SELECT artista_id, artista_nombre FROM artistas ORDER BY
artista_nombre";
  if ($result = $pdo->query($sql)) {
    while($row = $result->fetchObject()) {
      $artistas[$row->artista_id] = $row->artista_nombre;
    }
  }

  // obtiene los ratings
  $ratings = array();
  $sql = "SELECT rating_id, rating_nombre FROM ratings ORDER BY rating_
id";
  if ($result = $pdo->query($sql)) {
```

```
    while($row = $result->fetchObject()) {
      $ratings[$row->rating_id] = $row->rating_name;
    }
  }
?>
    <form method="post" action="pdo-music.php">
      Nombre: <br />
      <input type="text" name="nombre" />
      <p>
      Artista: <br />
      <select name="artista">
      <?php
      foreach ($artistas as $k => $v) {
        echo "<option value=\"$k\">$v</option>\n";
      }
      ?>
      </select>
      <p>
      Rating: <br />
      <select name="rating">
      <?php
      foreach ($ratings as $k => $v) {
        echo "<option value=\"$k\">$v</option>\n";
      }
      ?>
      </select>
      <p>
      <input type="submit" name="submit" value="Guardar" />
    </form>

<?php
// si el formulario ya ha sido enviado
// valida los datos de entrada y guarda el registro
} else {
  $nombre = $_POST['nombre'];
  $artista_id = (int)$_POST['artista'];
  $rating_id = (int)$_POST['rating'];

  // verifica datos de entrada
  if (empty($nombre)) {
    die('ERROR: Por favor escriba un nombre');
  }

  if (empty($artista_id)) {
    die('ERROR: Por favor seleccione un artista');
  }

  if (empty($rating_id)) {
    die('ERROR: Por favor seleccione un rating');
  }
```

```
  // limpia caracteres especiales de los datos de entrada
  $name = $pdo->quote($nombre);
  $artista_id = $pdo->quote($artista_id);
  $rating_id = $pdo->quote($rating_id);

  // inserta registro
  $sql = "INSERT INTO canciones (canción_título, ex_canción_artista,
ex_canción_rating) VALUES ($name, $artista_id, $ rating_id)";
  $ret = $pdo->exec($sql);
  if(ret === false) {
    echo "ERROR: No fue posible ejecutar $sql. " . print_r($pdo
->errorInfo());
  } else {
    echo 'Nueva canción añadida.';
  }
}

// cierra conexión
unset($pdo);
?>
  </body>
</html>
```

6. El código PHP relevante para MySQL es

```
<?php
// intenta establecer conexión con la base de datos
$mysqli = new mysqli("localhost", "usuario", "contra", "prueba");
if(mysqli === false) {
  die("ERROR: No fue posible establecer la conexión. " . mysqli_
connect_error());
}

// intenta ejecutar la consulta
// añade una tabla nueva
$sql =
  "CREATE TABLE grados (
    id int(4) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    subj_a INT(4) NOT NULL,
    subj_b INT(4) NOT NULL,
    subj_c INT(4) NOT NULL,
    )";
if ($mysqli->query($sql) === true) {
  echo 'Nueva tabla creada.';
} else {
  echo "ERROR: No fue posible ejecutar el query: $sql. " . mysqli
->error;
}

// cierra conexión
$mysqli->close();
?>
```

El código relevante para SQLite es

```
<?php
// intenta establecer la conexión con la base de datos
$sqlite = new SQLiteDatabase('test.db') or die ("No fue posible abrir
la base de datos");

// intenta ejecutar el query
// añade una nueva tabla
$sql = "CREATE TABLE grados (
          id INTEGER PRIMARY KEY,
          subj_a INTEGER NOT NULL,
          subj_b INTEGER NOT NULL,
          subj_c INTEGER NOT NULL,
)";
if($sqlite->queryExec($sql) == true) {
  echo 'Nueva tabla creada.';
} else {
  echo "ERROR: No fue posible ejecutar $sql. " . sqlite_error_
string($sqlite->lastError());
}

// cierra conexión
unset($sqlite);
?>
```

7. El código PHP relevante es

```
<?php
// intenta establecer la conexión
try {
   $mysql = new PDO('mysql:dbname=test;host=localhost', 'usuario',
'contra');
   $sqlite = new PDO('sqlite:a.db');
} catch (PDOException $e) {
   die("ERROR: No fue posible establecer la conexión: " . $e->
getMessage());
}

// prepara declaración INSERT para SQLite
$sql_1 = "INSERT INTO grados (id, subj_a, subj_b, subj_c) VALUES
(?,?,?,?)";
if ($stmt = $sqlite->prepare($sql_1)) {

  // crea y ejecuta el query SELECT para MySQL
  $sql_2 = "SELECT id, subj_a, subj_b, subj_c FROM grados";
  if($result = $mysql->query($sql_2)) {

    // para cada registro
    // prepara la declaración y la inserta en SQLite
    while($row = $result->fetch()) {
      $stmt->bindParam(1, $row[0]);
      $stmt->bindParam(2, $row[1]);
```

```
      $stmt->bindParam(3, $row[2]);
      $stmt->bindParam(4, $row[4]);
      $stmt->execute();
    }
    echo 'Registro(s) integrados correctamente.';

  } else {
    echo "ERROR: No fue posible ejecutar $sql_2. " . print_r($mysql
->errorInfo());
  }

} else {
  echo "ERROR: No fue posible ejecutar $sql_1. " . print_r($sqlite
->errorInfo());
}

// cierra conexión
unset($mysql);
unset($sqlite);
?>
```

Capítulo 8: Trabajar con XML

1. Los dos métodos para analizar sintácticamente un documento XML son la API Simple para XML (SAX) y el método de objetos de documento (DOM). SAX avanza por el documento, invocando diferentes funciones definidas por el usuario para procesar diversos tipos de nodos. DOM lee todo el documento en la memoria, genera una estructura de árbol para representarlo y proporciona métodos para recorrer el árbol.
2. Un documento XML bien formado tiene un solo elemento raíz, elementos correctamente anidados y sintaxis válida de elementos y atributos.
3. El código PHP relevante es

```
<?php
// carga archivo XML
$xml = simplexml_load_file('email.xml') or die ("Imposible cargar
documento XML!");

// hace un loop sobre elementos XML <persona>
// obtiene nodos hijo de <email> y los presenta
foreach ($xml->persona as $p) {
  echo $p->email . "\n";
}
?>
```

4. El código PHP relevante es

```
<?php
// inicializa un nuevo documento DOMDocument
$doc = new DOMDocument();
```

```
// quita sólo nodos de texto en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('arbol.xml');

// datos de salida: 'John'
echo $doc->firstChild->childNodes->item(2)->childNodes->item(2)
->childNodes->item(0)->childNodes->item(0)->childNodes->item(0)
->nodeValue;

// datos de salida: 'John'
echo $doc->getElementsByTagName('nombre')->item(0)->nodeValue;

// datos de salida: 'John'
echo $doc->getElementsByTagName('persona')->item(4)->childNodes
->item(0)->nodeValue;
?>
```

5. El código PHP relevante es

```
<?php
// inicializa un nuevo documento DOMDocument
$doc = new DOMDocument();

// quita sólo nodos de texto en blanco
$doc->preserveWhiteSpace = false;

// lee el archivo XML
$doc->load('biblioteca.xml');

// obtiene todos los elementos
// presenta el tamaño del conjunto
echo $doc->getElementsByTagName('*')->length . ' elemento(s)
encontrados.';
?>
```

6. El código PHP relevante es

```
<?php
// carga archivo XML
$xml = simplexml_load_file('biblioteca.xml') or die ("Imposible cargar
XML!");

// hace un bucle sobre los elementos XML <libro>
// incrementa cada rating 1
foreach ($xml->libro as $b) {
  $b['rating'] = $b['rating'] + 1;
}

// presenta el XML modificado
header('Content-Type: text/xml');
echo $xml->asXML();
?>
```

7. El código PHP relevante es

```
<?php
// conecta con la base de datos
try {
    $pdo=new PDO('mysql:dbname=test;host=localhost','usuario','contra');
} catch (PDOException $e) {
    echo "ERROR: No fue posible conectar. " . $e->getMessage();
}

// establece el modo de los errores
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

try {
    // ejecuta query SELECT
    $sql = "SELECT * FROM países";
    $stmt = $pdo->query($sql);
    if($stmt->rowCount() > 0) {
      // crea un nuevo árbol DOM
      $doc = new DOMDocument('1.0');

      // crea y adjunta elemento raíz <resultset>
      $raiz = $doc->createElement('resultset');
      $conjuntoresult = $doc->appendChild($raiz);

      while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        // hace un loop sobre el conjunto de resultados
        // establece un elemento <record> para cada fila
        $registro = $doc->createElement('record');
        $conjuntoresult->appendChild($registro);

        foreach ($row as $name => $value) {
          // adjunta campos y valores como <field>value</field>
          $campo = $doc->createElement($name);
          $texto = $doc->createTextNode($value);
          $registro->appendChild($campo);
          $campo->appendChild($texto);
        }
      }
    }
} catch (Exception $e) {
    echo "ERROR: No fue posible ejecutar el query \"$sql\". " .
$e->getMessage();
    unset($pdo);
}

// cierra conexión
// muestra la cadena XML
unset($pdo);
header('Content-Type: text/xml');
echo $doc->saveXML();
?>
```

Capítulo 9: Trabajar con cookies, sesiones y encabezados

1. Con una cookie, la información específica del usuario se almacena en el equipo cliente como un archivo de texto. Con una sesión, la información se almacena en el servidor como un archivo de texto o en una base de datos, y sólo un identificador único que haga referencia a dicha información se almacena en el equipo cliente como cookie.
2. Para eliminar una cookie almacenada, se establece una fecha de caducidad ya pasada.
3. Para registrar una variable de sesión, guárdala como un par clave-valor en la matriz asociativa especial `$_SESSION`. Está disponible de manera global, por lo que puede accederse a la variable de sesión en cualquier página haciendo referencia a `$_SESSION[`*`clave`*`]`, después de invocar la función `session_start()`.
4. Los encabezados HTTP deben enviarse antes que cualquier otro dato de salida generado por el script. Como el script del ejemplo genera una línea de salida antes de enviar el encabezado `'Location:'`, generará un error fatal.
5. El código PHP relevante es

   A. Dentro de la misma sesión:

```
<?php
session_start();
if (!isset($SESSION['count'])) {
  $_SESSION['count'] = 0;
}
echo 'Has visitado esta página ' . $_SESSION['count'] . 'veces.';
$_SESSION['count']++;
?>
```

   B. En diferentes sesiones:

```
<?php
if (!isset($_COOKIE['count'])) {
  $count = 0;
} else {
  $count = $_COOKIE['count'];
}
setcookie('count', ($count+1), mktime()+2592000, '/');
echo 'Has visitado esta página ' . $count . 'veces.';
?>
```

6. El código PHP relevante es

```
<?php
session_start();
session_destroy();
header('Location:login.php');
?>
```

Capítulo 10: Manejo de errores

1. El modelo basado en excepciones:
   - Permite tratamiento diferente de distintos tipos de errores.
   - Produce un código más limpio y sencillo.
   - Obliga a realizar mejor análisis de las condiciones de error.
   - Permite el manejo de errores inadvertidos.
2. Errores fatales, errores de arranque, errores en el tiempo de compilación y errores en el tiempo de ejecución de la segmentación.
3. La memoria reservada es un área temporal de almacenamiento para los datos de salida del script. Es útil porque proporciona un "área de captura" para que los datos de salida se procesen antes de que el usuario los vea. Resulta útil cuando se trabaja con funciones PHP como `setcookie()`, `header()` y `session_start()`, porque permite que estas funciones sean invocadas incluso después de que el script ha generado algunos datos de salida.

   El siguiente ejemplo lo ilustra:

```
<?php
// abre memoria reservada de salida
ob_start();

// envía algunos datos de salida
// que son almacenados en la memoria reservada
echo 'Algunos datos de salida';

// envía algunos encabezados
header('Content-Type: text/plain');
session_start();

// ahora muestra los datos de salida
ob_end_flush();
?>
```

4. Para el primero, el controlador de errores por defecto de PHP. Para el segundo, `handlerA()`.
5. El código PHP relevante es

```
<?php
class Warning extends Exception {}
class Notice extends Exception {}
error_reporting(E_ALL);
set_error_handler('miControlador');
```

```
// controlador de errores personalizado
function miControlador($type, $msg, $archivo, $line, $context) {
   switch($type) {
    case E_NOTICE:
      throw new Notice($msg);
      break;

    case E_WARNING:
      throw new Warning($msg);
      break;
  }
}

try {
  echo 45/0;
} catch (Notice $e) {
  echo 'NOTICE: ' . $e->getMessage();
} catch (Warning $e) {
  echo 'WARNING: ' . $e->getMessage();
} catch (Exception $e) {
  echo 'ERROR: ' . $e->getMessage();
}
?>
```

6. El código PHP relevante es

```
<?php
// define excepciones
class OtraExcepcion extends Exception {}
class AunOtraExcepcion extends Exception {}

// establece controlador personalizado
set_exception_handler('miControlador');

// personaliza manejo de errores
function miControlador($e) {
  $body = "Excepción sin atrapar: " . $e->getMessage() . " en la línea
" . $e->getLine() . " de " . $e->getFile();
  @mail('admin@dominio', 'Excepción sin atrapar', $body, 'De: bounce@
dominio');
}

try {
  throw new OtraExcepcion('Atrápame si puedes.');
} catch (AunOtraExcepcion $e) {
  echo $e->getMessage();
}
?>
```

Capítulo 11: Seguridad con PHP

1. Un *ataque de inyección de SQL* significa que los atacantes manipulan tu base de datos utilizando datos de entrada que contienen declaraciones SQL. Una higiene y validación apropiada de los datos de entrada previenen este tipo de ataques.

   Un *ataque de script de sitios cruzados* significa que los atacantes incrustan código Java Script o HTML en tus páginas Web. La estandarización de los datos de salida puede prevenir este tipo de ataques.

   Un *secuestro de sesión* significa que los atacantes interceptan y utilizan la sesión del cliente para ganar acceso y privilegios que no tienen. Añadir claves específicas de cliente a la seguridad de tus sistemas de seguridad en sesiones puede prevenir este tipo de ataques.

2. El código PHP relevante es

```
<?php
$cadena = '<div>
  <form action="http://www.mujermala.com/scriptmalo.php">
    <input type="hidden" name="op" value="add" />
    <input type="hidden" name="user" value="me" />
    <input type="hidden" name="pass" value="guessme" />
  </form>
';

// utiliza strip_tags()
$limpia['str'] = strip_tags($cadena);

// utiliza htmlentities()
$limpia['str'] = htmlentities($cadena);
?>
```

3. El código PHP relevante es

```
<?php
// función para validar un código postal de EU
// verifica si los datos de entrada tienen la forma ddddd-dddd
function validaCodigoPostal($str) {
    return preg_match("/^\d{5}(-\d{4})?$/" , $str);
}
?>
```

4. Los mensajes de error pueden contener información valiosa sobre el esquema de tu aplicación de base de datos, conversión de nombres de archivo, estructuras de sistemas de archivo y mecanismos persistentes de datos. Esta información puede ser utilizada por los atacantes para encontrar áreas vulnerables en tu aplicación y realizar ataques más específicos. Por esta razón, debe deshabilitarse la función de mostrar los mensajes de error en ambientes de producción (aunque estos mensajes deben almacenarse en un archivo o una base de datos aparte).

5. La siguiente tabla muestra lo que hace cada función:

Función	Lo que hace
`ctype_alnum()`	Verifica si los datos de entrada contienen sólo caracteres alfabéticos o numéricos
`addslashes()`	Limpia caracteres de entrada con diagonales
`filter_var()`	Prueba si los datos de entrada coinciden con ciertos patrones
`htmlentities()`	Convierte automáticamente los caracteres especiales provenientes de los datos de entrada en su correspondiente código HTML
`sqlite_escape_string()`	Limpia los datos de entrada con comillas dobles antes de insertarlos en una base de datos SQLite
`preg_match()`	Verifica si los datos de entrada coinciden con expresiones regulares
`strval()`	Regresa el valor de una cadena de texto de los datos de entrada

Capítulo 12: Extender PHP

1. Los paquetes de PEAR constan de clases PHP que pueden cargarse y utilizarse directamente en un script PHP. Los paquetes PECL constan de módulos de lenguaje que deben compilarse en el motor PHP.

2. Los pasos para instalar extensiones PECL en el ambiente de desarrollo de Windows son los siguientes:
   A. Visita el sitio Web PECL4WIN y descarga el archivo .DLL precompilado para las extensiones.
   B. Copia este archivo precompilado al directorio de extensiones PHP.
   C. Activa las extensiones escribiendo una línea como `extension=ext.dll` en el archivo de configuración *php.ini*.
   D. Reinicia el servidor Web para cargar las nuevas extensiones.

3. El código PHP relevante es

```
<?php
// crea un objeto
require_once 'Net/POP3.php';
$pop3 =& new Net_POP3();

// conecta con el servidor anfitrión
if(PEAR::isError($ret = $pop3->connect($host, $port))) {
    die($ret->getMessage());
}

// ingresa
if(PEAR::isError($ret = $pop3->login($user, $pass, 'USER'))) {
```

```
    die($ret->getMessage());
}
// obtiene la cantidad de mensajes y el tamaño del buzón
echo $pop3->numMsg() . 'mensajes en el buzón, ' . $pop3->getSize() . '
bytes <p/>';

// obtiene el contenido del mensaje más reciente
if ($pop3->numMsg() > 0) {
  echo '<pre>' . $pop3->getMsg($pop3->numMsg()) . '</pre>';
}

// desconecta
$pop3->disconnect();
?>
```

4. El código PHP relevante es

```
<?php
// inicializa el iterador
// pasa el directorio que será procesado
$iterator = new RecursiveIteratorIterator(new RecursiveDirectoryIterator
("music/"));

// itera sobre el directorio
// obtiene la etiqueta ID3 por cada archivo
foreach ($iterator as $key=>$value) {
  $tag = id3_get_tag(realpath($key));
  echo trim($tag['artista']) . ' - ' . trim($tag['título']);
  echo "<br/>";
}
?>
```

Índice

$ (dólar), 87
$_GET, 41, 89
$_POST, 40-41, 89
&&, 59
//, 15
||, 59
=, 23
===, 33
=>, 88
?, 52

A

Alcance de variables, 128-129
API reflexión, 138
Aplicaciones ejemplo, 16-18
Aplicaciones, ejemplos, 16-18
App.php, 325-327
Archivos,
    calcular el tamaño de un archivo, 173
    copiar, 176
    eliminar, 177-179
    encontrar ruta de acceso absoluta, 173-174
    funciones de archivo y directorio, 172
    recuperar atributos del archivo, 174-175
    renombrar, 176-177
    sensibilidad a mayúsculas, 174
    verificar si existe un archivo, 172-173
Archivos binarios, leer y escribir, 164
Archivos de configuración,
    leer y escribir, 165-168
    leer y escribir XML, 284-289
    seguridad, 353-354
Archivos externos, leer y evaluar, 179
Archivos locales, leer, 160-161
Archivos remotos, leer, 161-162
Archivos Zip, 386-387
Argumentos, 123, 124-125
    establecer valores de argumentos por defecto, 126-127
    listas dinámicas de argumentos, 127-128
ArrayIterator, 94-95
Arreglos, 86-87
    anidar, 91-92
    asignar el siguiente índice disponible automáticamente, 88
    asignar valores, 87-89
    asociativo, 87
    eliminar elementos, 90

límites en los elementos incluidos, 91
modificar valores del arreglo, 89-90
porcesar con loops y reiteradores, 92-96
recuperar el tamaño del arreglo, 90-91
utilizar con formularios, 96-99
Arreglos anidados, 91-92
Arreglos asociativos, 87
Arte.php, 335-338
Asignar exploradores a su tienda, 57-58
Ataque de inyección SQL, 351
Ataques script de sitios cruzados, 351
Autoexámenes,
Capítulo 1, 19-20
Capítulo 2, 45-47
Capítulo 3, 83
Capítulo 4, 119-120
Capítulo 5, 154-155
Capítulo 6, 183-184
Capítulo 7, 248
Capítulo 8, 290-291
Capítulo 9, 314
Capítulo 10, 348
Capítulo 11, 376
Capítulo 12, 388

B

Bases de datos, 186-187
añadir empleados, 209-216
añadir o modificar datos, 205-208
añadir tablas, 192-193
asegurar el acceso a la base de datos, 354-355
campos, 187
claves de acceso encriptadas, 246
colecciones de resultados, 188, 196-197
comodines, 197-198, 200
conmutar a diferentes bases de datos, 246-247
crear y alimentar bases de datos, 191-200
declaración CREATE DATABASE, 192
declaración CREATE TABLE, 192-193
declaraciones preparadas, 206-208, 239-240
fusionar tablas, 198-199
introducción, 187-188
llave primaria, 188
llaves externas, 188-189
manejo de errores, 209
normalización, 189
queries, 188, 194-196
registros, 187, 193-194, 199-200
relaciones, 188-189
tablas, 187, 198-199
tipos de campos, 200
ver también extensiones MySQLi; extensiones Objetos de datos PHP (PDO); SQL; extensión SQLite,
Bases de datos relacionales, 187
Biblioteca.xml, 256
Bloques catch, 330-332
Bloques try, 330-331
Bloques try-catch, 330

C

caballo.php, script, 10
Cadenas definidas por el usuario, 87
Calculador de edad, 116-118
Calculadora factorial, 64-66
Calculaedad.php, 116-117
Campos requeridos, 356-358
Caracteres subrayados, 87
Carro.php, 39-40
Categorías de error, 320
Clases, 135
API reflexión, 138
definir y utilizar, 135-138
encriptar y desencriptar texto, 139-143
extender, 144-147
métodos, 135
objetos, 135
propiedades, 135
Claves, 87
Claves de acceso, encriptar, 246
Código libre, 6
Coincidencia de patrones, 362-365
Color,
combinaciones de color RGB, 44
muestrario de color HTML, 42-44
Color.html, 42
Comando phpinfo(), 385
Comentarios, 12
Comillas dobles, 16
Comillas sencillas, 16

Config.xml, 287
Configure.php, 165-167, 284-287
Constantes, 29
    cuándo utilizarlas, 30
Constructores, 143-144
Convertidor dólares a euros, 37-38
Convertir entre bases numéricas, 75
Convertir.php, 37-38
Cookies, 294-295
    atributos, 295-296
    eliminar, 298
    encabezados, 296
    establecer, 297
    guardar y restaurar preferencias del usuario, 298-301
    leer, 297
    leer cookies almacenadas en tu computadora, 296

D

Datos de entrada de formulario, 39-41
    validar, 368-373
Datos de entrada, limpieza, 350-353
Declaración return, 123, 124-125
Declaración switch-case, 55-56
Declaraciones condicionales, 50
    combinar, 58-59
    declaraciones if, 50-51
    declaraciones if-else, 51-52
    declaraciones if-elseif-else, 55
    declaraciones switch-case, 55-56
Declaraciones if, 50-51
Declaraciones if-else, 51-52
Declaraciones if-elseif-else, 55
Declaraciones preparadas, 206-208, 239-240
Delimitadores, 7
Depurar errores, 342-347
Desarrollo,
    componentes, 9
    conceptos básicos, 7-9
Desempeño, 5-6
Destructores, 143-144
Diagonales invertidas, 15
Dirección.xml, 258
Directivas de configuración, 373-375
Directorios,
    crear, 175-176
    eliminar, 177-179
    funciones de archivo y directorio, 172
    procesar, 169-171
    renombrar, 176-177
    verificar si existe un directorio, 172-173
Dobles, 26

E

Empleados.php, 210-211
Encabezados, 306-308
Encriptar claves de acceso, 246
Encriptar y desencriptar texto, 139-143
Enteros, 26
    límites en, 31
Errores de acceso, 341-342
Escribir archivos, 163-164
    archivos binarios, 164
    configurar archivos, 165-168
Especificadores de formato, 76-77
Especificadores de precisión, 76
Especificadores de relleno numérico, 76
Etiquetas, 7
Excepciones, 330-333
    excepciones personalizadas, 334-335
    excepciones sin atrapar generan errores fatales, 335
    métodos del Objeto Excepción, 331
    validar datos de entrada de un formulario, 335-340
Expresiones regulares, 362-365
    extensibilidad, 144-147
Extensión DOM, 270
    alterar elementos y valores de atributo, 279-280
    atributos, 275-276
    conversión entre DOM y SimpleXML, 283-284
    crear nuevos documentos XML, 281-283
    elementos, 270-275
    nodos de texto vacíos, 272
    procesar recursivamente un árbol documento XML, 276-279

Extensión Objetos de Datos PHP (PDO), 186, 200-201, 234
añadir y modificar datos, 237-240
cadenas DSN comunes, 235
calcular la cantidad de registros en una colección de datos, 236
construir un formulario de ingreso, 241-245
declaraciones preparadas, 239-240
manejo de errores, 240
recuperar datos, 234-237
Extensión SQLite, 216-220
añadir y modificar datos, 224-225
conmutar a diferentes bases de datos, 246-247
crear lista personalizada de pendientes, 226-234
manejo de errores, 225
recuperar datos, 220-224
recuperar registros como arreglos y objetos, 221-224
tipos de datos, 218
Extensiones SimpleXML, 257
alterar elementos y valores de atributo, 262
añadir nuevos elementos y atributos, 263-264
atributos, 259-260
convertir entre DOM y SimpleXML, 283-284
convertir XML a SQL, 260-261
crear nuevos documentos XML, 264-266
elementos, 257-259
leer fuentes RSS, 266-269
ver también XML

F

Facilidad de uso, 6
factorial.php, 64-65
Fechas y horas,
calcular tiempo GMT, 115
convertir cadenas en sellos temporales, 114-115
convertir números de días y meses en nombres, 115
formatear, 112-113
funciones, 113
generar, 111-112
sello temporal UNIX, 111
verificar validez de fecha, 114
Flotantes, 26
Formulario de acceso, 241-245, 308-313
Formulario para registrar miembros, 77-81
Formularios normales, 189
Formularios, usar arreglos con, 96-99
Fuentes RSS, leer, 266-269
Función abs(), 74
Función addlashes(), 71
Función array_diff(), 107
Función array_intersect(), 107
Función array_key_exists(), 104-105
Función array_merge(), 106
Función array_pop(), 103
Función array_push(), 103
Función array_reverse(), 104
Función array_shift(), 103
Función array_slice(), 102
Función array_unique(), 103-104
Función array_unshift(), 103
Función asort(), 105-106
Función bindec(), 75
Función ceil(), 73-74
Función checkdate(), 114, 367-368
Función closedir(), 170
Función copy(), 176
Función count(), 90-91
Función ctype_alpha(), 361-362
Función ctype_digit(), 360
Función date(), 112-113, 115
Función decbin(), 75
Función dechex(), 75
Función decoct(), 75
Función define(), 29
Función die(), 81
Función empty(), 66-67, 80
Función exp(), 74
Función explode(), 101
Función fgets(), 161, 162
Función file(), 161
Función file_exists(), 164
Función file_get_contents(), 160-161, 163-164
Función file_put_contents(), 163-164
Función filter_var(), 366-367
Función flock(), 164
Función floor(), 73-74
Función fopen(), 162

Función fseek(), 162
Función fwrite(), 164
Función getdate(), 111-112
Función gettype(), 28
Función gmdate(), 115
Función hexdec(), 75
Función html_entity_decode(), 72
Función htmlentities(), 72
Función htmlspecialchars(), 72
Función htmlspecialchars_decode(), 72
Función implode(), 101
Función in_array(), 104
Función include(), 179
Función is_numeric(), 359
Función key(), 95
Función ksort(), 106
Función log(), 74
Función mail(), 80-81
Función max(), 101-102
Función min(), 101-102
Función mkdir(), 175-176
Función mktime(), 111
Función next(), 95
Función number_format(), 75-76
Función octdec(), 75
Función opendir(), 170
Función pathinfo(), 174
Función pow(), 74
Función preg_match(), 364
Función prin_values(), 131
Función print_r(), 92
Función printDir(), 171
Función printf(), 76-77
Función rand(), 74-75
Función range(), 101
Función readBlock(), 162-163
Función readdir(), 170-171
Función realpath(), 173
Función rename(), 176-177
Función require(), 179
Función rewind(), 95
Función rmdir(), 177-179
Función scandir(), 170
Función shuffle(), 104
Función sort(), 105
Función sprintf(), 76-77
Función str_repeat(), 68
Función str_replace(), 70
Función str_word_count(), 70
Función strcmp(), 70
Función strip_tags(), 72
Función striplashes(), 72
Función strlen(), 68, 361
Función strtolower(), 71
Función strtotime(), 114-115
Función strtoupper(), 71
Función strval(), 359-360
Función substr(), 68-69
Función trim(), 70-71
Función ucfirst(), 71
Función ucwords(), 71
Función unlink(), 177
Función unset(), 24-25, 28, 90
Función valid(), 95
Función var_dump(), 25-26, 28, 92
Funciones,
    funciones definidas por el usuario, 122-131
    para probar variables de tipos de datos, 29
Funciones de arreglo, 100
    añadir y eliminar elementos del arreglo, 103
    arreglos aleatorios e invertidos, 104
    búsquedas en arreglos, 104-105
    comparar arreglos, 107
    convertir entre cadenas y arreglos, 100-101
    eliminar elementos duplicados del arreglo, 103-104
    extraer segmentos del arreglo, 102
    fusionar arreglos, 106
    ordenar arreglos, 105-106
Funciones de cadena de texto, 66-67
    buscar cadenas vacías, 66-676
    cadenas HTML, 71-72
    comparar cadenas, 70
    contar cadenas, 70
    formatear, 70-71
    invertir y repetir cadenas, 68
    reemplazar cadenas, 70
    seguridad, 361-367
    subcadenas, 68-69
Funciones de cálculo, 73-74
Funciones definidas por el usuario, 122-123
    alcance de variables, 128-129
    argumentos, 123, 124-125
    calcular MCD y MCM, 132-134
    crear e invocar, 123-124

establecer valores de argumentos por defecto, 126-127
función body, 123
funciones recursivas, 129-131
lista dinámica de argumentos, 127-128
regresar valores, 123, 124-125
Funciones numéricas, 73
convertir entre bases numéricas, 75
formatear números, 75-77
funciones de cálculo, 73-74
generar números aleatorios, 74-75
Funciones recursivas, 129-131

G

Galería, 17
Galería fotográfica, crear, 180-182
Galería.php, 180-181
Grados.php, 95-96
Guarda.php, 226-228
Guía de seguridad PHP, 355, 356
Gutmans, Andi, 4

H

HTML
cadenas de texto, 71-72
combinar con PHP, 13-15
HTTP, 294
encabezados, 306-308

I

Inventario.xml, 260, 276-277

J

Jumbler.php, 139-142

L

LAMP,
conjunto de herramientas, 8
plataforma, 9
Leer archivos,
archivos binarios, 164
archivos de configuración, 165-168
archivos externos, 179
archivos locales, 160-161
archivos remotos, 161-162
segmentos específicos de un archivo, 162-163
Lenguaje de control de datos (DCL), 190
Lenguaje de definición de datos (DDL), 190
Lenguaje de manipulación de datos (DML), 190
Lenguaje de Marcado Extensible. *Ver* XML
Lenguajes interpretados, 7
Lerdorf, Rasmus, 4
Libros.php, 368-371
Limpiar datos de entrada y de salida, 350-353
Líneas en blanco, 11
Lista de argumentos tamaño de variable, 127-128
Lista de pendientes, 226-234
Lista.php, 229-231
Listas de selección, 148-153
Listas dinámicas de argumentos, 127-128
Login.php, 242-243, 309-310
Loops, 59-60
combinar, 62-63
interrumpir y evitar, 63-64
loops do-while, 60-61
loops for, 61-62
loops foreach, 93-94
loops while, 60
procesar arreglos con, 92-96
Loops do-while, 60-61
Loops for, 61-62
Loops foreach, 93-94
Loops while, 60

M

Main.php, 311-312
Manejo de errores, 12-13, 209, 225, 240, 318-320
controladores de error personalizados, 322-324, 330
depuración de errores, 342-347
errores de acceso, 341-342

generar una página de errores limpia, 325-329
restaurar controlador de errores por defecto, 325
Mantis, 18
Marca.php, 232-233
Máximo común múltiplo (MCM), calcular, 132-134
Mensajes de error, "variable indefinida", 25
Metacaracteres, 362-363
Método exec(), 238
Método fetch(), 236
Métodos, 135
Mexclar PHP con HTML, 13-15
Mínimo común múltiplo (MCM), calcular, 132-134
Muestra.php, 42-43
Muestrario de color HTML, 42-44
MySQL, tipos de datos, 193
MySQLi, extensión, 200-201
añadir empleados a una base de datos, 209-216
añadir o modificar datos, 205-208
declaraciones preparadas, 206-208
manejo de errores, 209
recuperar datos, 201-205
regresar registros como arreglos y objetos, 203-205

N

Niveles de error, 320
Nivel de error E_STRICT, 321-322
Notación científica, 27
Notación hexadecimal, 27
Notación octal, 27
Números aleatorios, 74-75

O

Objetos, 135
api reflexión, 138
encriptar y desencriptar texto, 139-143
Opcode cache, 6
Operador de asignación de suma, 34
Operador ternario, 52
Operadores, 30
aritméticos, 30-31
asignación, 23, 34-35
asignación de suma, 34
autoincremento y autodecremento, 35-36
comparación, 32-33, 51
concatenación, 31-32
limpiar datos de salida, 350-353
lógicos, 33-34
procedencia, 36-37
ternarios, 52
Operadores aritméticos, 30-31
Operadores de asignación, 23, 34-35
Operadores de comparación, 32-33, 51
Operadores de concatenación, 31-32, 35-36
Operadores lógicos, 33-34, 59

P

ParesNones.php, 53-54
PEAR, 6, 378-379
accesar buzones electrónicos POP3, 380-383
instalar, 379-380
PECL, 6, 384
crear archivos Zip, 386-387
instalar, 384-386
PHP,
características, 5-7
historia, 4-5
mezclar con HTML, 13-15
PhpBB, 17
PhpMiAdmin, 17
Pizza.html, 98
PoMMo, 17
POO,
configuración de visibilidad, 147-148
constructores y destructores, 143-144
extender clases, 144-147
generar listas de selección en formularios, 148-153
Pop3.php, 380-382
Portabilidad, 6
Preferencias-vuelo.php, 298-300
Primos.php, 107-109
Probar números pares y nones, 53-54

Procesador de Hipertexto PHP. *Ver* PHP
Procesamiento de directorios, 169-171
Promediar las calificaciones de un grupo, 95-96
Propiedades, 135
Publicación eZ, 18
Punto y coma, 11

R

Rangos numéricos, 101-102
Rastrear visitas previas a una página, 305-306
Registro.html, 77-78
Registro.php, 79-80
Reglas de procedencia, 36-37
Reiteradores,
    ArrayIterator, 93-94
    reiteradores, 94-95
Reportes de error, controlar, 321
Respuestas de autoexámenes,
    Capítulo 1, 390
    Capítulo 2, 390-392
    Capítulo 3, 392-394
    Capítulo 4, 394-395
    Capítulo 5, 395-397
    Capítulo 6, 398-400
    Capítulo 7, 400-405
    Capítulo 8, 405-407
    Capítulo 9, 408
    Capítulo 10, 409-410
    Capítulo 11, 411-412
    Capítulo 12, 412-413

S

Script,
    comentarios, 12
    comillas dobles, 16
    comprender, 11-12
    diagonales invertidas, 15
    escribir y ejecutar tu primer script, 10
    líneas en blanco, 12
    manejo de errores, 12-13
    punto y coma, 11
Secuencias de escape, 15, 16
Seguridad,
    acceso a bases de datos, 354-355
    archivos de configuración, 353-354
    cadenas de texto, 361-367
    configurar seguridad en PHP, 373-375
    fechas, 367-368
    limpiar datos de entrada y de salida, 350-353
    números, 358-360
    sesiones, 355-356
    validar datos de entrada de un formulario, 368-373
    validar datos de entrada del usuario, 356-358
Selección.php, 152-153
Selecciona.html, 39
Seleccionar mejores pizzas, 98-99
Sello temporal UNIX, 111
Sensibilidad a mayúsculas, 25,174
Sesiones, 302
    crear sesiones y variables de sesión, 302-303
    eliminar sesiones y variables de sesión, 304
    guardar variables de sesión, 303-304
    rastrear visitas previas a una página, 305-306
    seguridad, 355-356
Shiflett, Chris, 355
Signo de interrogación, 52
Signo de pesos ($), 87
Smarty, 18
Soporte comunitario, 6
Soporte para aplicaciones de terceros, 6-7
SQL, 187
    convertir XML a SQL, 260-261
    declaraciones SQL, 189-190
    sintaxis para declaraciones comunes, 190
    soporte para, 191
Squirrelmail, 18
Suraski, Zeev, 4

T

Tent.php, 57-58
Tipo de dato booleano, 26
Tipo de dato cadena de texto, 26
Tipo de datos NULL, 26, 27
Tipo juggling, 27-29

Tipos de datos, 26
establecer y verificar variables de tipos de datos, 27-29
funciones para probar variables de tipos de datos, 29
MySQL, 193
SQLite, 218
Transformar variables, 28

V

Validar datos de entrada del formulario, 335-340, 368-373
Validar datos de entrada del usuario, 356-358, 365-367
Validar fechas, 367-368
Validar URL, 365-367
Valores de punto flotante, 26
Variables,
asignar valores, 23
comparar, 32-33
convertir, 28
cuándo utilizarlas, 30
definir, 22
destruir, 24-25
inspeccionar contenido, 25-26
manipular con operadores, 30-37
mensaje de error "variable indefinida", 25
nombrar conversiones, 22-23
nombrar dinámicamente, 24
sensibilidad a mayúsculas, 25
tipos de datos, 26-29
Verificar números primos, 107-110
Visitas.php, 305

W

Wordpress, 18

X

XML,
anatomía de un documento XML, 251-253
atributos, 252, 255
conceptos básicos, 250-251
convertir XML a SQL, 260-261
crear documentos XML, 255-257
datos de carácter, 252
documentos bien formados, 253
documentos válidos, 253
elemento raíz, 252
elementos, 252, 253, 255
elementos de documento, 252
encriptación, 255
esquemas, 254
firma, 255
leer y escribir archivos de configuración, 284-289
mathML, 255
métodos de segmentación, 253-254
programas para usar con XML, 251
prólogo del documento, 252
query, 255
SOAP, 255
SVG, 255
xforms, 254
xhtml, 254
xlink, 254
xpointer, 254
xsl, 254
ver también extensiones DOM; extensiones SimpleXML
Xml2sql.php, 261

Z

zip.php, 386-387