

JACK FELLERS

php

**COME DIVENTARE PRODUTTIVI IN SOLI 7 GIORNI
CON IL MANUALE PRATICO E TEORICO AGGIORNATO
SULLA PROGRAMMAZIONE WEB LATO SERVER.
INCLUSI ESEMPI DI CODICE.**



WEBHAWK

PHP

JACK FELLERS



WEBHAWK

INDICE

Premessa

1. Php
2. L'installazione
3. Codice PHP
4. Tipi di dati
5. I cicli
6. Le funzioni
7. Oggetti
8. Eccezioni
9. Riferimento agli oggetti
10. Metodi e classi
11. Classi avanzate
12. Classi astratte e interfacce

Conclusioni

Caro lettore, per ringraziarti per la fiducia dimostratami acquistando il mio libro, ecco per te in **regalo**, una guida per fortificare ancora di più la tua conoscenza nella programmazione web!

Scansiona il codice o clicca sul link per riscattarlo in meno di un minuto:



Link alternativo al Qr code: <https://webhawk.tech/optin-it/>

Buona lettura!

PREMESSA

Nel corso degli anni si sono susseguiti linguaggi diversi, alcuni solo per un breve periodo, altri più duraturi. PHP è certamente un linguaggio longevo, la cui prima versione risale al 1994. Nel corso degli anni, molti altri linguaggi hanno cercato di sostituirlo, alcuni ci sono andati vicino, altri hanno fallito, ma PHP è un linguaggio che ha continuato a evolversi con i tempi. Ha facilitato lo sviluppo dei programmatori web, aggiungendo nuove ed entusiasmanti funzionalità ottimizzate in modo eccellente.

Il PHP non è un linguaggio difficile, ma nemmeno molto semplice, perché ogni sito web è diverso, quindi è impossibile prendere uno script, metterlo su un sito web e aspettarsi che funzioni. L'obiettivo è quello di aiutare i web designer con scarse o nulle conoscenze di programmazione ad acquisire la fiducia necessaria per addentrarsi nel codice e adattarlo alle proprie esigenze.

PHP

Ufficialmente, PHP sta per PHP Hypertext Preprocessor.

Questo libro ha lo scopo di aiutarti a programmare con PHP e, nel frattempo, farti capire cosa rende così felici i programmatori PHP e perché lo usano. PHP è un linguaggio di scripting che dà vita ai siti Web in diversi modi:

- Caricare i file attraverso una pagina Web;
- Generare miniature da immagini più grandi;
- Leggere e scrivere su file;
- Usare un database per visualizzare e archiviare informazioni;
- Rendere indicizzabili i siti Web;
- Visualizzare e aggiornare le informazioni in modo dinamico;
- E tanto altro ancora.

PHP è facile da imparare. È indipendente dalla piattaforma quindi, lo stesso codice, funziona sia su Windows, che su Mac OS X che su Linux e tutto il software che devi sviluppare.

PHP è open source, quindi è gratuito. È una tra le tecnologie più utilizzate per la creazione di siti Web dinamici, ma uno dei suoi obiettivi principali era quello di creare un libro che raccogliesse le informazioni da un modulo online per visualizzarle su una pagina web.

PHP viene utilizzato per creare contenuti dinamici da oltre l'80% dei 10 milioni di siti Web che analizza regolarmente. È il linguaggio che muove i più famosi sistemi di gestione dei contenuti CMS come Drupal, Joomla! e WordPress. Oltre a questi viene usato anche da Facebook e Wikipedia.

Già questo basterebbe per dare fiducia ad un linguaggio ma una delle grandi attrazioni di PHP, tuttavia, è che rimane fedele alle sue radici. Il creatore originale di PHP, Rasmus Lerdorf, lo descrisse come: *«un linguaggio di scripting molto adatto ai programmatori, adatto a persone con poca o nessuna esperienza di programmazione così come allo sviluppatore Web esperto che ha bisogno di risolvere problemi rapidamente»*. Puoi iniziare a scrivere script interessanti senza dover

imparare molta teoria, ma sai che la stessa tecnologia può essere usata anche a livello industriale o per progetti molto grandi.

Un breve sguardo al mondo PHP

PHP è stato originariamente progettato per essere incorporato nell'HTML di una pagina Web, ed è così che viene spesso utilizzato. Ad esempio, se si desidera visualizzare l'anno corrente in un avviso di copyright, è possibile inserirlo nel piè di pagina:

```
<p>&copy; <?php echo date('Y'); ?> Copyright</p>
```

Su un server Web abilitato per PHP, il codice tra i tag <?php e ?> viene elaborato automaticamente e visualizza l'anno corrente in questo modo:

© 2020 Copyright

Questo è solo un esempio banale, ma illustra alcuni dei vantaggi dell'utilizzo di PHP:

- Ogni utente vedrà l'anno corretto perché è il server che lo imposta;
- La data viene calcolata dal server Web, quindi non viene influenzata dall'orologio del computer dell'utente che potrebbe essere impostato in modo errato.

Sebbene sia conveniente incorporare il codice PHP in HTML in questo modo, è ripetitivo e può causare errori. Può anche rendere difficile la gestione delle tue pagine Web, in particolare quando inizi a utilizzare un codice PHP più complesso. Di conseguenza, è buona norma memorizzare il codice in file separati e quindi utilizzare PHP per creare le tue pagine dai diversi componenti.

Questi file separati possono contenere solo PHP, solo HTML o una combinazione di entrambi creando delle vere e proprie componenti

riusabili.

A titolo di esempio, puoi inserire il menu di navigazione del tuo sito Web in un file da includere e utilizzare PHP per includerlo in ogni pagina. Ogni volta che è necessario apportare modifiche al menù, si modifica solo un file e le modifiche si riflettono automaticamente in ogni pagina che include il menu. Immagina quanto tempo puoi risparmiare in un sito Web con centinaia di pagine!

Con una normale pagina HTML, il contenuto viene fissato dallo sviluppatore Web in fase di progettazione e caricato sul server Web. Quando qualcuno visita la pagina, il server Web invia semplicemente l'HTML e altre risorse, come le immagini e i fogli di stile. È un'architettura molto semplice: la richiesta proviene dal browser e il contenuto, impostato dal programmatore, viene rispedito dal server. Quando crei pagine Web con PHP, non è così semplice infatti quando viene visitato un sito Web basato su PHP, viene attivata la seguente sequenza di eventi:

1. Il browser invia una richiesta al server web;
2. Il web server passa la richiesta al motore PHP, che è incorporato nel server;
3. Il motore PHP elabora il codice. In molti casi, vengono eseguite anche delle query su un database;
4. Il server invia la pagina completa al browser.

Questo processo di solito richiede solo qualche frazione di secondo, quindi è improbabile che il visitatore di un sito Web PHP noti alcun ritardo. Poiché ogni pagina è costruita individualmente, i siti PHP possono rispondere all'input dell'utente, visualizzando contenuti specifici, ad esempio, dopo aver effettuato l'accesso al sito.

PHP è una lingua lato server ed il codice PHP rimane sul server web. Dopo che è stato elaborato, il server invia solo l'output dello script al browser del visitatore. Normalmente si tratta di HTML, ma PHP può essere utilizzato anche per generare altri linguaggi Web, come JSON (JavaScript Object Notation).

PHP ti consente di introdurre una logica nelle tue pagine web basata su alternative. Alcune decisioni vengono prese utilizzando le informazioni che PHP raccoglie dal server: la data, l'ora, il giorno della settimana, le informazioni nell'URL della pagina e così via. Immaginiamo di creare un

sito Web con i programmi TV del giorno corrente, siamo sicuri che se è mercoledì, mostrerà i programmi TV del mercoledì. Altre volte, le decisioni si basano sull'input dell'utente, che PHP estrae dai moduli online. Se ti sei registrato ad un sito, verranno visualizzate informazioni personalizzate come nome, cognome, immagine del profilo ecc. Tutto questo è sicuro perché il sito Web riceve solo l'output di uno script ma non sa come è fatto lo script.

Ma parliamo per un secondo di sicurezza.

PHP è come l'elettricità o i coltelli da cucina della tua casa: gestiti correttamente, sono molto sicuri; gestiti in modo irresponsabile, possono causare molti danni. PHP non è pericoloso, né tutti devono diventare esperti di sicurezza per utilizzarlo. L'importante è comprendere il principio di base della sicurezza PHP: controllare sempre l'input dell'utente prima di elaborarlo. La maggior parte dei rischi per la sicurezza può essere eliminata con il minimo sforzo. Il modo migliore per proteggerti è capire il codice che stai utilizzando e metterti nei panni di un utente malevolo che vuole rubare dati dal tuo server.

Gli strumenti

Per essere precisi, non è necessario alcun software speciale per scrivere script PHP.

Infatti il codice PHP è un semplice testo e può essere creato in qualsiasi editor di testo, come Blocco note su Windows o TextEdit su Mac OS X. Detto questo, programmare sarà molto più semplice se usi un software con funzionalità progettate per accelerare la processo di sviluppo. Potrai contare su un ventaglio molto ampio, sia gratuiti che a pagamento.

Grazie a questi strumenti se c'è un errore nel tuo codice, te ne potrai accorgere ben prima di arrivare al browser bensì direttamente in fase di programmazione e non in fase di test. Dovresti scegliere un editor che abbia le seguenti caratteristiche:

- Colorazione della sintassi PHP: il codice viene evidenziato in diversi colori in base al ruolo che svolge. Se il tuo codice ha un colore rosso o inaspettato, molto probabilmente hai commesso un errore;
- Controllo della sintassi PHP: i correttori di sintassi monitorano il codice durante la digitazione e evidenziano subito gli errori, risparmiando molto tempo e frustrazioni;
- Numerazione delle righe: trovare rapidamente una riga specifica semplifica notevolmente la risoluzione dei problemi;
- Suggerimenti sul codice PHP: PHP ha così tante funzioni integrate che può essere difficile ricordare come usarle, anche per un utente esperto. Molti editor di script visualizzano automaticamente i suggerimenti con promemoria su come funziona o sui parametri in ingresso;
- Una funzione per bilanciare le parentesi: parentesi (`()`), parentesi quadre (`[]`) e parentesi graffe (`{ }`) devono sempre trovarsi in coppie corrispondenti. È facile dimenticare di chiudere un paio, tuttavia, gli editor di script ti aiutano a risolvere questo problema.

Anche se non hai intenzione di sviluppare molto codice in PHP, dovresti prendere in considerazione l'uso di un editor dedicato se il tuo programma di sviluppo Web non supporta il controllo della sintassi. I seguenti editor hanno tutte le funzionalità essenziali, come il controllo della sintassi e suggerimenti sul codice. Non è un elenco esaustivo, ma piuttosto basato sull'esperienza personale.

- Atom: è un IDE davvero personalizzabile, esiste un plugin per tutto, PHP incluso;
- Komodo: IDE gratuito e open source per PHP e una serie di altri linguaggi informatici popolari. È disponibile per Windows, Mac OS X e Linux;
- PhpStorm: è programma di editing PHP dedicato ed offre un eccellente supporto per HTML, CSS e JavaScript. Attualmente è il mio programma preferito per lo sviluppo con PHP;
- Sublime: se sei un fan di Sublime Text, ci sono plug-in per la colorazione della sintassi PHP, il controllo della sintassi e la documentazione;
- Zend Studio: se sei davvero serio sullo sviluppo di PHP, Zend Studio è l'ambiente di sviluppo integrato (IDE) più completo per PHP. Zend Studio funziona su Windows, Mac OS X e Linux. In passato era costoso, ma il prezzo per i singoli sviluppatori è ora molto più conveniente.

Ora siamo pronti per partire ed iniziare a sviluppare.

L'INSTALLAZIONE

Ora che hai deciso di utilizzare PHP per arricchire le tue pagine web, devi assicurarti di avere tutto il necessario per andare avanti. Anche se puoi testare tutto sul tuo server remoto, di solito è più conveniente testare le pagine PHP sul tuo computer locale: tutto ciò che devi installare è gratuito. In questo capitolo, spiegherò le varie opzioni per Windows e Mac OS X. I componenti necessari sono normalmente installati di default su Linux.

Il modo più semplice per scoprire se il tuo sito Web supporta PHP, è chiedere alla tua società di hosting ma solitamente PHP è supportato di default. L'altro modo per scoprirlo è caricare una pagina PHP sul tuo sito Web e vedere se funziona. Anche se sai già che il tuo sito supporta PHP, fai il seguente test per vedere quale versione è in esecuzione:

1. Aprire un editor di testo, ad esempio Blocco note o TextEdit, e digitare il seguente codice in una pagina vuota: `<? php echo phpversion (); ?>`
2. Salvare il file come *phpversion.php*. È importante assicurarsi che il sistema operativo in uso non aggiunga l'estensione *.txt* dopo *.php*. Gli utenti Mac dovrebbero inoltre assicurarsi che TextEdit non salvi il file in formato RTF (Rich Text Format);
3. Caricare *phpversion.php* sul tuo sito web come faresti con una pagina HTML, quindi digita l'URL in un browser. Supponendo che tu abbia caricato il file al livello più alto del tuo sito, l'URL sarà simile a <http://www.tuosito.it/phpversion.php>. Se sullo schermo vedi un numero diviso in tre parti come 5.6.1 PHP è abilitato. Il numero indica quale versione di PHP è in esecuzione sul tuo server. È necessario disporre almeno della versione 5.4.0 per le funzioni di questo ebook;
4. Se ricevi un messaggio di errore probabilmente PHP è supportato ma hai commesso un errore nel digitare il codice nel file;
5. Se vedi solo il codice che hai inserito senza alcuna versione di PHP, significa che PHP non è supportato.

Se il tuo server esegue PHP 5.3 o precedente, contattalo per informarlo che desideri la versione più recente di PHP.

A differenza delle normali pagine Web, non puoi semplicemente fare doppio clic sulle pagine PHP e visualizzarle nel tuo browser. Queste pagine devono essere analizzate o elaborate attraverso un server Web che supporta PHP. Se la tua società di hosting supporta PHP, puoi caricare i tuoi file sul tuo sito Web e testarli in remoto, tuttavia, devi caricare il file ogni volta che apporti una modifica. Se vuoi lavorare subito con PHP, usa il tuo sito web come banco di prova. Scoprirai presto la necessità di un ambiente di test PHP in locale quindi vediamo come fare.

Per testare le pagine PHP sul tuo computer locale, devi installare quanto segue:

- Un server Web: un software che visualizza pagine Web che può risiedere nel tuo stesso pc;
- PHP;
- MySQL e un front-end per MySQL chiamato *phpMyAdmin*, necessari per funzionare con un database.

Per molti anni, ho raccomandato di installare separatamente ogni componente di un ambiente di test PHP, piuttosto che utilizzare un pacchetto contenente *Apache*, *PHP*, *MySQL* e *phpMyAdmin* in un'unica operazione. All'inizio erano facili da installare ma erano quasi impossibili da disinstallare o aggiornare. I pacchetti all-in-one attualmente disponibili sono eccellenti e non esito a raccomandarli ora infatti sui miei computer, utilizzo **XAMPP** per Windows e **MAMP** per Mac OS X. Sono disponibili altri pacchetti, non importa quale scegli.

Windows

Per impostazione predefinita, la maggior parte dei computer Windows nasconde l'estensione di tre o quattro lettere, come *.doc* o *.html*, quindi tutto ciò che vedi nelle finestre di dialogo così come in Esplora file di Windows nasconde la sua estensione. Windows 8 visualizza l'estensione del file per i file PHP, ma è utile attivare la visualizzazione dell'estensione del nome per tutti i file. Utilizza queste istruzioni per abilitare la visualizzazione delle estensioni di file in Windows 8/10:

1. Apri Esplora file;
2. Seleziona Visualizza per espandere la barra multifunzione nella parte superiore della finestra Esplora file;
3. Selezionare la casella di controllo "Estensioni nome file".

Visualizzando l'estensione dei file sei più al sicuro: puoi sapere se un documento dall'aspetto innocente è in realtà un virus.

La maggior parte delle installazioni PHP vengono eseguite sul server Web Apache. Entrambi sono open source e funzionano bene insieme. Tuttavia, Windows ha un proprio server Web, Internet Information Services (IIS), che supporta anche PHP. Microsoft ha lavorato a stretto contatto con il team di sviluppo di PHP per migliorare le prestazioni di PHP su IIS eguagliando Apache. Quindi, quale dovresti scegliere? La risposta dipende dal fatto che si sviluppino pagine Web utilizzando ASP o ASP.NET o che si intenda farlo. A meno che tu non abbia bisogno di IIS per ASP o ASP.NET, ti consiglio di installare Apache, usando XAMPP o uno degli altri pacchetti all-in-one. Se è necessario utilizzare IIS, il modo più conveniente per installare PHP è utilizzare il programma di installazione della piattaforma Web Microsoft (Web PI).

Esistono tre pacchetti popolari per Windows che installano Apache, PHP, MySQL, phpMyAdmin e molti altri strumenti sul tuo computer in un'unica operazione: XAMPP, WampServer e EasyPHP. Il processo di installazione richiede normalmente solo pochi minuti e, una volta installato il pacchetto, potrebbe essere necessario modificare alcune impostazioni, come vedremo in seguito.

MacOS X

Sia il web server Apache sia PHP sono preinstallati su Mac OS X, ma non sono abilitati per impostazione predefinita. Al posto di utilizzare le versioni preinstallate, ti consiglio di utilizzare **MAMP**, che installa Apache, PHP, MySQL, phpMyAdmin e molti altri strumenti in una sola operazione. Per evitare conflitti con le versioni preinstallate di Apache e PHP, MAMP individua tutte le applicazioni in una cartella dedicata sul disco rigido semplificando la disinstallazione di tutto semplicemente trascinando la cartella MAMP nel Cestino, qualora necessario.

Prima di iniziare, assicurati di aver effettuato l'accesso al tuo computer con privilegi di amministratore.

- Scarica l'immagine gratuita di MAMP;
- Al termine del download, avvia l'immagine del disco e prosegui accettando il contratto di licenza. Prosegui per continuare con il montaggio dell'immagine del disco;
- Segui le istruzioni visualizzate;
- Verifica che MAMP sia stato installato nella cartella Applicazioni.

MAMP di default utilizza porte non standard per Apache e MySQL. A meno che non si utilizzino più installazioni di Apache e MySQL, è necessario modificare le impostazioni della porta come segue:

1. Fare doppio clic sull'icona MAMP in Applicazioni / MAMP. Il tuo browser predefinito dovrebbe avviarsi e presentarti la pagina di benvenuto di MAMP. Nota che l'URL nella barra degli indirizzi del browser inizia con localhost:8888. Questo indica che il sistema gira in locale e sulla porta 8888 ovvero indica che Apache è in attesa di richieste sulla porta non standard 8888;
2. Apri il pannello di controllo MAMP, che dovrebbe essere in esecuzione sul desktop. Le piccole luci verdi vicino ad Apache Server e MySQL Server indicano che entrambi i server sono in esecuzione;

3. Fai clic sull'icona *Preferenze* e selezionare il tab *Porte* nella parte superiore del pannello. Qui vedrai le porte su cui Apache e MySQL sono in esecuzione ovvero 8888 e 8889;
4. Fai clic su "Imposta le porte Web e MySQL su 80 e 3306" per impostare la configurazione standard;
5. Fai clic su OK e inserisci la password del Mac quando richiesto in modo che MAMP possa riavviare entrambi i server;
6. Quando entrambe le spie sono di nuovo verdi, fare clic su "Apri pagina iniziale" nel pannello di controllo di MAMP. Questo ricarica la pagina di benvenuto di MAMP nel tuo browser. Questa volta, l'URL non dovrebbe avere due punti seguiti da un numero dopo localhost perché Apache ora è in ascolto sulla porta predefinita.

Dove sono i file?

È necessario creare i file in una posizione in cui il server Web possa elaborarli. Normalmente, ciò significa che i file devono trovarsi ad un livello alto del server o in una sottocartella di quel livello. Il percorso predefinito per le impostazioni più comuni è il seguente:

- XAMPP: C:\xampp\htdocs
- WampServer: C:\wamp\www
- EasyPHP: C:\EasyPHP\www
- IIS: C:\inetpub\wwwroot
- MAMP: Macintosh HD:Applications:MAMP\htdocs

Per visualizzare una pagina PHP, è necessario caricarla in un browser utilizzando un URL. L'URL per la pagina principale del server Web nell'ambiente locale è <http://localhost/>.

Verificare l'installazione


Dopo aver installato PHP, è una buona idea controllare le sue impostazioni di configurazione. Oltre alle funzionalità principali, PHP ha un gran numero di estensioni opzionali. Entrambi i pacchetti all-in-one e Microsoft Web PI installano tutte le estensioni necessarie per questo ebook. Tuttavia, alcune delle impostazioni di configurazione di base potrebbero essere leggermente diverse. Per evitare problemi imprevisti, controlla che la tua configurazione di PHP sia uguale alla seguente in modo che corrisponda alle impostazioni consigliate.

PHP ha un comando integrato, `phpinfo()`, che mostra i dettagli di come PHP è configurato sul server. La quantità di dettagli prodotti da questo comando può sembrare un enorme sovraccarico di informazioni ma è preziosa per determinare perché qualcosa funziona perfettamente sul tuo computer locale ma non sul tuo sito web. Di solito il problema si trova nel server remoto che ha disabilitato una funzione o non ha installato un'estensione opzionale. I pacchetti all-in-one semplificano l'esecuzione di `phpinfo()`:

- XAMPP: fai clic sul collegamento `phpinfo` nel menu a sinistra della schermata di benvenuto di XAMPP;
- MAMP: fai clic su `phpinfo` nel menu principale nella parte superiore della pagina iniziale di MAMP;
- WampServer: apri il menu WampServer e fai clic su Localhost. Il collegamento per `phpinfo()` si trova in Strumenti.

La pagina che troverai sarà qualcosa di simile a questa:

PHP Version 7.3.18



Build Date	May 12 2020 10:48:37
Compiler	MSVC15 (Visual C++ 2017)
Architecture	x64
Configure Command	script configure --enable-snapshot-build --enable-debug-pack --with-pdo-oci --with-snap-build-deps_auroraora4instantclient_12_16db_shared --with-oci-12c --with-snap-build-deps_auroraora4instantclient_12_16db_shared --enable-openssl --enable-openssl --enable-openssl --without-analyzer --with-pgo
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\xampp2020\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20180731
PHP Extension	20180731
Zend Extension	320180731
Zend Extension Build	AP320180731.TS.VC15
PHP Extension Build	AP20180731.TS.VC15
Debug Build	no
Thread Safety	enabled
Thread API	Windows Threads
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, https, phar

Dovresti eseguire `phpinfo()` anche sul tuo server remoto per verificare quali funzionalità sono abilitate. Se le estensioni elencate non sono supportate, parte del codice in questo ebook non funzionerà quando caricherai i tuoi file sul tuo sito web. Qualora dovessi trovare una configurazione diversa sarà necessario modificare il file `php.ini`. Si tratta di un file molto lungo, che tende a turbare i nuovi arrivati nella programmazione, ma non c'è nulla di cui preoccuparsi. È un semplice testo e uno dei motivi della sua lunghezza è che contiene molti commenti che spiegano le varie opzioni. Detto questo, è una buona idea fare una copia di backup prima di modificare `php.ini` in caso di errore. Dopo averlo aperto troverai delle righe che iniziano con un punto e virgola (;) che sono commenti. Le righe che devi modificare non iniziano con un punto e virgola. La maggior parte delle direttive sono precedute da uno o più esempi di come dovrebbero essere impostate. Assicurati di non modificare uno degli esempi commentati per errore. Per le direttive che usano On o Off, basta cambiare il valore in quello raccomandato. Ad esempio, se è necessario attivare la visualizzazione dei messaggi di errore, modificare questa riga:

```
display_errors = Off
```

cambiandolo in questo:

```
display_errors = On
```

Per impostare il livello di segnalazione degli errori, è necessario utilizzare le costanti PHP, scritte in maiuscolo e con distinzione tra maiuscole e minuscole. La direttiva dovrebbe apparire così:

```
error_reporting = E_ALL
```

Dopo aver modificato php.ini, salva il file e riavvia Apache o IIS in modo che le modifiche abbiano effetto. Se il server Web non si avvia, controlla il file di registro degli errori del server. Si può trovare nelle seguenti posizioni:

- XAMPP: nel pannello di controllo di XAMPP, fai clic sul pulsante Registri accanto ad Apache, quindi seleziona Apache error.log.
- MAMP: in Applicazioni: MAMP: registri, fare doppio clic su apache_error.log per aprirlo in Console.
- WampServer: nel menu WampServer, selezionare Apache poi Registro errori Apache.
- EasyPHP: fare clic con il pulsante destro del mouse sull'icona EasyPHP nella barra delle applicazioni e selezionare File di registro poi Apache;
- IIS: il percorso predefinito dei file di registro è C:\inetpub\logs.

La voce più recente nel registro degli errori, dovrebbe fornire un'indicazione di ciò che ha impedito il riavvio del server. Utilizza tali informazioni per correggere le modifiche apportate al file php.ini. Se il problema persiste, usa il file di backup che hai salvato prima delle modifiche. Ricomincia e controlla attentamente le modifiche che apporti al file.

CODICE PHP

A prima vista, il codice PHP può sembrare abbastanza complesso, ma una volta comprese le basi, scoprirai che la struttura è straordinariamente semplice. Se hai già lavorato con qualsiasi altro linguaggio di programmazione, come JavaScript o jQuery, scoprirai che hanno molto in comune.

Ogni pagina PHP deve avere quanto segue:

- L'estensione del file corretta, in genere *.php*;
- Tag di apertura e chiusura PHP che circondano ogni blocco di codice PHP.

Una tipica pagina PHP utilizzerà alcuni o tutti i seguenti elementi:

- Dichiarazioni condizionali per prendere decisioni;
- Funzioni o oggetti per eseguire attività preimpostate;
- Loop per eseguire attività in modo ripetuto;
- Matrici per contenere più valori;
- Variabili che fungono da segnaposto per valori sconosciuti o che cambiano.

Diamo una rapida occhiata a ciascuno di questi elementi, iniziando dal nome del file e dai tag di apertura e chiusura.

PHP è un linguaggio usato lato server cioè il server Web elabora il tuo codice PHP e invia solo i risultati, in genere sottoforma di HTML, al browser. Poiché tutte le azioni sono sul server, devi indicare che le tue pagine contengono codice PHP. Ciò comporta due semplici passaggi, vale a dire:

- Assegnare a ogni pagina un'estensione del nome file PHP; il valore predefinito è *.php*;
- Racchiudere tutto il codice PHP all'interno dei tag PHP.

Il tag di apertura è `<?php` e il tag di chiusura è `?>`. Se inserisci i tag sulla stessa riga del codice circostante, non è necessario che ci sia uno spazio prima del tag di apertura o dopo quello di chiusura, ma deve esserci uno spazio dopo il php nel tag di apertura in questo modo:

```
<p> Questo è HTML con PHP incorporato <? php // codice PHP?>.
</p>
```

Quando inserisci più di una riga di PHP, è una buona idea mettere il tag di apertura e di chiusura su righe separate per motivi di chiarezza.

```
<?php
// riga di codice PHP
// un'altra riga di codice PHP
?>
```

Potresti imbatterti in `<?` come versione breve alternativa del tag di apertura, tuttavia, questo tag non funziona su tutti i server. Usa `<?php` in modo da non avere alcun problema di compatibilità tra server.

PHP è un linguaggio che viene incorporato in pagine Web, ciò significa che è possibile inserire blocchi di codice PHP all'interno di normali pagine Web. Quando qualcuno visita il tuo sito e richiede una pagina PHP, il server la invia al motore PHP, che legge la pagina dall'alto verso il basso alla ricerca di tag PHP. L'HTML non viene alterato ma, ogni volta che il motore PHP incontra un tag `<?php`, inizia l'elaborazione del codice e continua fino a raggiungere il tag di chiusura `?>`. Se il codice PHP produce qualcosa in output, verrà inserito in quel punto della pagina.

Variabili

PHP fa uso anche di variabili, ovvero un nome che dai a qualcosa che può cambiare o che non conosci in anticipo. Le variabili in php iniziano sempre con \$ (un segno di dollaro). Sebbene il concetto di variabile sembri astratto, usiamo continuamente le variabili nella vita di tutti i giorni. Quando incontri qualcuno per la prima volta, una delle prime cose che chiedi è "Come ti chiami?" Non importa se la persona che hai appena incontrato è Antonio, Filippo o Gerry, la parola "nome" rimane costante. Allo stesso modo, con il tuo conto bancario, i soldi entrano ed escono tutto il tempo ma la quantità disponibile è sempre indicata come saldo. Quindi, "nome" e "saldo" sono variabili quotidiane. Metti un segno di dollaro davanti a loro e hai due variabili PHP già pronte, in questo modo:

\$nome

\$saldo

Puoi scegliere qualsiasi cosa ti piaccia come nome una variabile, purché tieni a mente le seguenti regole:

- Le variabili iniziano sempre con un simbolo di dollaro (\$);
- Il primo carattere dopo il simbolo del dollaro non può essere un numero;
- Non sono ammessi spazi o segni di punteggiatura, ad eccezione del carattere di sottolineatura (_).
- I nomi delle variabili fanno distinzione tra maiuscole e minuscole: \$saldoiniziale e \$saldoIniziale non sono uguali.

Quando si scelgono i nomi per le variabili, ha senso scegliere qualcosa che ti dica a cosa serve. Le variabili che hai visto finora sono dei buoni esempi. Poiché non puoi utilizzare spazi nei nomi delle variabili, è una buona idea scrivere in maiuscolo la prima lettera della seconda o delle parole successive parole quando le combini (detta notazione camelCase o a cammello). In alternativa, puoi usare un trattino basso (\$saldo_iniziale) detta notazione a serpente o snake_case.

Non cercare di risparmiare tempo utilizzando variabili molto brevi infatti l'uso di \$s, \$p, \$n e \$b rende il codice più difficile da capire e ciò

rende difficile la scrittura. Ancora più importante, rende gli errori più difficili da individuare. Come sempre, ci sono eccezioni ad una regola. Per convenzione, \$i, \$j e \$k vengono spesso utilizzati per tenere conto del numero di volte in cui è stato eseguito un ciclo e \$ è utilizzato nel controllo degli errori.

Le variabili ottengono i loro valori da diverse fonti, tra cui:

- Il risultato di un calcolo;
- Inclusione diretta nel codice PHP;
- Input dell'utente tramite moduli online;
- Un database;
- Una fonte esterna, come un feed di notizie o un file XML.

Ovunque provenga il valore, viene sempre assegnato con un segno di uguale (=), in questo modo:

```
$variabile = valore;
```

La variabile va a sinistra del segno di uguale e il valore va a destra e poiché assegna un valore, il segno di uguale viene chiamato operatore di assegnazione.

PHP è scritto come una serie di comandi o istruzioni. Ogni istruzione normalmente dice al motore PHP di eseguire una determinata azione, e deve sempre essere seguita da un punto e virgola, come questo:

```
<?php  
Fai questo;  
ora fai questo;  
?>
```

Come per tutte le regole, esiste un'eccezione: puoi omettere il punto e virgola se nel blocco di codice è presente una sola istruzione. Tuttavia, ti consiglio di non farlo perché a differenza di JavaScript, PHP non suppone automaticamente che ci dovrebbe essere un punto e virgola alla fine di una riga. Questo ha un piacevole effetto collaterale: puoi distribuire lunghe istruzioni su più righe e disporre il tuo codice per facilitarne la lettura. PHP, come HTML, ignora gli spazi bianchi nel codice. Al contrario, si basa sui punti e virgola per capire dove termina un comando e inizia quello successivo.

Gli array

In comune con altri linguaggi di calcolo, PHP consente di memorizzare più valori in un tipo speciale di variabile chiamato array. Immagina una matrice come una lista della spesa e sebbene ogni articolo possa essere diverso, puoi fare riferimento a essi collettivamente con un singolo nome. Immagina una variabile `$listaSpesa` che si riferisce collettivamente a cinque elementi: vino, pesce, pane, uva e formaggio.

I singoli elementi, o elementi dell'array, vengono identificati mediante un numero tra parentesi quadre che segue immediatamente il nome della variabile. PHP assegna il numero automaticamente, ma è importante notare che la numerazione inizia sempre da 0. Quindi il primo elemento dell'array, vino nel nostro esempio, viene indicato come `$listaSpesa[0]`, e non come `$listaSpesa[1]` come ci si potrebbe aspettare. Per lo stesso motivo, sebbene ci siano cinque elementi, l'ultimo (formaggio) si trova con `$listaSpesa[4]`. Il numero viene indicato come chiave o indice dell'array e questo tipo di array viene chiamato **array indicizzato**.

PHP utilizza un altro tipo di array in cui la chiave è una parola (o qualsiasi combinazione di lettere e numeri). Ad esempio, un array potrebbe apparire così:

```
$articolo['titolo']='Spugna per cucina';  
$articolo['produttore']='Azienda Pippo';  
$articolo['quantita'] = '100';  
$articolo['id'] = '9238-1-442-036-2';
```

Questo tipo di array è chiamato **array associativo**. Nota che la chiave dell'array è racchiusa tra virgolette (possono essere singole o doppie, non importa) e non deve contenere spazi o punteggiatura.

Le matrici sono una parte importante e utile di PHP infatti sono molto usate. Le matrici vengono ampiamente utilizzate con i database quando si recuperano i risultati di una ricerca che vengono storicizzati in una serie di matrici.

PHP ha diversi array integrati che vengono automaticamente compilati con informazioni utili. Sono chiamati **array superglobali** e iniziano tutti con un segno di dollaro seguito da un trattino basso. Due che vedrai frequentemente sono `$_POST` e `$_GET`. Contengono informazioni passate dai form attraverso in post tramite il protocollo HTTP (Hypertext Transfer

Protocol). Gli array superglobali sono tutti array associativi e le chiavi di `$_POST` e `$_GET` vengono automaticamente derivate dai nomi degli elementi del modulo o delle variabili in una stringa di query alla fine di un URL. Supponiamo che tu abbia un campo di input di testo chiamato "indirizzo" in un modulo; PHP crea automaticamente un elemento array chiamato `$_POST['indirizzo']` quando il modulo è inviato dal metodo post o `$_GET['indirizzo']` se usi il metodo GET. `$_POST['indirizzo']` contiene il valore immesso da un visitatore nel campo di testo, che consente di visualizzarlo sullo schermo, inserirlo in un database, inviarlo alla posta in arrivo o fare ciò che si desidera.

Valori speciali

Sebbene il testo debba essere racchiuso tra virgolette, esistono tre valori speciali - true, false e null – che non dovrebbero mai essere racchiusi tra virgolette a meno che non si desideri trattarli come testo (o stringhe). I primi due significano cosa ti aspetteresti; l'ultimo, null, significa "niente" o "nessun valore". Questi valori sono molto utili per PHP per prendere delle decisioni, ad esempio se il valore di una variabile è true esegui un comando.

Dai un'occhiata al seguente codice:

```
$variabile = false;
```

Fa esattamente quello che ti aspetti: rende \$variabile falso. Ora guarda questo:

```
$variabile = 'false';
```

Questo fa esattamente il contrario di quello che potresti aspettarti: rende \$variabile vero! Perché? Perché le virgolette attorno a false lo trasformano in una stringa e PHP considera le stringhe come vere. L'altra cosa da notare su questi valori è che non fanno distinzione tra maiuscole e minuscole. I seguenti esempi sono tutti validi:

```
$variabile = TRUE;
```

```
$variabile = tRuE;
```

```
$variabile = true;
```

Istruzioni condizionali

La vita è piena di decisioni così come PHP. Se hai esperienza con altri linguaggi di programmazione conosci già buona parte di questo sottocapitolo. Il processo decisionale in PHP utilizza dichiarazioni condizionali ed il più comune usa il `se`, avvicinandosi molto al linguaggio comune. Nella vita reale, potresti trovarti di fronte alla seguente decisione: se fa caldo, vado in spiaggia.

Nello pseudo-codice PHP, la stessa decisione è simile alla seguente:

```
if (fa caldo) {  
    vado in spiaggia;  
}
```

La condizione da testare è racchiusa tra parentesi tonde mentre l'azione è racchiusa tra parentesi graffe. Questo è il modello decisionale di base:

```
if (la condizione è vera) {  
    // codice da eseguire se la condizione è vera  
}
```

Il codice all'interno delle parentesi graffe viene eseguito solo se la condizione è vera. Se è falso, PHP ignora tutto ciò che è tra parentesi graffe e passa alla sezione successiva del codice. Nel codice precedente abbiamo inserito anche un commento, inizia con i simboli `//`.

A volte, l'istruzione `if` è tutto ciò di cui hai bisogno, ma spesso vuoi che venga invocata un'azione se la condizione non è soddisfatta. Per fare questo, usa `else`, in questo modo:

```
if (la condizione è vera) {  
    // codice da eseguire se la condizione è vera  
} else {  
    // codice da eseguire se la condizione è falsa  
}
```

Se vuoi più alternative, puoi aggiungere più istruzioni condizionali come questa:

```
if (la condizione è vera) {  
    // codice da eseguire se la prima condizione è vera  
} elseif (la seconda condizione è vera) {  
    // codice da eseguire se la prima condizione fallisce ma la seconda  
    condizione è vera
```

```
} else {  
    // codice se entrambe le condizioni sono false  
}
```

Le dichiarazioni condizionali sono utili solo per verificare che la condizione testata è vera infatti se non è vera, deve essere falsa. Le condizioni dipendono spesso dal confronto di due valori. Questo valore è maggiore di quello? Sono uguali? E così via.

Per verificare l'uguaglianza, PHP utilizza due segni di uguale (==), in questo modo:

```
if ($profilo == 'amministratore') {  
    // invia alla pagina di amministrazione  
} else {  
    // invia alla pagina utente  
}
```

Allo stesso modo puoi usare i simboli maggiore di (>), minore di (<), maggiore o uguale (>=) o minore uguale di (<=), diverso da (!=) per comparare dei valori.

Spesso, confrontare due valori non è sufficiente. PHP consente di impostare una serie di condizioni utilizzando operatori logici per specificare se tutti o solo alcuni devono essere soddisfatti.

Gli operatori logici più importanti in PHP sono elencati nella seguente tabella:

&&
And
\$a && \$b
Equivale a vero se entrambi \$a e \$b sono veri
Or
\$a \$b
Equivale a vero se \$a o \$b sono true; altrimenti, falso
!
Not
!\$a
Equivale a vero se \$a non è true

Mostrare risultati

Ovviamente è possibile visualizzare i risultati nella tua pagina web e per farlo esistono due modi in PHP: usare echo o print. Ci sono alcune sottili differenze tra i due, ma puoi considerarli equivalenti. È possibile utilizzare echo con variabili, numeri e stringhe; mettilo semplicemente davanti a qualunque cosa tu voglia visualizzare, in questo modo:

```
$nome = 'Filippo';  
echo $nome; // visualizza Filippo  
echo 5; // visualizza 5  
echo 'David'; // visualizza David
```

Quando si utilizzano echo e print con una variabile, funzionano solo con variabili che contengono un singolo valore. Non è possibile utilizzarli per visualizzare il contenuto di un array o di un'estrazione da un database. Per fare ciò sono necessari i loop anche detti cicli: usi echo o print all'interno del loop per visualizzare ogni elemento singolarmente. Vedremo qualche esempio di questo tipo nei capitoli successivi.

Potresti vedere degli script che usano le parentesi con echo e print, in questo modo:

```
echo ( 'David'); // visualizza David
```

Le parentesi non fanno alcuna differenza e a meno che non ti piaccia scriverle per il gusto di farlo, non utilizzarle.

Quando si desidera visualizzare il valore di una singola variabile o espressione è possibile utilizzare un tag PHP abbreviato in questo modo:

```
<p> Il mio nome è <? = $nome; ?>. </p>
```

Questo produce esattamente lo stesso output di questo:

```
<p> Il mio nome è <? php echo $nome; ?>. </p>
```

Poiché si tratta di una scorciatoia, nessun altro codice può trovarsi nello stesso blocco PHP, ma è particolarmente utile quando si incorporano i risultati del database in una pagina Web. È ovvio che il valore della

variabile deve essere impostato in un precedente blocco PHP prima di poter utilizzare questo tipo di visualizzazione.

TIPI DI DATI

PHP è ciò che è noto come un linguaggio **debolmente tipizzato**, in pratica, a differenza di altri linguaggi di programmazione (ad esempio Java o C #), a PHP non importa quale tipo di dati archiviate in una variabile.

Il più delle volte, questo è molto conveniente, anche se devi stare attento con l'input dell'utente. Potresti aspettarti che un utente inserisca un numero in un campo di un form, ma PHP non restituirà un errore se l'utente inserisce una parola. Per questo motivo è fondamentale controllare attentamente l'input dell'utente. Sebbene PHP sia tipizzato in modo debole, utilizza i seguenti otto tipi di dati:

- **Integer:** un numero intero, ad esempio 1, 25, 42 o 2006. I numeri interi non devono contenere virgole o punteggiatura per i separatori. Puoi anche usare numeri esadecimali, che dovrebbero essere preceduti da 0x (ad es. 0xFFFFF, 0x00000).
- **Floats:** un numero che contiene un punto decimale, come 9.99, 98.6 o 2.1. PHP non supporta l'uso della virgola come punto decimale, comune in molti paesi europei infatti devi usare un punto. Come per i numeri interi, i numeri in virgola mobile non devono separatori per le migliaia.
- **String:** una stringa è un testo di qualsiasi lunghezza. Può essere breve con zero caratteri (una stringa vuota) e non ha limiti superiori.
- **Boolean:** questo tipo ha solo due valori: true o false.
- **Array:** un array è una variabile in grado di memorizzare più valori, sebbene possa anche non contenerne affatto (un array vuoto). Gli array possono contenere qualsiasi tipo di dati, inclusi altri array. Un array di array è chiamato array multidimensionale.
- **Object:** un oggetto è un tipo di dati complesso in grado di memorizzare e manipolare valori.
- **Resource:** quando PHP si connette ad una fonte di dati esterna, come un file o un database, memorizza un riferimento ad esso come risorsa.

- **NULL**: questo è un tipo di dati speciale che indica che una variabile non ha valore.

Un importante effetto collaterale della tipizzazione debole di PHP è che se racchiudi un numero intero o in virgola mobile tra virgolette, PHP lo converte automaticamente da una stringa in un numero, permettendoti di eseguire calcoli senza la necessità di alcuna gestione speciale. Sotto questo aspetto è diverso da JavaScript e può avere conseguenze inattese. Quando PHP vede il segno più (+), presume che tu voglia eseguire l'addizione, e quindi prova a convertire stringhe in numeri interi o in virgola mobile, come nell'esempio seguente:

```
$frutti = '2 mele';  
$verdure = '2 carote';  
echo $frutti + $verdure; // visualizza 4
```

PHP vede che sia \$frutti che \$verdure iniziano con un numero, quindi estrae il numero e ignora il resto. Tuttavia, se la stringa non inizia con un numero, PHP la converte in 0, come mostrato in questo esempio:

```
$frutti = '2 mele';  
$verdure = 'e 2 carote';  
echo $frutti + $verdure; // visualizza 2
```

La tipizzazione debole è una croce e delizia ma rende PHP molto semplice per i principianti. Ricorda che spesso è necessario verificare che una variabile contenga il tipo di dati corretto prima di utilizzarlo.

I CICLI

While e do...while

Un ciclo è una sezione di codice che viene ripetuta fino a quando non viene soddisfatta una determinata condizione. I loop (o cicli) sono spesso controllati impostando una variabile che conta il numero di iterazioni. Aumentando la variabile di una ogni volta, il loop si ferma quando la variabile raggiunge un numero preimpostato. I loop sono inoltre controllati eseguendo ogni elemento di un array. Quando non ci sono più elementi da elaborare, il ciclo si interrompe. I loop contengono spesso istruzioni condizionali, quindi sebbene siano molto semplici nella struttura, possono essere utilizzati per creare codice che elabora i dati in modi complesso.

Il tipo più semplice di ciclo è chiamato ciclo while. La sua struttura di base si presenta così:

```
while (la condizione è vera) {  
    fai qualcosa;  
}
```

Il codice seguente mostra ogni numero compreso tra 1 e 100 in un browser. Inizia impostando una variabile (\$i) ad 1 e quindi utilizza la variabile come contatore per controllare il loop, oltre a visualizzare il numero corrente sullo schermo.

```
$i = 1; // imposta il contatore a 1  
while ($i <= 100) {  
    echo "$i <br>";  
    $i++; // aumenta il contatore di 1  
}
```

Una variante del ciclo while utilizza la parola chiave do e segue questo modello di base:

```
do {  
    codice da eseguire  
} while (condizione da testare);
```

La differenza tra un do...while e un ciclo while è che il codice all'interno del blocco do viene eseguito almeno una volta, anche se la condizione non è mai vera. Il seguente codice mostra il valore di \$i una volta, anche se è maggiore del massimo previsto.

```
$i = 1000;  
do {  
echo "$i <br>";  
$i++; // aumenta il contatore di 1  
} while ($i <= 100);
```

Il pericolo con il do...while consiste nel dimenticare di impostare una condizione che porta alla fine del loop o l'impostazione di una condizione impossibile. Questo è noto come un ciclo infinito che blocca il computer o provoca l'arresto anomalo del browser.

For e foreach

Il ciclo for è meno incline a generare un ciclo infinito perché è necessario dichiarare tutte le condizioni del ciclo nella prima riga. Il ciclo for utilizza il seguente modello di base:

```
for (inizializzazione; condizione; codice da eseguire dopo ogni iterazione) {  
    codice da eseguire  
}
```

Il codice seguente fa esattamente lo stesso del ciclo while precedente, mostrando tutti i numeri da 1 a 100:

```
for ($i = 1; $i <= 100; $i++) {  
    echo "$i <br>";  
}
```

Le tre espressioni tra parentesi controllano l'azione del ciclo (nota che sono separate da punti e virgola, non da virgole):

1. La prima espressione viene eseguita prima dell'inizio del loop. In questo caso, imposta il valore iniziale della variabile contatore \$i su 1.
2. La seconda espressione imposta la condizione che determina per quanto tempo il ciclo dovrebbe continuare a funzionare. Può essere un numero fisso, una variabile o un'espressione che calcola un valore.
3. La terza espressione viene eseguita alla fine di ogni iterazione del ciclo. In questo caso, aumenta di \$i di 1, ma non c'è nulla che ti impedisca di incrementare di 4, 10 o 50. Ad esempio, la sostituzione di \$i++ con \$i+=10 in questo esempio visualizzerà 1, 11, 21, 31 e così via.

L'ultimo tipo di ciclo in PHP viene utilizzato con matrici e oggetti. Assume due forme diverse, entrambe utilizzano variabili temporanee per gestire ciascun elemento. Se devi usare solo il valore dell'elemento, il ciclo foreach assume la forma seguente:

```
foreach (nomeVariabile as elemento) {
```

```
fai qualcosa con l'elemento  
}
```

L'esempio seguente crea un ciclo sull'array \$listaSpesa e visualizza il nome di ciascun elemento:

```
$listaSpesa = ['vino', 'pesce', 'pane', 'uva', 'formaggio'];  
foreach ($ listaSpesa as $item) {  
    echo $item. '<br>';  
}
```

Sebbene l'esempio precedente utilizzi un array indicizzato, è anche possibile utilizzare la forma semplice del ciclo foreach con un array associativo. La forma alternativa del ciclo foreach consente di accedere sia alla chiave che al valore di ciascun elemento. Prende questa forma leggermente diversa:

```
foreach (nomeVariabile as chiave => valore) {  
    fai qualcosa con chiave e valore }
```

Per terminare prematuramente un ciclo quando viene soddisfatta una determinata condizione, inserire la parola chiave break all'interno di un'istruzione condizionale. Non appena lo script viene interrotto, si uscirà dal ciclo.

Per saltare la valutazione del codice in un ciclo quando viene soddisfatta una determinata condizione, utilizzare la parola chiave continue. Invece di uscire dal ciclo, ritorna all'inizio del ciclo e si occupa dell'elemento successivo. Ad esempio, il ciclo seguente salta l'elemento corrente se \$foto non ha valore:

```
foreach ($immagini as $foto) {  
    if (empty ($foto)) continue;  
    // codice per visualizzare una foto  
}
```


LE FUNZIONI

Le funzioni offrono un modo conveniente per eseguire operazioni eseguite di frequente. Oltre al gran numero di funzioni integrate, PHP ti consente di creare le tue. Il vantaggio principale consiste nello scrivere il codice una sola volta, piuttosto che doverlo scrivere nuovamente ovunque tu ne abbia bisogno. Non solo accelera lo sviluppo, ma semplifica anche la lettura e la manutenzione del codice. Se c'è un problema con il codice nella tua funzione, puoi aggiornarlo in un solo punto anziché cercare ovunque. Inoltre, le funzioni di solito accelerano l'elaborazione delle tue pagine.

Costruire le tue funzioni in PHP è facile. È sufficiente racchiudere un blocco di codice in una coppia di parentesi graffe e utilizzare la parola chiave `function` per dare un nome alla nuova funzione. Il nome della funzione è sempre seguito da una coppia di parentesi. Il seguente esempio mostra la struttura di base di una funzione personalizzata:

```
function saluta() {  
    echo 'Ciao!';  
}
```

Puoi invocare questa funzione semplicemente scrivendo `saluta()`; in un blocco di codice PHP e sullo schermo vedrai `Ciao!`. Questo tipo di funzione esegue sempre esattamente la stessa operazione. Affinché le funzioni rispondano alle circostanze, è necessario passare loro dei valori come argomenti.

Supponiamo che tu voglia adattare la funzione in modo che mostri il nome di qualcuno. Puoi farlo inserendo una variabile tra parentesi nella dichiarazione di funzione. La stessa variabile viene quindi utilizzata all'interno della funzione per visualizzare qualunque valore venga passato alla funzione. Per passare più di un argomento a una funzione, separare le variabili con virgole tra parentesi aperte. Ecco come appare la funzione adesso:

```
function saluta($nome) {  
    echo "Ciao $nome!";  
}
```

Ora puoi usare questa funzione all'interno di una pagina per visualizzare il valore di qualsiasi variabile passata a `saluta()`. Ad esempio, se hai un

modulo online che salva il nome di qualcuno in una variabile chiamata \$visitatore e Antonio visita il tuo sito, puoi salutarlo in modo personalizzato invocando la funzione saluta(\$visitatore); nella tua pagina.

È anche importante capire che le funzioni creano un ambiente separato che è come una scatola nera. Normalmente ciò che accade all'interno della funzione non ha alcun impatto sul resto dello script, a meno che non restituisca un valore. Le variabili all'interno di una funzione rimangono esclusive della funzione. Questo esempio dovrebbe illustrare cosa intendo:

```
function duplica ($numero) {  
    $numero *= 2;  
    echo '$numero è '. $numero. '<br>';  
}  
$numero = 4;  
duplica($numero);  
echo '$numero è '. $numero;
```

Le prime quattro righe definiscono una funzione chiamata duplica(), che accetta un numero, lo raddoppia e lo visualizza sullo schermo. Il resto dello script assegna il valore 4 a \$numero. Quindi passa \$numero come argomento a duplica(). La funzione elabora \$numero e visualizza 8. Al termine della funzione, \$numero viene visualizzato sullo schermo dalla funzione echo. Questa volta il valore sarà 4 e non 8 perché il valore non è stato restituito dalla funzione ma solo usato per la visualizzazione a schermo.

Ciò dimostra che \$numero nello script principale è totalmente estraneo alla variabile con lo stesso nome all'interno della funzione. Questo è noto come **scope** o ambito della variabile. Anche se il valore della variabile cambia all'interno di una funzione, le variabili con lo stesso nome all'esterno non sono interessate. Per evitare confusione, è una buona idea utilizzare nomi di variabili nel resto dello script diversi da quelli utilizzati all'interno delle funzioni. Questo non è sempre possibile, quindi è utile sapere che le funzioni sono delle piccole scatole nere e normalmente non hanno alcun impatto diretto sui valori delle variabili nel resto dello script. Un altro aspetto importante è che una funzione non può normalmente accedere ai valori nello script esterno a meno che non vengano passati alla funzione come argomenti.

Esiste più di un modo per una funzione per modificare il valore di una variabile data in input, ma il metodo più importante è utilizzare la parola

chiave return e assegnare il risultato alla stessa variabile o ad un'altra. Questo può essere dimostrato modificando la funzione duplica() in questo modo:

```
function duplica ($numero) {  
    return $numero *= 2;  
}  
$num = 4;  
$raddoppiato = duplica($num);  
echo '$num è: ' . $num . '<br>';  
echo '$raddoppiato è: ' . $raddoppiato;
```

Questa volta, ho usato nomi diversi per le variabili per evitare di confonderle. Ho anche assegnato il risultato di duplica(\$num) a una nuova variabile. Il vantaggio di fare questo è che sono ora disponibili sia il valore originale che il risultato del calcolo. Non sempre è necessario mantenere il valore originale, ma a volte può essere molto utile.

Sebbene le funzioni non cambino normalmente il valore delle variabili passate come argomenti, ci sono occasioni in cui si desidera cambiare il valore originale anziché avere un valore di ritorno. Per fare ciò, quando si definisce la funzione, si antepone il parametro che si desidera modificare con una e commerciale (&), in questo modo:

```
function duplica(&$numero) {  
    $numero *= 2;  
}
```

Nota che questa versione della funzione duplica() non stampa a video il valore della variabile, né restituisce il valore del calcolo. Poiché il parametro tra parentesi è preceduto da una e commerciale (&), il valore originale di una variabile passato come argomento alla funzione verrà modificato. Questo è noto come passaggio per riferimento.

Il seguente codice dimostra l'effetto:

```
$num = 4;  
echo '$num è: ' . $num . '<br>';  
duplica($num);  
echo '$num ora è: ' . $num;
```

Se la tua funzione viene trovata nella stessa pagina in cui viene utilizzata, non importa dove la dichiarare; può essere prima o dopo essere stata utilizzata. È una buona idea, tuttavia, dichiarare le funzioni insieme, nella parte superiore o inferiore di una pagina. Questo le rende più facili da

trovare e mantenere. Le funzioni utilizzate in più di una pagina vengono archiviate in modo ottimale in un file esterno incluso in ciascuna pagina. L'inclusione di file esterni avviene tramite `include` e `request`. Quando le funzioni sono memorizzate in file esterni, è necessario includere il file esterno prima di usare una funzione.

Require e include

La possibilità di includere il contenuto di un file all'interno di un altro è una delle funzionalità più interessanti di PHP ed è anche una delle più facili da implementare. La maggior parte delle pagine di un sito Web condivide elementi comuni, come intestazione, piè di pagina e menu di navigazione. Puoi modificare l'aspetto di questi elementi in tutto il sito modificando le regole di stile in un foglio di stile esterno. Ma i CSS hanno solo una capacità limitata di modificare il contenuto degli elementi della pagina. Se vuoi aggiungere un nuovo elemento al tuo menu, devi modificare il codice HTML per ogni pagina che lo visualizza.

PHP supporta le inclusioni lato server (SSI) ovvero un file esterno che contiene codice dinamico o HTML (o entrambi) che si desidera incorporare in più pagine. PHP unisce il contenuto in ciascuna pagina Web sul server e poiché ogni pagina utilizza lo stesso file esterno, puoi aggiornare un menu o un altro elemento comune modificando e caricando un singolo file con un grande risparmio di tempo.

La possibilità di includere codice da altri file è una parte fondamentale di PHP. Tutto ciò che è necessario è utilizzare uno dei comandi include di PHP ed indicare al server dove trovare il file.

PHP dispone di quattro comandi che possono essere utilizzati per includere il codice da un file esterno:

- include
- include_once
- require
- require_once

Fanno tutti praticamente la stessa cosa, quindi perché averne quattro a disposizione? La differenza fondamentale è che include cerca di continuare l'elaborazione di uno script, anche se manca il file esterno, mentre require è usato nel senso inverso: se il file manca, il motore PHP interrompe l'elaborazione e genera un errore. In termini pratici, questo significa che dovresti usare include se la tua pagina rimarrebbe utilizzabile anche senza il

contenuto del file esterno. Utilizza `require` se la pagina dipende dal file esterno.

Gli altri due comandi, `include_once` e `require_once`, funzionano allo stesso modo, ma impediscono che lo stesso file venga incluso più di una volta in una pagina, ciò è particolarmente importante quando si includono file che definiscono funzioni o classi. Il tentativo di definire una funzione o una classe più di una volta in uno script provoca un errore irreversibile. Pertanto, l'utilizzo di `include_once` o `require_once` garantisce che le funzioni e le classi siano definite una sola volta, anche se lo script tenta di includere il file esterno più di una volta.

Per includere un file esterno, usa uno dei quattro comandi seguito dal percorso del file come stringa: in altre parole, il percorso del file deve essere tra virgolette (singole o doppie, non importa). Il percorso del file può essere assoluto o relativo al documento corrente. Ad esempio, funzionerà in uno dei seguenti modi (purché esista il file di destinazione):

```
include 'includes/menu.php';  
include 'C:/xampp/htdocs/phpsols/includes/menu.php';  
include '/Applications/MAMP/htdocs/phpsols/includes/menu.php';
```

Quando si utilizza un percorso file relativo, si consiglia di utilizzare `./` per indicare che il percorso inizia nella cartella corrente. Pertanto, è più efficiente riscrivere il primo esempio in questo modo:

```
include './includes/menu.php';
```

Ciò che non funziona è l'utilizzo di un percorso di file relativo alla radice del sito, in questo modo:

```
include '/includes/menu.php';
```

Ecco come includere un footer comune a più pagine:

```
<?php include './includes/footer.php'; ?>
```

Quando il motore PHP rileva un comando `include`, interrompe l'elaborazione di PHP all'inizio del file esterno e riprende nuovamente alla fine. Se si desidera che il file esterno utilizzi il codice PHP, il codice deve essere racchiuso tra tag PHP. Poiché il file esterno viene elaborato come parte del file PHP che lo include, un file include può avere qualsiasi estensione.

Alcuni sviluppatori utilizzano `.inc` come estensione del nome file per chiarire che il file deve essere incluso in un altro file. Tuttavia, la maggior

parte dei server considera i file .inc come testo normale. Ciò comporta un rischio per la sicurezza se il file contiene informazioni riservate, come nome utente e password del database. Se il file è archiviato nella cartella principale del tuo sito Web, chiunque scopra il nome del file può semplicemente digitare l'URL nella barra degli indirizzi del browser e il browser visualizzerà tutti i dati sensibili!

D'altra parte, qualsiasi file con estensione .php viene automaticamente inviato al motore PHP per l'analisi prima di essere inviato al browser. Finché le tue informazioni sensibili si trovano all'interno di un blocco di codice PHP e in un file con estensione .php, queste non saranno esposte. Ecco perché alcuni sviluppatori utilizzano .inc.php come include in modo da avere una doppia estensione. La parte .inc ti ricorda che si tratta di un file include, ma i server sono interessati solo al .php alla fine, il che assicura che tutto il codice PHP sia analizzato correttamente.

Per molto tempo, ho seguito la convenzione di usare .inc.php per i file include. Ma dal momento che immagazzino tutti i miei file include in una cartella separata chiamata include, ho deciso che la doppia estensione è superflua infatti ora uso solo .php. Quale convenzione scegliere dipende da te, ma usare .inc è la meno sicura.

Per essere sicuri che una pagina esista puoi verificare tramite delle funzioni che PHP offre nativamente. Se una pagina sarebbe priva di significato senza il file include, è necessario reindirizzare l'utente a una pagina di errore se il file include è mancante.

Un modo per farlo consiste nel generare un'eccezione, in questo modo:

```
<?php
$file = './includes/menu.php';
if (file_exists($file) && is_readable($file)) {
include $file;
} else {
throw new Exception("$file non trovato");
}
?>
```

Una caratteristica utile dei file include di PHP è che possono essere posizionati ovunque, purché la pagina con il comando include sappia dove trovarli. I file da includere non è nemmeno necessario che siano all'interno della radice del tuo server web, ciò significa che è possibile proteggere i file che contengono informazioni riservate, come le password, in una directory

privata (cartella) a cui non è possibile accedere tramite un browser. Quindi, se la tua società di hosting fornisce un'area di archiviazione esterna alla radice del tuo server, dovresti prendere in considerazione l'opportunità di inserire lì i tuoi file da includere.

Ma perché non è possibile inserire un riferimento alla radice del sito? Quando si fa clic su un collegamento per passare a un'altra pagina, il percorso nel tag <a> indica al browser come passare dalla pagina corrente a quella successiva. La maggior parte degli strumenti di creazione web specifica il percorso relativo al documento corrente. Se la pagina di destinazione si trova nella stessa cartella, viene utilizzato solo il nome file. Se è superiore di un livello rispetto alla pagina corrente, il nome del file è preceduto da ../. Questo è noto come percorso o collegamento relativo al documento.

L'altro tipo di collegamento inizia sempre con una barra, che indica la radice del sito. Il vantaggio di un percorso relativo alla radice del sito è che non importa quanto sia profonda la pagina corrente nella gerarchia del sito, la barra (/) garantisce che il server Web cercherà dal livello superiore del sito. Sebbene i collegamenti relativi alla radice del sito siano molto più facili da leggere, PHP non riesce a gestirli.

È possibile convertire un percorso relativo alla radice del sito in uno assoluto concatenando la variabile globale `$_SERVER['DOCUMENT_ROOT']` all'inizio del percorso, in questo modo:

```
include $_SERVER['DOCUMENT_ROOT'].'/includes/nomefile.php';
```

La maggior parte dei server supporta `$_SERVER['DOCUMENT_ROOT']`, ma è necessario controllare la sezione Variabili PHP nei dettagli di configurazione visualizzati da `phpinfo()` per essere sicuri che funzioni.

OGGETTI

Probabilmente hai sentito dire che la maggior parte dei linguaggi di programmazione dà il meglio di sé prima di essere implementata.

Questo può essere vero per i linguaggi progettati secondo specifiche ambiziose, ma PHP non è sicuramente uno di quelli. I suoi umili inizi sono illustrati dal significato originario dell'acronimo PHP: Personal Home Page. È nato come un modo semplice per aggiungere contenuto dinamico alle pagine HTML ed è diventato un linguaggio di programmazione sempre più completo. L'orientamento agli oggetti non era originariamente parte del linguaggio, ma è gradualmente ha assunto sempre maggiore importanza. Oggi fornisce la maggior parte delle funzionalità che i programmatori si aspettano in un linguaggio orientato agli oggetti, pur mantenendo la sua dinamicità e lasciandoci comunque scegliere il nostro stile di programmazione. Ecco perché PHP è uno strumento importante nel tuo toolkit.

Ti consente di concentrarti maggiormente sull'ottimizzazione del design e sulla sua implementazione, invece di lottare per soddisfare le esigenze del linguaggio. Ad oggi sono state implementate in PHP la maggior parte delle funzionalità che esistono da molto tempo in Java e in altri linguaggi orientati agli oggetti. L'uso di queste nuove funzionalità è per lo più facoltativo; quindi, non sei costretto a uno stile di programmazione più simile a Java se sei abituato allo stile PHP e vuoi mantenerlo. Inizieremo questo capitolo esaminando le basi degli oggetti; quindi, esamineremo una delle funzionalità più utili in PHP: la gestione delle eccezioni. Dopodiché, ci assicureremo di comprendere la caratteristica più importante del modello a oggetti PHP: il fatto che gli oggetti siano trattati per riferimento. Ciò rende la gestione degli oggetti molto più naturale, infine, vedremo come manipolare magicamente le chiamate ai metodi utilizzando una delle funzionalità più avanzate introdotte in PHP.

Ci sono due chiavi per capire come funzionano gli oggetti e le classi. Uno è conoscere i meccanismi di scrittura di una classe e utilizzare i costrutti del linguaggio che supportano la programmazione orientata agli oggetti. L'altro argomento, più difficile e avanzato, è capire come far interagire gli oggetti in un modo che raggiunga l'obiettivo principale della programmazione orientata agli oggetti: codice manutenibile, ovvero la

progettazione orientata agli oggetti. L'idea qui è di iniziare con solo un po' di teoria e fare pratica per consolidarla prima di passare a idee più avanzate. Ma prima di iniziare, un breve commento esplicativo sul perché vogliamo confrontare PHP con Java.

Confronto con Java

Le ragioni del confronto sono pratiche; non ci sono giudizi di valore impliciti. Non c'è intenzione di fare una gara tra i due linguaggi o di implicare che Java sia l'unica o la migliore alternativa a PHP. Piuttosto, l'idea è di imparare qualcosa dal confronto e assicurarci di ottenere i dettagli giusti. La maggior parte della sintassi orientata agli oggetti in PHP è simile a Java. Poiché molte delle differenze sono sottili, è facile confondere i due linguaggi. Ad esempio, il costrutto dell'interfaccia è quasi identico nei due linguaggi ma in PHP, a differenza di Java, il costruttore può essere specificato nell'interfaccia proprio come gli altri metodi. Il confronto di due linguaggi di programmazione simili evidenzia e chiarisce i dettagli specifici di ciascuno. E poiché Java è innegabilmente un linguaggio estremamente popolare, è probabile che molti sviluppatori utilizzino entrambe i linguaggi e passino da uno all'altro.

Ci sono sviluppatori PHP che programmano in Java occasionalmente o frequentemente, ed è probabile che alcuni lettori siano programmatori che non conoscono PHP ma hanno una certa esperienza con Java.

Oggetti e classi

Gli aspetti di base, oggetti e classi, sono documentati nel manuale PHP, ma li analizzeremo brevemente e speriamo di ottenere una prospettiva leggermente diversa e fresca. Secondo il manuale, una classe è “una raccolta di variabili e funzioni che usano queste variabili”. Potrebbe essere questa frase concisa, sebbene sia del tutto possibile avere una classe senza variabili. Le funzioni sono chiamate metodi nella terminologia corretta object-oriented. Si potrebbe dire che una classe è come una casa. I metodi sono stanze e la dichiarazione di classe rappresenta i muri esterni della casa. Diverse attività si svolgono in stanze diverse: si può cucinare in cucina, dormire in camera da letto ecc. Allo stesso modo, ogni metodo in una classe svolge un lavoro specifico. Le pareti hanno un senso perché proteggono il codice all'interno della casa dal codice all'esterno e viceversa. Se tutte le variabili sono globali, non puoi mai cambiare il modo in cui una variabile viene utilizzata senza il rischio di creare un bug in qualche altra parte del programma. Le funzioni in PHP aiutano a proteggere le variabili rendendole locali alla funzione. Le classi estendono ulteriormente questo concetto introducendo variabili che appartengono a un oggetto in modo che possano essere utilizzate in più metodi senza essere globali. Il nome di fantasia per questo è incapsulamento.

Proviamo un semplice esempio. È difficile trovare esempi orientati agli oggetti che siano semplici e realistici. Quindi creiamo uno scenario: è necessario creare un'applicazione web che produca "Hello world!". Creiamo una classe che genera codice HTML per un messaggio "Hello World".

```
class HelloWorld {  
public $world = 'World';  
function getHtml() {  
return "<html><body>".  
"Hello, ".$this->world."!".  
"</body></html>";  
}  
}
```

Per usare questa classe, dovresti fare qualcosa del genere:

```
$greetings = new HelloWorld; // Create the object  
echo $greetings->getHtml(); // Display the greeting message
```

La classe genera il documento HTML concatenando stringhe costanti e inserendo il nome che ha memorizzato nella variabile. Questa variabile è chiamata variabile di istanza. Una variabile di istanza appartiene all'oggetto ed è disponibile come `$this-> world` in qualsiasi metodo all'interno della classe. La parola chiave `public` dichiara la variabile e la rende disponibile a livello globale. L'uso di variabili pubbliche non è necessariamente una buona pratica in PHP, ma mantiene le cose semplici finché stiamo sperimentando. Per utilizzare la variabile di istanza nel metodo `getHtml()`, fai riferimento ad essa come `$this->world`. Questa variabile viene utilizzata solo in un metodo, ma le variabili di istanza diventano davvero utili solo quando vengono utilizzate in più di un metodo.

Costruttori: creazione e inizializzazione

L'applicazione "hello world" è un successo clamoroso ma sfortunatamente, un paio di giorni dopo, il tuo capo ti dice che il programma non è compatibile con il motto dell'azienda, "eccellenza universale". Bene, allora dovrai solo rendere possibile specificare il nome del pianeta durante la creazione dell'oggetto:

```
$greetings = new HelloWorld('Epsilon Eridani II');  
echo $greetings->getHtml();
```

La parola chiave `new` crea una nuova istanza della classe. Inoltre, esegue un metodo chiamato costruttore che possiamo usare per inizializzare e configurare l'oggetto. In PHP, i costruttori sono chiamati `construct()`. Quindi ora possiamo usare il costruttore per impostare il nome utente:

```
class HelloWorld {  
    public $world;  
    function __construct($world) {  
        $this->world = $world;  
    }  
    function getHtml() {  
        return "<html><body>".  
            "Hello ".$this->world."!".  
            "</body></html>";  
    }  
}
```

Le variabili di istanza consentono a metodi diversi di condividere variabili anche se non sono globali. Quindi, se hai un'applicazione PHP legacy che utilizza liberamente le variabili globali, un trucco utile è trasformarle in variabili di istanza in una classe. Un sistema BBS potrebbe avere una funzione `display_messages()` con la seguente dichiarazione di variabile globale:

```
global $db, $strings, $mode;
```

`$db` è un oggetto che rappresenta la connessione al database, `$strings` è una raccolta di stringhe indipendenti dal linguaggio e `$mode` è una modalità di visualizzazione (con o senza thread). Facciamo finta di voler eseguire il refactoring di questa applicazione. Una possibilità è invece farli appartenere a una classe, ad esempio `MessageView`. Quindi le variabili sarebbero invece variabili di istanza:

```
class MessageView {  
    public $db;  
    public $strings;  
    public $mode;  
    function __construct($db, $strings, $mode) {  
        $this->db = $db;  
        $this->strings = $strings;  
        $this->mode = $mode;  
    }  
    function display_messages() {  
        $result = $this->db->query('SELECT * FROM messages');  
        //etc...  
    }  
}
```

Questo esempio illustra anche come una variabile di istanza può contenere un altro oggetto, in questo caso un oggetto che rappresenta la connessione al database. In PHP, questo significa che l'oggetto contiene un riferimento all'altro oggetto. I riferimenti agli oggetti verranno spiegati più avanti in questo capitolo. La classe `MessageView` ha tre variabili di istanza. `$strings` e `$mode` sono rappresentati come attributi. Poiché `$db` è un oggetto, e probabilmente un po' complesso, può essere inteso come una classe separata.

Ereditarietà

Concettualmente, l'ereditarietà nella programmazione object-oriented è un modo per esprimere relazioni tra categorie. Tecnicamente, l'ereditarietà è un modo in cui una classe può ottenere tutte o alcune delle caratteristiche di un'altra classe in modo facile. L'alternativa è creare un'istanza dell'altra classe e usarla, ma comporta molto più lavoro da fare. Una classe eredita le caratteristiche di un'altra classe semplicemente aggiungendo la parola chiave `extends` alla dichiarazione di classe. Vediamo cosa succede se creiamo una classe vuota che ne estende un'altra:

```
class NewsArticle extends Document {  
}
```

Esiste ora quella che viene chiamata relazione genitore-figlio tra le due classi. La classe `NewsArticle` è la classe figlia; la classe `Document` è il genitore. Il risultato pratico di ciò che abbiamo fatto è che la classe `NewsArticle` funziona esattamente come la classe `Document`. Se avessimo copiato e incollato l'intera classe `Document` e ne avessimo cambiato il nome, ciò avrebbe avuto lo stesso effetto. La differenza non sta nel modo in cui funziona il codice, ma nel fatto che non dobbiamo duplicare il codice. Stiamo riutilizzando la classe `Document`, ed è una buona idea.

Questo va bene, tranne per il fatto ovvio che di solito non abbiamo bisogno di due classi che funzionino esattamente allo stesso modo. Ha più senso se la classe figlia contiene qualche implementazione. Possiamo aggiungere nuovi metodi e dati o sovrascrivere i metodi. Ad esempio, la classe `NewsArticle` può avere variabili, ad esempio `$newsSource` o `$byline`, che non sono presenti nella classe `Document`. Continuiamo con il nostro esempio precedente. Ancora una volta, questo non è molto realistico, ma facciamo finta di volere una classe più generale: una che possa rappresentare qualsiasi documento HTML, non solo quelli contenenti messaggi di benvenuto. Quindi iniziamo inserendo il documento HTML di base:

```
class HtmlDocument {  
  function getHtml() {
```



```

return "<html><body>".$this->getContent().
"</body></html>";
}
function getContent() { return ""; }
}

```

Il metodo getHtml() inserisce il risultato del metodo getContent() tra i tag di inizio e di fine per il documento HTML e restituisce il risultato. La funzione getContent() è abbastanza inutile, poiché restituisce una stringa vuota, ma HtmlDocument è solo la nostra classe genitore. Possiamo aggiungere una classe figlio che fa qualcosa di più simile a quello che abbiamo fatto prima:

```

class HelloWorld extends HtmlDocument {
public $world;
function __construct($world ) {
$this->name = $world ;
}
function getContent() {
return "Hello, ".$this->world."!";
}
}

```

Il metodo getContent() nella classe HelloWorld ora sovrascrive il metodo getContent() nella sua classe padre. E il metodo getHtml() funziona come se lo avessimo copiato dalla classe HtmlDocument alla classe HelloWorld. Quindi questa classe fa lo stesso lavoro della nostra precedente classe HelloWorld, ma il metodo getHtml() è ora nella classe genitore. Ciò significa che possiamo creare un'altra classe che estenda la classe HtmlDocument e metta qualcos'altro all'interno del documento. L'eredità non è solo un privilegio dei metodi ordinari, anche i costruttori possono trarne vantaggio. Parte del lavoro necessario alla costruzione di un oggetto può essere comune a classi simili e altre possono essere diverse. In PHP è stato introdotto un nuovo stile di costruttore che lo rende più semplice.

Invece di usare un costruttore con lo stesso nome della classe, possiamo usare il nome del metodo speciale `__construct()`:

```
class Document {  
protected $title;  
protected $text;  
function __construct($title,$text) {  
$this->title = $title;  
$this->text = $text;  
}  
}
```

Ciò rende leggermente più semplice ereditare il comportamento del costruttore rispetto ai costruttori vecchio stile:

```
class NewsArticle extends Document{  
private $introduction;  
function __construct($title,$text,$introduction) {  
parent::__construct($title,$text);  
$this->introduction = $introduction;  
}  
}
```

L'istruzione `parent::__construct` chiama il costruttore dalla classe `Document`. Un'altra domanda è quanto sia utile ereditare il comportamento del costruttore. Si potrebbe incorrere nel rischio di rendere più difficile il refactoring. In questo esempio, potrebbe essere meglio spostare tutta la costruzione nella classe figlia e duplicarla nelle altre classi figlio per rendere il codice più leggibile e più facile da modificare. È improbabile che quella piccola quantità di duplicazione danneggi la leggibilità o manutenibilità del tuo codice. Finora abbiamo studiato come funzionano gli oggetti in circostanze normali. In precedenza, abbiamo notato come una classe sia come una casa con stanze. Ora vogliamo sapere cosa fare in caso di incendio. Vogliamo poter uscire velocemente ma in sicurezza. Per renderlo possibile, i linguaggi orientati agli oggetti (incluso PHP a partire dalla versione 5) utilizzano una funzionalità chiamata eccezioni.

ECCEZIONI

Il modo più semplice per gestire un errore in PHP 4 era usare il metodo `die()` in caso di errore. Martin Fowler lo definisce "l'equivalente software del suicidio nel caso in cui tu avessi perso un volo", ma aggiunge che "se il costo di un crash del programma è basso e l'utente è tollerante, interrompere un programma va bene". In PHP, come in molti altri linguaggi, abbiamo un'alternativa al suicidio: lanciare un'eccezione. Questo è l'equivalente software di lanciarsi da una finestra di una casa o di un edificio e sperare che ci siano i pompieri o qualcuno con un materassino prima di toccare terra. Se non gestiamo l'eccezione utilizzando `catch` a un certo punto del codice che chiama questa classe (direttamente o indirettamente), il programma si fermerà e stamperà un messaggio con uno stack trace di chiamate. Quindi l'effetto immediato è approssimativamente lo stesso di `die()`, ma se decidiamo in seguito di voler gestire l'errore, possiamo farlo più facilmente. I meccanismi di utilizzo delle eccezioni sono una cosa; usarli con saggezza e giudizio è decisamente un'altra cosa. Questa sezione non mostrerà tutti i dettagli delle eccezioni; piuttosto si concentrerà sul mostrare modi ragionevoli per utilizzare le eccezioni e sugli aspetti della gestione delle eccezioni di PHP che sono più utili per supportarlo. Maggiori dettagli sugli aspetti tecnici delle eccezioni PHP sono disponibili nel manuale PHP online. Per un approfondimento su come utilizzare le eccezioni, dai un'occhiata al capitolo dedicato nel libro *Object Design* di Rebecca Wirfs-Brock e Alan McKean. In questa sezione, inizieremo scoprendo come funzionano le eccezioni. Quindi considereremo come e quando è opportuno utilizzare le eccezioni e quando potrebbe essere meglio utilizzare dei `return` in vecchio stile. Vedremo come creare le nostre classi di eccezioni e faremo del nostro meglio per scoprire come sostituire gli errori PHP integrati con le eccezioni. Infine, vedremo come evitare un uso eccessivo delle eccezioni.

Il costrutto del linguaggio di programmazione noto come eccezione è un modo per comunicare condizioni di errore o di eccezione tra diverse parti del programma. Ad esempio, potremmo avere il nome del database utilizzato dall'applicazione in una variabile di ambiente denominata `DB_NAME`. Senza la gestione delle eccezioni, potremmo consentire a un metodo di recuperare quel nome come segue:

```

public function getDatabaseName() {
    if (!array_key_exists('DB_NAME',$_ENV))
        die("Environment variable DB_NAME is not set");
    return $_ENV['DB_NAME'];
}

```

Questa, quindi, è la versione suicida, che usa die() invece di exit per rendere chiaro il parallelo, sebbene i due abbiano esattamente lo stesso effetto. Ma usare un'eccezione invece è davvero molto semplice:

```

public function getDatabaseName() {
    if (!array_key_exists('DB_NAME',$_ENV))
        throw new Exception(
            "Environment variable DB_NAME is not set");
    return $_ENV['DB_NAME'];
}

```

Se non facciamo nulla per catturare l'eccezione, questo ha lo stesso effetto di die(): interrompe l'applicazione. Tuttavia, fa una cosa aggiuntiva: stampa uno stack trace che può essere utile per il debug. È possibile ottenere uno stack trace senza utilizzare eccezioni grazie alle funzioni debug_backtrace() e debug_print_backtrace(). Ma lanciare un'eccezione è un modo ancora più semplice.

```

Fatal error: Uncaught exception 'Exception' with message
'Environment variable DB_NAME is not set' in
/path/exception.php:6

```

Stack trace:

```

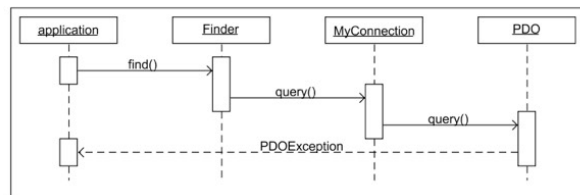
#0 /path/exception.php(6): Config::getDatabaseName()
#1 /path/exception.php(12): Config->getDatabaseName()
#2 {main}
thrown in /path/exception.php on line 6

```

Se non vogliamo che gli utenti vedano il rapporto di errore tecnico (e in generale non lo facciamo per motivi di sicurezza), siamo più sicuri avendo

usato le eccezioni piuttosto che die(). Se abbiamo usato die() in più punti, potremmo dover trovare tutte le occorrenze e sostituirle una per una. Se abbiamo utilizzato le eccezioni, tutto ciò che dobbiamo fare è catturarle in un unico posto, ad esempio il livello più alto dell'applicazione. Potremmo registrare il messaggio e reindirizzare l'utente a una pagina che dice semplicemente che si è verificato un errore:

```
$config = new Config;  
try {  
    $config->getDatabaseName();  
}  
catch(Exception $e) {  
    $logger->log($e->getMessage());  
    header("Location: unrecoverable.php");  
}
```



L'immagine è un diagramma di sequenza UML che mostra come funzionano le eccezioni. I dettagli del diagramma, nomi di classi e metodi, non sono importanti. L'essenza è questa: quando l'oggetto PDO genera un'eccezione, invece di tornare a MyConnection, bypassa sia MyConnection che Finder, poiché nessuno di questi ha blocchi try e catch. Ma l'applicazione cattura l'eccezione e la gestisce. Se si ruota il diagramma di 90 gradi in senso antiorario, sarà orientato in modo da corrispondere all'analogia con l'edificio: il flusso di messaggi sale le scale, supera i piani uno per uno, salta fuori dalla finestra e viene catturato nella rete al piano terra: dall'applicazione. C'è un importante vantaggio che le eccezioni condividono con die(): interrompono ulteriori elaborazioni. Il motivo per cui ciò è utile è che di solito, quando viene generata un'eccezione, il resto di ciò che accade nel metodo corrente è privo di significato. Un modesto esempio è il metodo getDatabaseName() che abbiamo appena visto: restituire il nome del database dalla funzione è inutile poiché non c'è un

nome del database da restituire. Nessuno dei programmi PHP viene eseguito tra il momento in cui viene generata l'eccezione e il momento in cui viene rilevata l'eccezione.

Il significato sintattico di una parola chiave del linguaggio di programmazione è spesso diverso dal significato concettuale della parola. Ciò vale anche in caso di eccezioni. La parola "eccezione" significa qualcosa che accade raramente. Nella progettazione del software, c'è una distinzione tra errori ed eccezioni. Un errore è in genere qualcosa che è fatale o paralizzante per la capacità del programma di svolgere il proprio lavoro; un'eccezione è una situazione non comune, ma recuperabile. In pratica, le eccezioni (in senso sintattico) sono utilissime per gestire errori come quello che abbiamo visto. Se il nome del database non è disponibile e l'applicazione dipende totalmente da esso, la possibilità di eccezioni per impedire un'ulteriore elaborazione a quel punto è appropriata e utile. Il tentativo di eseguire query SQL con un database inesistente e il tentativo di elaborare dati inesistenti possono solo generare ulteriori errori potenzialmente confusi. Il codice di gestione degli errori può essere controproducente: se ne crei troppo, renderà il programma meno leggibile e renderà più difficile individuare i bug. Se abbiamo una buona copertura in termini di unit test, la gestione degli errori è per lo più necessaria ai limiti della nostra applicazione: le sue interfacce con altri sistemi e il resto del mondo. Anche se il tuo software è popolato esclusivamente da oggetti ben addestrati, devi pattugliare i confini. Un confine è rappresentato dalle risorse fornite dal sistema operativo, dai file, dalla rete e dai database. Potresti già saperlo, ma riassumiamo alcuni errori tipici in un'applicazione Web PHP:

- Errori da informazioni di configurazione errate come la password necessaria per connettersi a un database;
- Errori di sintassi SQL o XML;
- File importanti che non possono essere letti.

Ma potremmo anche aver bisogno di pattugliare l'altro confine: quello di fronte ad un altro software che sta usando il nostro programma. I controlli di sicurezza e la convalida dell'input dell'utente si verificano in genere ai livelli più alti dell'applicazione; è più probabile che la risposta sia un messaggio diretto all'utente piuttosto che un'eccezione. Ma cosa succede

se stiamo creando un software di libreria di basso livello che viene utilizzato da altri? Controllare gli input e assicurarsi che non vi siano valori assurdi può far risparmiare molto tempo in fase di debug.

Un possibile esempio è un pacchetto che fornisce informazioni statistiche sui dati memorizzati in un database. In genere, i client di questa libreria dovranno fornire l'ora di inizio e di fine dell'intervallo di tempo per le statistiche. Cosa succede se l'ora di inizio, o l'ora di fine, o entrambe, sono NULL, 0 o qualche altro valore inappropriato? Se sono entrambi zero e li interpretiamo come 1 gennaio 1970, molto probabilmente restituiremo un set di dati vuoto. E scoprire perché il set di dati è vuoto potrebbe richiedere anche ore di tempo.

Partendo dal presupposto che siamo responsabili solo del generatore di statistiche, vogliamo assicurarci che abbia una copertura di test; perciò, controlliamo la presenza di errori e input non validi nelle interfacce. Questo per quanto riguarda gli errori. Per le eccezioni in senso concettuale, per situazioni rare ma recuperabili, può essere spesso più utile utilizzare la logica condizionale ordinaria e restituire codici per ripristinare immediatamente l'errore, piuttosto che generare un'eccezione. Ci sono diversi modi per farlo. Il modo migliore è di solito che la classe chiamante chieda alla classe che sta chiamando informazioni che le consentiranno di decidere se esiste una situazione eccezionale in primo luogo. Un altro è restituire un codice di errore. Un altro ancora è semplicemente ignorare il problema. Ad esempio, se manca uno di un insieme di file da elaborare, l'utente finale potrebbe preferire un risultato incompleto ad un messaggio di errore.

Creare un'eccezione

La classe `Exception` è incorporata in PHP stesso ed è sempre disponibile. Creando classi figlie che estendono la classe `Exception`, possiamo definire i nostri tipi di eccezioni. Può essere semplice come questo:

```
class ConfigurationException extends Exception {}
```

Come abbiamo visto prima, una classe che estende un'altra classe ma non contiene alcuna implementazione funziona esattamente come la classe originale. Allora perché dovremmo volerlo fare? Perché le eccezioni sono leggermente diverse dalle classi ordinarie. Per distinguere diversi tipi di eccezioni, è consuetudine utilizzare classi diverse. E poiché la clausola `catch` ti consente di specificare la classe di eccezione, puoi usarla per catturare diverse eccezioni in luoghi diversi. Ad esempio, se si utilizza una classe di eccezione per la mancata connessione a un database e un'altra per errori di sintassi SQL, è possibile che le due vengano intercettate in posizioni diverse nel codice. D'altra parte, è una cattiva idea creare grandi gerarchie di classi di eccezioni.

WirfsBrock e McKean consigliano un massimo di cinque o sette classi di eccezione diverse per il semplice motivo che è difficile ricordare troppe classi. È invece possibile utilizzare i codici di errore per distinguere i diversi sottotipi all'interno di una classe di eccezione. Un modo leggibile e sicuro per farlo è utilizzare la capacità della classe `Exception` di memorizzare il codice di errore insieme alle costanti di classe. Se vogliamo una `ConfigException` con la possibilità di segnalare sia gli errori di connessione al database che gli errori di sintassi SQL, possiamo definire la classe `Exception` come segue:

```
class ConfigException extends Exception {  
    const SQL_SYNTAX_ERROR = 1;  
    const DB_CONNECTION_ERROR = 2;  
}
```

Quando lanciamo l'eccezione, possiamo specificare sia il messaggio di errore che il codice di errore, poiché questi sono accettati dal costruttore per la classe Exception.

```
throw new ConfigException(  
    "Could not connect to database $dbname",  
    ConfigException::DB_CONNECTION_ERROR);
```

E quando catturiamo l'eccezione, possiamo testare il codice di errore e agire di conseguenza:

```
catch(ConfigException $e) {  
    switch ($e->getCode()) {  
        case ConfigException::DB_CONNECTION_ERROR:  
            echo "Connection error\n";  
            break;  
        case ConfigException::SQL_SYNTAX_ERROR:  
            echo "SQL error\n";  
            break;  
    }  
}
```

Nella vita reale, ovviamente, faremmo qualcosa di più sofisticato della semplice stampa a video della stringa. Ma cosa succede se vogliamo gestire solo uno dei nostri sottotipi di eccezione qui e gestire l'altro tipo da qualche altra parte? È semplice: possiamo rilanciarlo in modo che possa essere catturato con un metodo o un oggetto diverso:

```
case ConfigException::SQL_SYNTAX_ERROR:  
    throw $e;  
    break;
```

È una buona idea denominare le classi di eccezione in base a ciò che è andato storto invece di usare un nome con riferimento a dove si è verificato il problema. La classe ConfigException negli esempi precedenti ha lo scopo

di trasmettere l'idea che si tratta di eccezioni che sono in genere causate da errori di configurazione o bug nell'applicazione.

Dopo aver utilizzato le eccezioni, è un po' irritante che gli errori di PHP vengano segnalati come errori in stile PHP 4 anziché come eccezioni, questo soprattutto se stai effettuando un porting di una vecchia applicazione. Per fortuna è possibile costruire un “ponte” dal vecchio sistema di gestione degli errori al nuovo. Sebbene ciò non rilevi tutti gli errori (gli errori irreversibili di runtime come la chiamata di un metodo inesistente su un oggetto non verrà segnalata), renderà più coerente la gestione degli errori. Le prime cose di cui abbiamo bisogno sono una classe di eccezione per distinguere gli errori PHP da altre eccezioni e un semplice gestore di errori per ricevere un errore PHP e lanciare invece un'eccezione:

```
class ErrorFromPHPException extends Exception {}  
function PHPErrorHandler($errno, $errstr, $errfile, $errline) {  
    throw new ErrorFromPHPException($errstr,$errno);  
}
```

Ora possiamo impostare il gestore degli errori. Se procediamo a provare ad aprire un file inesistente, otterremo un'eccezione invece dell'errore vecchio stile:

```
$oldHandler = set_error_handler('PHPErrorHandler');  
fopen('/tmp/non-existent','r');
```

E se per qualche motivo vogliamo tornare al modo ordinario di gestire questi errori, possiamo farlo:

```
set_error_handler($oldHandler);
```

Vogliamo evitare di ingombrare il nostro codice con troppa gestione degli errori e le eccezioni ci aiutano a farlo, poiché le istruzioni catch possono essere inferiori alle condizioni di gestione degli errori che devono testare i codici di ritorno da ogni chiamata al metodo. Ma anche con le eccezioni, non c'è motivo di controllare ogni possibile problema. Come dicono Wirfs-Brock e McKean:

Le collaborazioni difensive, la progettazione di oggetti per prendere precauzioni prima e dopo aver chiamato un collaboratore, sono costose e soggette a errori. Non tutti gli oggetti dovrebbero essere incaricati di queste responsabilità.

Fortunatamente, PHP non ti obbliga mai a controllare nulla. La gestione delle eccezioni è una delle funzionalità più importanti introdotte in PHP. Un cambiamento ancora più importante è stato il nuovo modo di gestire i riferimenti agli oggetti.

RIFERIMENTO AGLI OGGETTI

Quando la polizia è alla ricerca di un criminale ricercato o di una persona scomparsa, è utile avere una fotografia dell'individuo. Una buona fotografia può facilitare il riconoscimento di una persona, ma mostra solo un particolare istante. Le persone cambiano i vestiti, si truccano, si tagliano i capelli, si radono, si fanno crescere la barba, mettono gli occhiali da sole e si sottopongono persino a interventi di chirurgia plastica. Prima o poi diventa difficile riconoscere la persona dalla fotografia. Ancora più ovvio e fondamentale è il fatto che fare qualcosa alla fotografia non influirà sulla persona. Mettere l'immagine in una cella di prigione è inutile. A meno che tu non creda nei riti voodoo, devi convivere con il fatto che l'immagine e la persona sono fisicamente entità separate. Quindi ci sono dei limiti a ciò che puoi fare se hai solo la fotografia a tua disposizione. Non è mai come avere la persona presente.

La gestione degli oggetti PHP è simile. Prima di PHP 5 veniva creata una copia di un oggetto ogni volta che si eseguiva un task o veniva restituito un oggetto da una funzione o da un metodo. Quindi si otteneva un'istantanea ingannevolmente simile all'originale, ma in realtà era un oggetto diverso e non rifletteva o causava modifiche nell'originale. Questo crea alcuni problemi, simili all'analogia della fotografia. Nella programmazione orientata agli oggetti, un oggetto rappresenta tipicamente un'entità, reale o astratta, che non può essere semplicemente modificata tramite proxy. La modifica di una copia di un documento non sarà di aiuto se l'originale è quello salvato nel database. Modificare un oggetto che rappresenta il titolo di una pagina HTML non aiuta se l'originale è quello mostrato nel browser. Ma a differenza di una fotografia, una copia di un oggetto ha tutta la massa e il peso dell'originale. Se l'oggetto originale contiene due megabyte di dati, anche la copia contiene due megabyte di dati, quindi ora hai quattro megabyte in tutto. Quindi la copia di oggetti fa consumare al programma più memoria del necessario.

Ecco perché la gestione degli oggetti in questo stile in PHP è universalmente riconosciuta come un aspetto negativo. Sembrava una buona idea nel momento in cui è stata implementata ma, in realtà, non lo era. Le persone che hanno sviluppato PHP non desideravano appassionatamente quel tipo di comportamento degli oggetti. È solo stato

più facile dato il modo in cui PHP era stato implementato. L'orientamento agli oggetti non era molto utilizzato in PHP in passato, ma non appena è stato rivalutato, la programmazione orientata agli oggetti in PHP è diventata piuttosto popolare. Alla fine, è diventato ovvio che il modo in cui PHP gestisce gli oggetti era una responsabilità, quindi è diventata una priorità urgente modificare il comportamento predefinito degli oggetti in PHP per utilizzare i riferimenti anziché le copie.

Questo è iniziato con PHP 5 e si è propagato nelle versioni successive. L'orientamento agli oggetti in PHP ora funziona allo stesso modo della maggior parte degli altri linguaggi orientati agli oggetti. Anche PHP 4 ha riferimenti, ma sono diversi dai riferimenti agli oggetti nella maggior parte dei linguaggi orientati agli oggetti. Possono essere usati, e sono stati usati, per la programmazione orientata agli oggetti in PHP 4 ma è difficile capire come funzionano e a volte fanno cose che potresti non aspettarti che facciano. Il loro comportamento è controintuitivo.

Gli oggetti PHP (a partire dalla versione 5), si comportano per la maggior parte del tempo in modo utile e naturale. La variabile contiene un riferimento all'oggetto e solo il riferimento viene copiato. Come accennato, i riferimenti agli oggetti aiutano a migliorare le prestazioni impedendo la copia degli oggetti e il consumo eccessivo di spazio di memoria. Nelle applicazioni PHP 4, sono stati fatti molti sforzi per evitare questo sovraccarico copiando esplicitamente gli oggetti per riferimento. Questo ha senso se hai molti oggetti o se sono molto grandi. Prova a scaricare un oggetto PEAR DB e vedrai cosa intendo per oggetti di grandi dimensioni. D'altra parte, se mantieni il tuo design semplice, ti aiuterà anche a mantenere i tuoi oggetti più piccoli.

In PHP 5, questi sforzi non sono più necessari, avere oggetti rappresentati da riferimenti ha anche vantaggi per la progettazione orientata agli oggetti. Semplifica la costruzione e la manipolazione di strutture di oggetti complesse. Supponiamo un oggetto \$dog all'interno dell'oggetto \$doghouse, quindi modifichi l'oggetto \$dog e vuoi che si rifletta all'interno di \$doghouse. Le GUI in genere hanno questo tipo di struttura complessa. Nella programmazione web, lavoriamo con documenti HTML, ma supponiamo di rappresentare gli elementi in un documento HTML come oggetti. Potremmo fare qualcosa del genere:

```
$checkbox = new Checkbox;  
$form = new Form;
```

```
$document = new Document;  
$document->add($form);  
$form->add($checkbox);
```

Ora cosa succede se cambiamo uno degli elementi interni?

In PHP 5 cambierà e quando genereremo il codice HTML dall'oggetto Document, avrà una casella di controllo selezionata. Questo è ovviamente ciò che vogliamo e illustra cosa intendo quando dico che il comportamento degli oggetti PHP è per lo più intuitivo, utile e naturale. I riferimenti agli oggetti possono essere meravigliosamente intuitivi per la maggior parte del tempo, ma altre volte vogliamo attivamente che gli oggetti vengano copiati piuttosto che passati per riferimento. Questo è il caso dei tipi di oggetti noti come oggetti valore (value objects). Se rappresentiamo date, somme di denaro e simili come oggetti, sarà più naturale copiarli, perché non hanno identità. Per copiare oggetti in PHP, usa la parola chiave clone.

METODI E CLASSI

In PHP esiste una funzionalità denominata chiamate di metodi sovraccaricabili (overloadable method calls). In pratica, la funzione ci consente di intercettare, reindirizzare e ridefinire le chiamate di un metodo. È come rubare la posta di qualcuno e aprirla. Quindi possiamo inviarla a qualcun altro, cambiarne il contenuto o addirittura gettarla nel cestino. Ciò significa che possiamo cambiare il solito modo in cui i metodi rispondono e persino rispondere a metodi inesistenti. Inizieremo questo capitolo chiarendo il nome di questa funzionalità e come si collega all'idea di intercettare le chiamate di metodo. Quindi vedremo un paio di esempi di come questo può essere utilizzato: sovraccarico del metodo in stile Java e un meccanismo di registrazione generale per le chiamate ai metodi. Infine, daremo un'occhiata a un argomento correlato: come utilizzare la funzione di overloading per controllare cosa succede quando viene istanziata una classe. "Sovraccarico del metodo" può essere un termine leggermente strano, poiché significa qualcosa di specifico in altri linguaggi. In Java e C++, l'overloading dei metodi significa scrivere metodi diversi che hanno lo stesso nome, ma numeri o tipi di argomenti diversi, il metodo che viene eseguito dipende dagli argomenti forniti. Ciò è particolarmente utile nei linguaggi tipizzati staticamente (come Java e C++). Senza l'overloading del metodo, potrebbero essere necessari due metodi con nomi diversi solo per gestire argomenti di tipo diverso (ad esempio, una data specificata come stringa o un timestamp numerico). In PHP si tratta di qualcosa di più generale. Puoi sovraccaricare le chiamate al metodo, ma devi definire tu stesso l'overloading. Funziona così: se provi a chiamare un metodo che non è definito, PHP chiamerà invece un metodo chiamato `__call()`. Quindi puoi fare quello che vuoi con la chiamata al metodo "fallito". Puoi eseguire un altro metodo, possibilmente su un altro oggetto, oppure puoi dare un messaggio di errore diverso dal solito. Non puoi nemmeno fare nulla; ciò farà sì che PHP ignori le chiamate al metodo non riuscite invece di generare un errore irreversibile.

Potrebbe essere utile occasionalmente, ma in generale, fai attenzione a tutto ciò che riduce il livello di controllo degli errori e consente ai bug di passare inosservati. Questo comportamento non è l'overloading del metodo, ma consente di definire l'overloading del metodo; quindi, rende le chiamate

al metodo sovraccaricabili. Il termine overloading significa che lo stesso elemento (in questo caso, il nome di un metodo) può avere significati diversi a seconda del contesto. E, poiché `__call()` ci consente di controllare il contesto e rispondere in base ad esso, l'overloading del metodo è una delle cose che possiamo fare con esso. A volte è conveniente poter chiamare lo stesso metodo con un numero variabile di argomenti. PHP lo rende possibile grazie alla sua capacità di definire argomenti opzionali con valori predefiniti. Tuttavia, a volte, è necessario che il metodo abbia comportamenti significativamente diversi a seconda dell'elenco di argomenti. Nei linguaggi che non hanno l'overloading del metodo, ciò significa aggiungere la logica condizionale all'inizio del metodo. Se invece puoi utilizzare l'overloading del metodo, puoi saltare la logica condizionale e il tuo codice sarà più pulito ed elegante. È possibile implementare questo tipo di sovraccarico del metodo utilizzando `__call()` in PHP 5. Diamo un'occhiata a un esempio. Assumiamo che riutilizzeremo il comportamento di sovraccarico, quindi mettiamolo in una classe padre astratta:

```
abstract class OverloadableObject {  
function __call($name,$args) {  
    $method = $name.".".count($args);  
    if (!method_exists($this,$method)) {  
        throw new Exception("Call to undefined method "  
        get_class($this). ":: $method");  
    }  
    return call_user_func_array(array($this,$method),$args);  
    }  
}
```

La maggior parte del comportamento di questa classe è definita da una sola riga. Se viene chiamato un metodo non definito, il metodo `__call()` genera un nuovo nome per il metodo costituito dal metodo originale e dal numero di argomenti, separati da un carattere di sottolineatura. Quindi chiama il metodo con il nome appena generato, passando gli argomenti originali. Ora, se vogliamo creare un metodo sovraccarico chiamato `multiply` che può essere chiamato con uno o due argomenti e li moltiplicherà in entrambi i casi, creiamo due metodi chiamati rispettivamente `multiply_2` e `multiply_3`:

```
class Multiplier extends OverloadableObject {
```

```
function multiply_2($one,$two) {  
    return $one * $two;  
}  
function multiply_3($one,$two,$three) {  
    return $one * $two * $three;  
}  
}
```

Per usarlo, chiamiamo semplicemente il metodo con due o tre argomenti:

```
$multi = new Multiplier;  
echo $multi->multiply(5,6)."\n";  
echo $multi->multiply(5,6,3)."\n";
```

Questo non è ancora esattamente lo stesso dell'overloading del metodo in Java e C++, poiché stiamo solo controllando il numero di argomenti, non i loro tipi. Tuttavia, potremmo usare anche le informazioni sul tipo. D'altra parte, come abbiamo visto, fare in modo che il comportamento dipenda dai tipi di argomento è meno importante in PHP che nei linguaggi tipizzati staticamente. Abbiamo esaminato il funzionamento delle chiamate di metodo sovraccaricabili. Per un esempio di come possono essere utilizzati, vediamo come possono essere utilizzati per loggare le chiamate ai metodi.

La programmazione orientata agli aspetti (aspect-oriented programming) è un modo relativamente nuovo di fare alcune cose che non sono del tutto eleganti nella semplice programmazione orientata agli oggetti. Si consideri, ad esempio, il problema di loggare l'inizio e la fine di tutte le chiamate di metodo in un'applicazione. Per fare ciò in OOP, dobbiamo aggiungere codice a ogni singolo metodo. Possiamo lavorare per rendere minimo questo codice aggiuntivo, ma sicuramente aggiungerà un disordine sostanziale alle nostre classi. Il logging è in genere il tipo di problema affrontato dalla programmazione orientata agli aspetti. Questi problemi, chiamati problemi trasversali, toccano moduli o sottosistemi diversi e sono difficili da isolare in classi separate. Un altro esempio potrebbe essere il controllo se l'utente corrente è autorizzato o meno all'utilizzo di un metodo. La programmazione orientata agli aspetti viene in

genere eseguita definendo aspetti, costrutti di classe che vengono inseriti nel codice durante un processo di generazione del codice. Qui faremo qualcosa di molto più semplice e primitivo usando `__call()` in PHP. Usiamo la classe PEAR Log e controlliamo il processo di logging dal metodo `__call()` in una classe genitore, come mostrato:

```
class LoggingClass {  
    function __call($method,$args) {  
        $method = "_$method";  
        if (!method_exists($this,$method))  
            throw new Exception("Call to undefined method "  
                .get_class($this). ":: $method");  
        $log = Log::singleton('file', '/tmp/user.log',  
            'Methods', NULL, LOG_INFO);  
        $log->log(" Just starting method $method");  
        $return = call_user_func_array(array($this,$method),$args);  
        $log->log(" Just finished method $method");  
        return $return;  
    }  
}
```

Questo è simile al nostro esempio di sovraccarico del metodo, in quanto il metodo effettivo ha un nome leggermente diverso dal nome che chiamiamo dal codice client. Il metodo che chiamiamo dal codice client non esiste, quindi `__call()` lo intercetta, registra l'inizio, chiama il metodo reale e registra la fine. Per usarlo, dobbiamo estendere `LoggingClass` e dare ai metodi nomi che iniziano con un trattino basso. Non c'è alcun motivo per cui debba essere un trattino basso; puoi usare qualsiasi cosa che renda unici i nomi. Ecco una classe semplificata per la gestione di date e orari:

```
class DateAndTime extends LoggingClass {  
    private $timestamp;  
    function __construct($timestamp=FALSE) {  
        $this->init($timestamp);  
    }  
    protected function _init($timestamp) {  
        $this->timestamp = $timestamp ? $timestamp : time();  
    }  
}
```

```

function getTimestamp() { return $this->timestamp; }
protected function _before(DateAndTime $other) {
return $this->timestamp < $other->getTimestamp();
}
}

```

I metodi `init()` e `before()` verranno loggati; il metodo `getTimestamp()` non verrà loggato, poiché il nome non inizia con un carattere di sottolineatura. Ho aggiunto il metodo `init()` per consentire anche il log della costruzione dell'oggetto. Il metodo `__call()` normalmente non viene attivato durante la costruzione. Non è sorprendente, dal momento che una classe non deve avere un costruttore. Avrai notato che i metodi sono dichiarati protetti, ciò significa che non possono essere chiamati dal codice client se non tramite il meccanismo `__call()`. Sono protetti anziché privati perché il metodo `__call()` è in una classe padre. Ora proviamo la classe e vediamo cosa succede. Creiamo due diversi oggetti `DateAndTime` e poi li confrontiamo:

```

$now = new DateAndTime;
$nexthour = new DateAndTime(time() + 3600);
print_r(array($now,$nexthour));
if ( $now->before($nexthour) ) {
echo "OK\n";
}

```

Le chiamate al metodo vengono loggate in questo modo:

```

04 15:20:08 Methods [info] Just starting method _init
July 04 15:20:08 Methods [info] Just finished method _init
July 04 15:20:08 Methods [info] Just starting method _init
July 04 15:20:08 Methods [info] Just finished method _init
July 04 15:20:08 Methods [info] Just starting method _before
July

```

È lontano dalla programmazione orientata agli aspetti (AOP) in un linguaggio AOP specializzato. E in pratica, se vuoi loggare le chiamate ai

metodi, potresti cercare uno strumento di profilazione. Tuttavia, sembra esserci un potenziale per applicazioni utili. Le chiamate di metodo sovraccaricabili sono una sorta di magia che ci consente di definire cosa accadrà ogni volta che viene chiamato un metodo, qualsiasi metodo. Il caricamento automatico delle classi è un concetto simile: possiamo definire cosa succede ogni volta che proviamo a utilizzare una classe non definita, qualsiasi classe non definita. PHP ha un modo per evitare di includere le classi manualmente, automatizzando il processo di caricamento delle classi. È possibile definire una funzione chiamata `__autoload()` che verrà eseguita ogni volta che si tenta di creare un'istanza di una classe non definita. Tale funzione può quindi includere il file di classe appropriato. Il codice seguente mostra un esempio leggermente più sofisticato:

```
function __autoload($className) {  
    include_once __autoloadFilename($className);  
}  
function __autoloadFilename($className) {  
    return str_replace('_', '/', $className) . ".php";  
}
```

La funzione `__autoloadFilename()` genera il nome del file da includere. Una funzione separata per questo renderebbe tutto più facile da testare. Possiamo eseguire un test sulla funzione `__autoloadFilename()` e verificare che il valore restituito sia corretto. Controllare che un file sia stato incluso è più difficile che controllare semplicemente il valore restituito.

La funzione `str_replace` sostituisce tutti i caratteri di sottolineatura con barre. Quindi, se il nome della classe è `HTML_Form`, la funzione `__autoload()` includerà il file `HTML/Form.php`. Ciò semplifica l'ordinamento delle classi in directory diverse nel modo standard PEAR. Se hai classi molto piccole (e ne hai già viste alcune in questo libro), potresti trovare conveniente tenere più di una classe in un file. Puoi combinare ciò con il caricamento automatico creando un collegamento nel file system. Supponiamo che tu abbia una classe `Template` e una classe `Redirect` e che siano entrambe in un file chiamato `Template.php`. In Linux o UNIX, potresti fare questo:

ln -s Template.php Redirect.php

Ora, se usi la classe Redirect, la funzione `__autoload()` includerà il file `Redirect.php`, che sembra essere un collegamento a `Template.php`, che a sua volta contiene la classe Redirect.

La programmazione orientata agli oggetti in PHP è un modo naturale di lavorare, specialmente con i miglioramenti introdotti nella versione 5. Alcune caratteristiche sono comuni a quasi tutti i linguaggi orientati agli oggetti. Puoi definire classi che ti consentono di creare oggetti con il comportamento che desideri; puoi usare i costruttori per controllare cosa succede quando viene creato l'oggetto; e puoi usare l'ereditarietà per creare variazioni su una classe. Le eccezioni forniscono una gestione degli errori più flessibile e leggibile. Essere in grado di gestire gli oggetti per riferimento rende la vita molto più semplice in PHP, in particolare quando si ha a che fare con strutture complesse orientate agli oggetti. La possibilità di chiamare un metodo sul risultato di una chiamata al metodo è conveniente nelle stesse circostanze. La capacità di intercettare le chiamate di metodo e di accedere a variabili di istanza ci consente di risolvere diversi problemi in un modo molto più elegante. Adesso possiamo fare il primo passo nella direzione della programmazione orientata agli aspetti, utilizzando l'overloading per inserire codice prima o dopo tutte le chiamate di metodo (o una selezione di esse) senza dover duplicare tutto. Ci stiamo muovendo gradualmente dalla sintassi di programmazione alla progettazione dell'applicazione. Nel prossimo capitolo daremo un'occhiata ad alcune funzionalità di PHP che fungono da strumenti per la progettazione orientata agli oggetti. Tra questi ci sono restrizioni di visibilità, metodi di classe, classi astratte e interfacce.

CLASSI AVANZATE

Dalle asce di pietra alle compagnie aeree, gli oggetti, reali e tangibili, sono onnipresenti nella tecnologia. Da questo punto di vista, non sorprende che la tecnologia software sia arrivata a dipendere da oggetti virtuali. Le classi, d'altra parte, sono un'altra cosa. Denominare, ordinare elementi in categorie o classi, è inerente al linguaggio naturale, ma parlare di categorie di cose e del processo di denominazione è estraneo alla tecnologia fisica. Le classi nascono dalla filosofia e dalla matematica, a cominciare dagli antichi greci. La combinazione dei due è straordinariamente potente. Nella tecnologia moderna, la fisica astratta e la matematica vengono applicate all'attività concreta di creare cose. La programmazione orientata agli oggetti ripete questo schema: l'astrazione concettuale delle classi e il funzionamento diretto dei singoli oggetti si uniscono, creando una forte sinergia. Inoltre, classi e oggetti hanno sia un'espressione sintattica pratica nel linguaggio che significati concettuali, astratti e semantici. In questo capitolo ci concentreremo su come utilizzare le classi e in particolare sulle funzionalità più utili. Iniziamo studiando le restrizioni di visibilità: come possiamo migliorare l'incapsulamento non lasciando che tutto all'interno della classe sia accessibile dall'esterno. Quindi studiamo come utilizzare la classe come contenitore per metodi, variabili e costanti che appartengono alla classe stessa piuttosto che come istanza di un oggetto. Passiamo a un'altra caratteristica restrittiva: classi e metodi astratti, che possono aiutare a strutturare l'ereditarietà delle classi. Infine, vediamo come funzionano i suggerimenti sui tipi di classe e, infine, osserviamo il funzionamento e il ruolo delle interfacce in PHP.

Visibilità

Un principio centrale del paradigma object-oriented è l'incapsulamento. Un oggetto raggruppa dati e comportamenti che sono naturalmente correlati tra loro. L'azione può aver luogo all'interno dell'oggetto senza che il resto del mondo se ne occupi. Nel capitolo precedente abbiamo confrontato una classe con una casa. L'incapsulamento è come avere del cibo in frigorifero, quindi non dovrai uscire ogni volta che vuoi mangiare. O, forse più opportunamente, quando stiamo programmando, il più delle volte non dobbiamo preoccuparci di quello che succede dentro le mura di casa. Non dobbiamo nutrire la classe dall'esterno. Se il cibo è costituito da dati archiviati in variabili di istanza, i metodi della classe possono mangiarlo senza alcun aiuto aggiuntivo da parte nostra. Per supportare l'incapsulamento, molti linguaggi orientati agli oggetti dispongono di funzionalità che ci aiutano a controllare la visibilità di ciò che è all'interno dell'oggetto. I metodi e le variabili all'interno degli oggetti possono essere resi invisibili all'esterno dell'oggetto dichiarandoli private. Un modo un po' meno restrittivo per farlo è usare protected. PHP ha funzioni e variabili membro private e protected. In realtà sono metodi privati e protetti, non funzioni, poiché sono sempre all'interno di una classe, ma la sintassi per definire un metodo utilizza la parola chiave function. I metodi private e le variabili sono disponibili solo all'interno della stessa classe. I metodi e le variabili protected sono disponibili all'interno della stessa classe e dalle classi genitore e figlio (o più precisamente, antenato e discendente). Un metodo viene contrassegnato come public, private o protected aggiungendo una parola chiave prima della parola function:

```
public function getEmail() {}  
protected function doLoad() {}  
private function matchString($string) {}
```

Le restrizioni di visibilità vengono utilizzate in modo diverso per i metodi e le variabili di istanza (e le variabili di classe), sebbene la sintassi sia simile. In questa sezione, discutiamo prima i metodi e poi le variabili. Esaminiamo perché e come utilizzare metodi privati e protetti, quindi discutiamo del

motivo per cui si consiglia di mantenere private tutte le variabili di istanza. Proviamo usando l'intercettazione invece dei metodi di accesso. Infine (e ironicamente), introduciamo il concetto di classi e metodi final.

Le funzionalità per modificare la visibilità sono spesso assenti o poco appariscenti nei linguaggi tipizzati dinamicamente; questo è logico, poiché questi linguaggi tendono a consentire ai programmatori di fare tutto ciò che sembra utile senza troppi vincoli. D'altra parte, la capacità di controllare la visibilità dei metodi e delle variabili di istanza può essere vista come un'estensione naturale della capacità di restringere l'ambito di una variabile ordinaria nel codice. Limitare la visibilità delle variabili di istanza in genere non è un problema, poiché possiamo sempre fornire un metodo per accedervi. Ma limitare la visibilità dei metodi comporta il rischio di diventare troppo restrittivi, dipende da chi utilizzerà i metodi. Potrebbe essere allettante usarli per assicurarsi che la tua classe venga utilizzata in un modo specifico, che venga utilizzata solo l'API "ufficiale" di una classe. Il problema è che è davvero difficile sapere in anticipo quali metodi saranno utili quando tu, e specialmente qualcun altro, inizierai a riutilizzare una classe. Utilizzeremo metodi privati e protetti negli esempi in questo libro, ma il presupposto di base è che un metodo privato o protetto può essere reso pubblico in qualsiasi momento. Un modo di pensare ai metodi privati e protetti è come una sorta di documentazione, un aiuto alla leggibilità. Rendono più facile vedere come vengono utilizzati i metodi e ti impediscono di usarli in modo errato ma non dettano necessariamente come dovrebbero essere usati in futuro. La visibilità è leggermente diversa in PHP e in Java. Java ha un concetto di package che influisce sulla visibilità. Per impostazione predefinita, un metodo o una variabile di istanza è visibile a qualsiasi classe nello stesso package. I metodi e le variabili protetti sono disponibili per le classi figlie e discendenti e per altre classi nello stesso package. Al contrario, in PHP, la visibilità predefinita è pubblica o meglio, se non indichiamo visibilità, tutto è pubblicamente visibile. PHP manca anche della capacità di Java di rendere le classi private e protette.

Abbiamo discusso il motivo per cui utilizzare le restrizioni di visibilità per i metodi. Supponendo quindi che vogliamo usarli, quando e come li applicheremmo nello specifico? Tratteremo prima i metodi private e poi quelli protected. I metodi private sono spesso metodi di utilità utilizzati ripetutamente in una classe (ma non in qualsiasi altra classe) o metodi utilizzati solo in un'unica posizione. Potrebbe sembrare strano avere un

metodo separato per qualcosa che accade in un solo posto, ma la ragione di ciò è in genere la leggibilità: inserire un pezzo di codice in un metodo separato che ha un nome rivelatore di intenzioni. Il codice seguente è un esempio semplificato di convalida dell'utente. Un amministratore ha modificato un account utente esistente e ha inviato il modulo. Se l'amministratore non ha modificato il nome utente, sta aggiornando l'account utente esistente. È consentito all'amministratore creare un nuovo account utente modificando il nome utente, ma il nome non deve entrare in conflitto con un account utente esistente, altrimenti tale account verrà sovrascritto o duplicato. Per rendere il codice più leggibile, esiste un metodo separato per testare ciascuna di queste situazioni in cui viene accettato l'invio del modulo (nameUnchanged() e nameNotInDB()).

```
class UserValidator {  
    function validateFullUser($user) {  
        if ($this->nameUnchanged($user) ||  
            $this->nameNotInDB()) return TRUE;  
        return FALSE;  
    }  
    private function nameUnchanged($user) {  
        return $_POST['username'] == $user->getUsername();  
    }  
    private function nameNotInDB() {  
        // Query the database, return TRUE if there is no user  
        // with a name corresponding to $_POST['username']  
    }  
}
```

\$user è un oggetto Utente che rappresenta l'utente esistente che viene modificato; in altre parole, l'oggetto le cui proprietà sono state visualizzate nel form.

I metodi protetti in PHP sono disponibili all'interno della stessa classe e da classi antenate o discendenti, ovvero quando la classe che utilizza il metodo eredita dalla classe che contiene il metodo o viceversa. Le opportunità per l'utilizzo di metodi protetti vengono visualizzate quando una classe figlia utilizza un metodo di una classe padre. È utile anche nel caso opposto, quando la classe genitore usa un metodo nella classe figlia.

Questo è leggermente più difficile da comprendere, ma è comunque importante. Vedremo in seguito alcuni esempi completi.

private o protected?

Tecnicamente, le parole chiave private e protected funzionano esattamente allo stesso modo con metodi e variabili di istanza. Ma in pratica c'è una differenza tra variabili e metodi, poiché puoi usare un metodo per ottenere una variabile, ma non il contrario. Ciò significa che è sempre possibile mantenere le variabili membro private o protected purché tu fornisca metodi per ottenere e impostare il valore della variabile:

```
class Document {  
  private $title;  
  //...  
  //...  
  function getTitle { return $this->title; }  
  function setTitle($arg) { $this->title = $arg }  
}
```

In questo modo non impedirai a nessuno di fare qualcosa, stai solo controllando il modo in cui lo fanno. Ecco perché non è quasi mai un problema mantenere private le variabili dei membri. E poiché non è un problema, è generalmente considerata una buona pratica, almeno nei linguaggi che non hanno modo di intercettare l'accesso a una variabile di istanza in modo che il suo significato possa cambiare se necessario. Come accennato, una variabile membro private è quella a cui è possibile accedere direttamente solo dall'interno della classe. In generale, l'ideale è che la variabile venga utilizzata solo all'interno della classe. Se puoi evitare di usarla dall'esterno, è un segno che stai seguendo il principio "dillo, non chiedere". Se è necessario accedere al valore dall'esterno della classe, utilizza gli accessor, noti anche come metodi getter e setter, per ottenere e impostare il valore. Alcuni ti diranno che questo è l'unico modo per farlo: le variabili membro non dovrebbero mai essere pubbliche. Ma trovare una ragione soddisfacente per cui dovrebbe essere così non è necessariamente facile. Non esporre affatto la variabile è una buona idea, ma quando è necessario esporla, perché è necessario utilizzare gli accessor? Alcuni dei ragionamenti non sono del tutto convincenti. Alcuni ti diranno, ad esempio,

che se hai una variabile codice postale, potrebbe essere necessario convalidarla prima di impostarla. Quindi è una buona idea rendere la variabile privata e avere un metodo setter, `setZipCode()`, che si occupa prima di convalidarla. In questo modo nessuno può impostare un valore non valido. Qualcosa di questo tipo:

```
class Address {  
private zipCode;  
function setZipCode($zip) {  
// Do some validation  
$this->zipCode = $zip;  
}  
}
```

Questo è convincente per il caso particolare di un codice postale. Ma spesso, abbiamo solo bisogno di memorizzare il valore così com'è. Allora perché dovremmo usare gli accessor quando non è necessaria alcuna elaborazione? Certo, potrebbe essere necessaria in futuro, ma non lo sappiamo. Quindi cosa succede se manteniamo la variabile `public` fino al momento in cui è necessario eseguire ulteriori elaborazioni? In tal caso tutte le occorrenze della variabile devono essere cambiate in chiamate di accesso. L'unico problema è che non abbiamo modo di essere sicuri dove sia stata utilizzata la variabile. Potremmo non trovarli tutti e questi potrebbero presentarsi come bug fastidiosi. Ecco perché è meglio usare gli accessor fin dall'inizio: cioè dal momento in cui è effettivamente necessario accedere alla variabile. Non vi è alcun motivo per aggiungere funzioni di accesso per tutte le variabili e non vi è alcun motivo per aggiungere un metodo setter per una variabile che può essere di sola lettura. Normalmente, getter e setter dovrebbero soddisfare i requisiti attuali, non ipotetici futuri. L'uso degli accessor è diffuso anche in PHP, puoi trattare una variabile come se fosse privata e fare in modo che tutti gli accessi dall'esterno della classe passino tramite gli accessor. PHP rende la vita un po' più semplice se usi variabili pubbliche e poi vuoi renderle private. Una volta dichiarata la variabile privata, PHP restituirà un messaggio ogni volta che esegui del codice che tenta di utilizzarla dall'esterno della classe. Però esiste un'altra possibilità: puoi usare l'overloading per trasformare quello che sembra un accesso

variabile dall'esterno in una chiamata di accesso. Quindi, per esempio, quando esegui:

```
$email = $message->text;
```

PHP eseguirà:

```
$message->getText();
```

Parola chiave final

La parola chiave final consente di impedire alle classi figlie di estendere una classe sovrascrivendo una classe o un metodo. Ecco un semplice esempio della restrizione imposta da una classe final:

```
final class AccessControl { }  
class MyAccessControl extends AccessControl { }
```

Questo produce il seguente messaggio di errore:

```
class bar may not inherit from final class (AccessControl)...
```

Un metodo final è un metodo di cui non è consentito eseguire l'override in una classe figlia. Un metodo final potrebbe assomigliare a questo:

```
class AccessControl {  
  public final function encryptPassword()  
}
```

Ora questo è vietato:

```
class MyAccessControl extends AccessControl {  
  public function encryptPassword() {}  
}
```

Quando sono utili le classi e i metodi final? Questa non è una domanda facile a cui rispondere. La maggior parte dei libri sul design orientato agli oggetti ignora semplicemente il final. C'è una certa divergenza di opinioni su questo tema. Alcuni dicono che dovresti usare final per prevenire una cattiva progettazione: se pensi che ereditare da una classe (o sovrascrivere un metodo) sarebbe una cattiva idea, rendilo final. Altri si chiedono se ciò sia realisticamente possibile, poiché si tratta di cercare di indovinare in anticipo quali estensioni di una classe saranno necessarie. Gli esempi

precedenti suggeriscono che potrebbero esserci situazioni in cui per ragioni di sicurezza sarebbe saggio l'uso di `final`. Un possibile e più specifico uso di `final` si ha quando un metodo o una classe sono contrassegnati come deprecati. Se un metodo o una classe non dovrebbero essere utilizzati affatto, sembra ragionevole prevenirne l'uso, sovrascrivendolo o estendendolo. In Java, `final` è anche usato con un significato diverso: per definire le costanti di classe. PHP utilizza invece `const`. Abbiamo discusso delle restrizioni di visibilità applicate ai metodi e alle variabili nelle istanze degli oggetti. Ma anche metodi e variabili di istanza possono appartenere alla classe stessa.

Classi senza oggetti

Una classe fornisce una casa virtuale per le istanze dell'oggetto appartenenti alla classe. Può anche memorizzare informazioni indipendenti dalle istanze. Ad esempio, se abbiamo una classe Product e creiamo le istanze Product da una tabella in un database, il nome della tabella appartiene logicamente alla classe piuttosto che a qualsiasi istanza specifica. E potrebbe essere necessario fare qualcosa prima di aver effettivamente creato qualsiasi istanza. Ad esempio, potrebbe essere necessario leggere i dati necessari per creare un'istanza da un database. Questo comportamento, leggendo dal database, è correlato alla classe ma non può essere eseguito da un'istanza della classe. Una possibile casa per questo comportamento è in un metodo di classe: uno che può essere chiamato usando solo il nome della classe anziché una variabile che rappresenta un'istanza:

```
$product = Product::find($productCode);
```

Usiamo i due punti doppi (::) ogni volta che vogliamo accedere a un metodo, variabile o costante appartenente a una classe piuttosto che a un oggetto. C'è sempre un'alternativa. Invece di usare metodi di classe, variabili e costanti, potremmo creare un'altra classe (o classi), le cui istanze forniscano le informazioni e il comportamento appartenenti alla classe. Per esempio:

```
$finder = new ProductFinder;
```

```
$product = $finder->find($productCode);
```

Questo potrebbe essere più flessibile, ma è anche più complesso: c'è una classe aggiuntiva e una riga di codice aggiuntiva per il client. Come in tutte le cose, ci sono sempre pro e contro.

I metodi di classe sono metodi che non vengono eseguiti su un'istanza di oggetto specifica. Sono definiti nella classe, ma funzionano proprio come semplici funzioni, tranne per il fatto che devi usare il nome della classe quando le chiami. La parola chiave per i metodi di classe e le variabili è static. Questa terminologia deriva da C++ e Java ed è di uso comune.

Quindi, sebbene il "metodo di classe" possa essere più appropriato e descrittivo, è comunemente detto metodo statico. In PHP, il metodo statico tipico è definito utilizzando la static function o la static public function:

```
static public function encryptPassword($password) {}
```

Diciamo che abbiamo una classe User ha un metodo insert() per salvare l'oggetto utente nel database. Ha anche un metodo encryptPassword() che accetta una password non crittografata come argomento e restituisce una password crittografata. Quindi, per creare un nuovo oggetto utente e salvarlo nel database, dovresti fare questo:

```
$user = new User(/* Arguments including user name, etc. */);  
$user->insert();
```

E per crittografare una password, dovresti fare questo:

```
$password = User::encryptPassword($password);
```

All'interno della classe User, possiamo usare self per fare riferimento alla classe:

```
$password = self::encryptPassword($password);
```

Questo ha esattamente lo stesso effetto di una semplice funzione:

```
$password = encryptPassword($password);
```

La funzione stessa potrebbe essere identica, ma in un caso è all'interno di una definizione di classe; nell'altro caso no. Se hai un insieme di funzioni procedurali che sembrano essere legate, puoi inserirle in una classe solo per motivi di ordinamento. Quello che hai quindi è in realtà una libreria di funzioni; il fatto che tu stia usando la parola chiave class per definirla non lo rende realmente object-oriented, dal momento che non stai creando

un'istanza di alcun oggetto. In PHP, puoi dichiarare metodi di classe usando la parola chiave static:

```
static public function encryptPassword($password) {  
    return md5($password);  
}
```

La parola chiave static è simile a private e protected in quanto documenta l'uso previsto del metodo e impedisce di utilizzarlo in modo errato involontariamente. Se un metodo è definito come static, non puoi fare nulla di utile con la variabile \$this. Quindi non dovresti provare a fare qualcosa del genere:

```
static public function encryptPassword($password) {  
    return $this->format(md5($password));  
}
```

Qualora scrivessi qualcosa di simile, PHP genererebbe un errore.

Esistono diversi usi per i metodi di classe. Alcuni di quelli più comuni lo sono:

- Metodi di creazione e metodi factory
- Metodi di ricerca
- Codice procedurale
- Sostituzione di costanti

I metodi di creazione e i metodi factory sono metodi che creano e restituiscono istanze di oggetti. Sono usati frequentemente quando la creazione ordinaria usando la parola chiave new diventa insufficiente. I metodi di ricerca (anche detti Finder), per trovare un oggetto in un database o in un altro archivio, possono essere considerati un caso speciale di metodi di creazione, poiché restituiscono una nuova istanza dell'oggetto. In alcuni casi è possibile eseguire con la stessa efficacia un frammento di codice procedurale come con un metodo orientato agli oggetti. Alcuni semplici calcoli o conversioni ne sono un esempio, a volte è importante inserire il codice procedurale in una classe invece di usare semplici funzioni. Il

motivo dell'inclusione in una classe potrebbe essere quello di evitare collisioni di nomi con altre funzioni o perché appartiene a una classe altrimenti basata su metodi di istanza. Il fatto che i metodi statici possano essere usati per tutte queste cose non dimostra che dovrebbero essere sempre usati. I metodi statici hanno il vantaggio della semplicità, ma sono difficili da sostituire al volo, se un metodo appartiene a un'istanza di oggetto, è potenzialmente collegabile. Possiamo sostituire l'istanza dell'oggetto con una diversa per modificare il comportamento in modo significativo senza modificare né il codice client né la classe originale. Esaminiamo il nostro precedente esempio:

```
$finder = new ProductFinder;  
$product = $finder->find($productCode);
```

Se sostituiamo il product finder con un'altra classe (ad esempio, potremmo invece voler ottenere le informazioni sul prodotto da un servizio Web), sia la vecchia classe ProductFinder che la seconda riga nell'esempio possono rimanere le stesse; il finder è collegabile. D'altra parte, usando il metodo statico:

```
$product = Product::find($productCode);
```

In questo caso, il comportamento è integrato nella classe Product e non è possibile modificarlo senza modificare quella riga di codice. Non è un grosso problema se si verifica solo una volta, ma se il nome della classe viene utilizzato ripetutamente, è un'altra questione. Questo problema può diventare particolarmente importante negli unit test: potremmo voler sostituire il metodo find() con un altro, opportunamente modificato, che restituisce dati di test fissi invece di dati effettivi dal database.

Le variabili di classe sono variabili che appartengono a una classe. Per definirne una, usi la parola chiave static come con i metodi di classe:

```
class Person {  
static private $DBTABLE = 'Persons';  
}
```

Il nome della tabella è ora disponibile per tutte le istanze ed è sempre lo stesso per tutte le istanze. Possiamo accedervi utilizzando la parola chiave `self`:

```
$select = "SELECT * FROM ".self::$DBTABLE;
```

In questo esempio, abbiamo dichiarato la variabile privata, quindi non è possibile accedervi dall'esterno della classe. Ma se lo rendiamo pubblico, possiamo riferirci ad esso in questo modo:

```
$select = "SELECT * FROM ".Person::$DBTABLE;
```

Ma quando è opportuno utilizzare una variabile di classe? In questo caso particolare, potremmo invece aver usato una costante di classe. Oppure potremmo aver usato una variabile di istanza, inizializzata allo stesso modo. In questo modo tutte le istanze avrebbero avuto il nome della tabella disponibile. Avremmo potuto ancora usarlo nei metodi di istanza all'interno della classe:

```
$select = "SELECT * FROM ".$this->$DBTABLE;
```

Ma non sarebbe disponibile per i metodi di classe e non sarebbe disponibile al di fuori della classe senza prima creare un'istanza della classe `Person`. Tutto ciò significa che uno degli usi tipici delle variabili di classe e delle costanti di classe è questo tipo di dati: nomi di tabelle, frammenti SQL, altri tipi di sintassi (espressioni regolari, formati `printf()`, formato `strftime()`, e così via). Un altro modo è considerare il fatto che avere molte variabili globali in un programma è sempre una cattiva idea. Se sono presenti, un modo semplice per migliorare la situazione (non necessariamente l'ideale, ma tutto è relativo) è semplicemente raccoglierle in una o più classi sostituendole con variabili di classe pubbliche. Quindi per una classe `Config`:

```
class Config {  
    public static $DBPASSWORD = 'secret';  
    public static $DBUSER = 'developer';  
    public static $DBHOST = 'localhost';
```



```
//...  
}
```

Ora possiamo connetterci a un server MySQL in questo modo:

```
mysqli_connect(Config::$DBHOST,Config::$DBUSER,  
Config::$DBPASSWORD);
```

Ho deliberatamente scritto in maiuscolo i nomi delle variabili per enfatizzare la loro somiglianza con variabili e costanti globali.

Le costanti di classe sono simili alle variabili di classe, ma ci sono alcune differenze chiave:

- Come indica il nome, non possono essere modificate
- Sono sempre pubbliche
- Ci sono delle restrizioni su ciò che puoi metterci dentro
- Sebbene il modo in cui li utilizzi sia simile, il modo in cui li definisci è completamente diverso.

Invece della parola chiave `static`, le costanti di classe vengono definite usando la parola chiave `const`:

```
class Person {  
    const DBTABLE = 'Persons';  
}
```

Ora possiamo accedere alla costante usando `self::DBTABLE` all'interno della classe e `Person::DBTABLE` al di fuori di essa. In questo caso, la costante può sembrare avere tutti i vantaggi rispetto a una variabile. Il nome della tabella non cambierà durante l'esecuzione del programma; quindi, non sembra esserci alcun motivo per utilizzare una variabile. E le costanti non possono essere sovrascritte accidentalmente. Ma c'è una ragione per cui potremmo voler usare comunque una variabile: il test. Potremmo voler usare una tabella di test per il test; sostituire la variabile di classe all'inizio

del test è un modo semplice per tale aspetto. D'altra parte, il fatto che una costante non possa essere modificata può essere positivo per la sicurezza, poiché non verrà mai modificata per scopi dannosi.

Le costanti di classe sono particolarmente utili per le enumerazioni. Se una variabile può avere solo un insieme fisso di valori, puoi codificare tutti i valori fissi come costanti e assicurarti che la variabile sia sempre impostata su uno di questi. Prendiamo come esempio un sistema di autorizzazione molto semplice. Il sistema di autorizzazione ha tre ruoli fissi o categorie di utenti: utente, webmaster e amministratore. Potremmo rappresentare i ruoli come semplici stringhe. Avremmo una variabile \$role il cui valore potrebbe essere "utente", "webmaster" o "amministratore". Quindi, per verificare che l'utente corrente abbia i privilegi di amministratore, potremmo fare qualcosa del genere:

```
<?php if ($role == 'amdinistrator'): ?>  
<a href="edit.php">Edit</a>  
<?php endif; ?>
```

L'unico problema è che la parola "administrator" è scritta in modo errato, quindi il test non funzionerà. È un bug, ma può essere evitato usando le costanti. In PHP c'è un modo più ordinato per fare tutto ciò, chiamato costanti di classe:

```
class Role {  
    const REGULAR = 1;  
    const WEBMASTER = 2;  
    const ADMINISTRATOR = 3;  
    //...  
}
```

Ora possiamo invece fare questo:

```
<?php if ($role == Role::ADMINISTRATOR): ?>
```

Non riusciremo a farla franca con errori di ortografia neanche in questo caso; l'utilizzo di una costante di classe non definita è un errore

irreversibile. Tuttavia, rispetto alle costanti globali, questo è più facile da capire, anche perché sappiamo dove è definita la costante (all'interno della classe Role), basta solo guardarla. Ma usare le costanti di classe dall'esterno di una classe non è necessariamente il modo migliore per farlo. Lasciare il compito di testare il ruolo a un oggetto potrebbe essere la cosa migliore da fare.

```
<?php if ($role->isAdministrator()): ?>
```

Ciò nasconde ulteriori informazioni dal codice client. È un esempio di un principio chiamato "dillo, non chiedere". In generale, è meglio lasciare che un oggetto lavori sui propri dati piuttosto che chiedere i dati ed elaborarli. Nel secondo esempio, stavamo usando la costante dall'esterno della classe Role. Se dovessimo usarlo all'interno della classe per decidere il comportamento dell'oggetto, potremmo iniziare a considerare un'altra opzione: usare l'ereditarietà per differenziare il comportamento delle diverse categorie di utenti. Quindi avremmo sottoclassi di Role che potrebbero essere chiamate AdministratorRole, WebmasterRole e UserRole.

Le costanti di classe vanno bene fintanto che sono disposte a fare ciò di cui abbiamo bisogno, ma i loro limiti tendono a manifestarsi presto. Il valore di una costante può essere impostato solo quando è definito e non può essere definito all'interno di metodi in una classe. È possibile assegnare solo valori semplici a una costante; non c'è modo di assegnargli un oggetto, infatti, non puoi nemmeno usare la concatenazione di stringhe quando definisci una costante. Come per la maggior parte delle limitazioni sintattiche, c'è sempre la possibilità che queste siano cambiate nel momento in cui lo leggerai. E come accennato, non c'è modo di sostituire la costante a scopo di test. Per tutti questi motivi, dobbiamo sapere cosa fare quando dobbiamo sostituire una costante di classe.

La sostituzione più semplice e più ovvia per una costante di classe è una variabile di classe, in genere pubblica. Poiché le variabili possono essere modificate dopo che sono state definite, possiamo farlo all'interno di un metodo o di una funzione, dandoci la possibilità di assegnargli un oggetto o il risultato di qualsiasi tipo di elaborazione. Ma assicurarsi che accada è leggermente complicato. Possiamo farlo nel costruttore dell'oggetto, ma poi la variabile non sarà disponibile finché non avremo creato la prima istanza della classe. Ovviamente, se possibile, potremmo crearne una subito dopo la

dichiarazione di classe. O più semplicemente, potremmo avere un metodo di classe per inizializzare le variabili di classe ed eseguirlo. Se utilizziamo due diversi database MySQL rbac e cms, potremmo rendere disponibile una connessione a ciascuno di essi in questo modo:

```
class Connections {  
public static $RBAC;  
public static $CMS;  
public function init() {  
self::$RBAC =  
new mysqli('localhost','user','password','rbac');  
self::$CMS =  
new mysqli('localhost','user','password','cms');  
}  
}  
Connections::init();
```

Potrebbe sembrare brutto e poco elegante, ma almeno funziona. Ma ora potremmo anche rendere private le variabili e aggiungere metodi di accesso statici:

```
public static function getRbac() { return self::$RBAC; }  
public static function getCms() { return self::$CMS; }
```

Un metodo di classe di sola lettura è spesso un sostituto perfettamente valido per una costante. Inoltre, possono sembrare quasi identici. Puoi sostituire `Person::DBTABLE` con `Person::DBTABLE()`:

```
public static function DBTABLE() { return 'Persons'; }
```

È semplice e funziona anche in vecchie versioni di PHP. All'interno di un metodo, non abbiamo limiti a ciò che possiamo fare. Ad esempio, se vogliamo riutilizzare una lunga istruzione SQL che può essere più facilmente formattata usando la concatenazione, possiamo farlo:

```
class UserMapper {  
public static function sqlSelect($id) {
```

```
"SELECT user_id,email,password,firstname,lastname,".  
"username,role+0 as roleID FROM Users WHERE id = $id";  
}  
}
```

Le variabili e le costanti di classe sono state introdotte in PHP 5, vediamole nel dettaglio.

CLASSI ASTRATTE E INTERFACCE

Le classi astratte, che sono un'altra caratteristica di PHP, sono composte da un aspetto concettuale e uno pratico. Poiché prediligiamo l'aspetto pratico, vediamo cosa fa effettivamente una classe astratta. Esamineremo il funzionamento di base di classi e metodi astratti e poi vedremo come possono essere applicati a una classe dall'esempio "Hello world" del capitolo precedente. Rendere astratta una classe è semplice, basta usare `abstract class` anziché solo `class`. Dopo aver fatto ciò, non siamo più autorizzati a creare un'istanza della classe. Quindi non dovresti fare:

```
abstract class Foo {}  
$foo = new Foo;
```

In tal caso, riceverai questo messaggio:

Cannot instantiate abstract class Foo

Allora che senso ha avere una classe astratta? È utile perché un'altra classe, che non è astratta, in altre parole una classe concreta, può ereditare da essa. Un metodo astratto è in realtà solo una dichiarazione di una firma di metodo che può essere utilizzata dalle classi figlie.

```
abstract protected function createAdapter(DomElement $element);
```

Questo cosiddetto metodo non fa nulla; semplicemente finge di essere importante. È davvero solo una firma del metodo, ma si chiama metodo nonostante ciò. Tecnicamente, la relazione tra metodi astratti e classi astratte è che se si dichiara un metodo astratto, anche la classe che lo contiene deve essere dichiarata astratta. In altre parole, una classe concreta non può avere metodi astratti, ma una classe astratta può avere metodi concreti. Nel nostro esempio di ereditarietà nel capitolo precedente, avevamo una classe `HtmlDocument` e una classe figlio "Hello world". Se si utilizza la classe `HtmlDocument` da sola, verrà restituita una pagina HTML vuota. Quindi non ha molto senso usarlo se non indirettamente tramite i

suoi figli. In altre parole, non c'è niente di male nel dichiararla astratta. Già che ci siamo, potremmo anche dichiarare abstract il metodo getContent().

```
abstract class HtmlDocument {  
  public function getHtml() {  
    return "<html><body>".$this->getContent().  
    "</body></html>";  
  }  
  abstract public function getContent();  
}
```

Il metodo abstract non fa altro che forzare le classi figlie a implementare il metodo getContent(). Non avevamo alcuna utilità per il comportamento precedente di questo metodo - restituire una stringa vuota - poiché tutto ciò che ne otteniamo è un documento HTML vuoto, che non è la cosa più eccitante che puoi visualizzare in un browser Web. Invece, ci aspettiamo che tutte le classi figlie implementino questo metodo con un metodo che restituisce del testo.

Cosa abbiamo ottenuto facendo questo? Abbiamo evitato due possibili errori: creare un'istanza della classe HtmlDocument stessa e dimenticare di implementare il metodo getContent() in una classe figlia. Inoltre, abbiamo reso il codice un po' più chiaro. Le parole chiave abstract class comunicano, a qualcuno che lo sta leggendo per la prima volta, che ci sono classi figlie di HtmlDocument. L'intenzione della classe, quella di essere una classe madre senza un lavoro indipendente da svolgere, è più chiara. Uno degli scopi delle classi astratte è supportare suggerimenti sui tipi di classe che non siano eccessivamente specifici, ma facciamo un esempio per chiarire.

Se ordini la pizza e ti viene consegnata un'enciclopedia, rimani affamato. Peggio ancora, è un segno che hai il numero di telefono sbagliato e non c'è stata una buona comunicazione con le persone che hanno preso il tuo ordine. O forse il corriere aveva l'indirizzo sbagliato. Allo stesso modo, se un metodo o una funzione riceve il tipo di input sbagliato, è spesso un sintomo della presenza di un bug grave. Una funzione che esegue calcoli matematici non funzionerà con stringhe o oggetti PEAR DB. E se gli stai passando oggetti PEAR DB, molto probabilmente hai commesso un errore.

I linguaggi tipizzati staticamente ti obbligano a specificare il tipo di ogni singolo argomento per ogni singolo metodo. Considerando il fatto che

i problemi con i valori di input sono relativamente rari, questo può sembrare eccessivo. D'altra parte, un bug può causare molti problemi. Quindi alcuni programmatori hanno cercato di implementare varie soluzioni alternative per il controllo del tipo in linguaggi tipizzati dinamicamente. Tuttavia, questi tendono ad essere incompleti e ingombranti. Diamo un'occhiata a come funzionano i suggerimenti per il tipo e poi discuteremo quando saranno utili. PHP ha una soluzione incompleta, ma non ingombrante. PHP può controllare i tipi di argomenti del metodo per te in modo da non dover scrivere codice condizionale esplicito per tale scopo. Al momento della stesura di questo articolo, questo è vero solo se gli argomenti sono oggetti o array. Come accennato, i linguaggi tipizzati staticamente come Java richiedono di specificare il tipo di ciascun argomento in un metodo. Un metodo Java inizierà qualcosa del genere:

```
public void addDocument(Document document) {
```

In tal caso void è il tipo restituito, significa che il metodo non restituirà nulla e Document significa che il singolo argomento deve essere un oggetto Document o un oggetto appartenente a una sottoclasse di Document. PHP ti consente di fare qualcosa di simile con qualsiasi argomento che sia un oggetto o un array. Quindi potrai scrivere questo:

```
public function addDocument(Document $document) {}
```

A differenza di Java, PHP non ti dirà che hai commesso un errore finché non esegui il codice. Il precedente equivale a:

```
public function addDocument($document) {  
    if (!$document instanceof Document) {  
        die("Argument 1 must be an instance of Document");  
    }  
}
```

Quindi, se NewsArticle è una sottoclasse di Document, puoi passare un oggetto NewsArticle al metodo, ma non una semplice stringa o un oggetto User. La buona notizia sul suggerimento del tipo è che ricevi un avviso in

anticipo se hai commesso un errore. Senza il suggerimento sul tipo, non riceverai un messaggio di errore finché non proverai a utilizzare l'oggetto o il valore in modo inappropriato, ad esempio chiamando il metodo sbagliato su di esso. Verificare che gli argomenti dei metodi siano validi può aumentare l'affidabilità e semplificare la fase di debug rilevando gli errori in anticipo. Ma questa è un'arma a doppio taglio: se aggiungi un codice di controllo degli errori dove non è necessario, ingombrerà il tuo codice e lo renderà meno leggibile. Questo, a sua volta, può far sì che il codice diventi meno affidabile, poiché rende più difficili da trovare bug e falle di sicurezza. Quindi, in generale, è meglio evitare troppi controlli a meno che l'interfaccia non sia quella che sai potrebbe essere utilizzata in modo errato. Una buona copertura del test riduce la necessità di controllo. D'altra parte, se stai scrivendo una classe che dovrebbe essere utilizzata da persone che potresti non conoscere, controllare i tipi di argomento è più che doveroso.

I suggerimenti sul tipo coinvolgono poco codice e migliorano la leggibilità piuttosto che diminuirla. In un codice ben strutturato e orientato agli oggetti con molte classi piccole, la cosa più difficile da capire potrebbe essere l'interazione tra le classi e conoscere i tipi di argomenti rende più facile svelare le relazioni (l'alternativa è usare i commenti). Ma il vero svantaggio dei suggerimenti sui tipi è la dipendenza. Ogni suggerimento di tipo è una dipendenza da qualsiasi classe o interfaccia a cui si riferisce il suggerimento. Pertanto, sebbene i suggerimenti sul tipo possano facilitare la ricerca dei bug, rendono anche più difficile modificare il codice. Se modifichi il nome di una classe, potrebbe essere necessario modificare tutti i suggerimenti per il tipo per quella classe. Ad esempio, potremmo avere una classe chiamata `Date` e trovarci in questa situazione:

```
$this->setDate(Date $date);
```

Ora, se cambiamo il nome della classe `Date` in `DateMidnight`, dovremo cambiare tutti quei suggerimenti di tipo. I programmatori che sono abituati a gestire questo genere di cose in Java potrebbero dirti che dovresti digitare suggerimento su un'interfaccia per evitarlo, ma scoprire quale interfaccia ti serve è tutt'altro che banale. È più probabile che i suggerimenti di tipo siano utili nei costruttori rispetto ad altri metodi. Spesso un costruttore accetta un oggetto come argomento, memorizza l'oggetto in una variabile di istanza e lo utilizza in seguito. Possono succedere molte cose tra il momento in cui

viene inserito e il momento in cui l'oggetto viene utilizzato; controllando il tipo quando viene passato al costruttore, possiamo ottenere un possibile errore molto prima e potrebbe essere più facile trovare il bug. I suggerimenti di tipo in PHP ti consentono di controllare solo oggetti e array, non semplici tipi di dati. A meno che non si scriva un codice orientato agli oggetti piuttosto sofisticato, è probabile che la maggior parte degli argomenti del metodo siano semplici stringhe e numeri. D'altra parte, potremmo usare suggerimenti di tipo per tipi di dati semplici avvolgendo (con il wrapping) l'elemento di una classe. Potremmo anche creare classi String e Integer che non hanno scopo se non quello di segnalare il tipo dei contenuti. Più utile, probabilmente, sarebbe introdurre una classe specializzata il cui nome indicasse il significato del dato. Si consideri un tipico metodo query() per un oggetto di connessione al database. Potremmo usarlo in questo modo:

```
$db->query('SELECT * FROM Log');
```

Se volessimo fare qualcosa per impedire la possibilità di passare una stringa o un numero irrilevante a questo metodo, potremmo introdurre un suggerimento di tipo che richiede un oggetto SqlStatement:

```
class DbConnection {  
    public function query(SqlStatement $sql) {}  
}
```

L'oggetto SqlStatement può essere creato dalla stringa in un modo semplice:

```
$db->query(new SqlStatement('SELECT * FROM Log'));
```

Nella sua forma più semplice, l'oggetto SqlStatement sarebbe solo un semplice wrapper per la stringa. E per renderlo utile in più di questo caso specifico, potremmo avere una classe wrapper generale ed estenderla per creare la classe SqlStatement:

```
abstract class StringHolder {  
    protected $string;
```

```
public function __construct($string) {  
    $this->string = $string;  
}  
public function getString() {  
    return $this->string;  
}  
}  
class SqlStatement extends StringHolder {}
```

Non sto sostenendo questo approccio; sto semplicemente sottolineando la possibilità. Sarebbe chiaramente eccessivo per un uso generale, ma potrebbero esserci circostanze in cui il controllo del tipo è abbastanza importante da rendere questo particolare tipo di complessità degno di nota. L'uso dei suggerimenti sul tipo può essere problematico se si è vincolati all'utilizzo di un nome di classe specifico, poiché la modifica del nome della classe richiede la modifica di tutti i suggerimenti sul tipo. Un modo per ottenere più libertà nell'uso dei suggerimenti sui tipi è utilizzare le interfacce.

Interfacce

La parola interfaccia ha un significato semantico e, in alcuni linguaggi di programmazione, anche sintattico. Nella programmazione orientata agli oggetti, la parola interfaccia in genere indica l'insieme di messaggi a cui un oggetto può rispondere, l'insieme di operazioni che può eseguire. È il linguaggio molto piccolo e limitato che l'oggetto comprende. In senso sintattico, le interfacce sono un modo per dichiarare formalmente questo minuscolo linguaggio. E puoi definire quali classi di oggetti rispondono a quel particolare linguaggio minuscolo. Se non si riesce a dare agli oggetti designati la capacità di comprendere il linguaggio, il compilatore restituirà degli errori. Ma quanto sono davvero utili e per quali scopi? Indaghiamo. Inizieremo vedendo come funzionano le interfacce e discuteremo se sono necessarie in PHP. Quindi vedremo un paio di usi per le interfacce: rendere il design più chiaro e migliorare i suggerimenti sui tipi. Infine, vedremo come le interfacce in PHP differiscono dalle interfacce in Java.

Tecnicamente, un'interfaccia è un costrutto simile a una classe che dichiara un numero di metodi. In effetti, è simile a una classe astratta che ha solo metodi astratti:

```
interface Template {  
public function __construct($path);  
public function execute();  
public function set($name,$value);  
public function getContext();  
}
```

Una classe basata su questa interfaccia può essere dichiarata usando la parola chiave implements:

```
class SmartyTemplate implements Template {}
```

Tutto ciò significa che la classe SmartyTemplate deve avere tutti i metodi nell'interfaccia e le firme dei metodi devono essere le stesse. Quindi set() deve avere due argomenti. La differenza tra questa e una classe astratta che ha solo metodi astratti è che una classe può estendere solo un'altra classe, ma può implementare più di un'interfaccia:

```
class DateRange extends Range implements TransposableRange,  
ComparableRange {}
```

Abbiamo anche bisogno di interfacce in PHP o sono solo formalità inutili, pretenziose e che ostacolano le prestazioni? Hanno un valore concreto e pratico? Bene, prendiamo prima in considerazione il problema delle prestazioni. Le interfacce non influiranno molto sulle prestazioni a meno che non inseriamo l'interfaccia in un file separato. L'apertura di un file richiede sempre tempo. Quindi dobbiamo esserne consapevoli. Quello che fanno le interfacce, in senso pratico, è quasi nulla. In effetti, se non stiamo utilizzando suggerimenti sul tipo di classe e il nostro codice è corretto, le interfacce non hanno alcun effetto sull'esecuzione del codice. Le interfacce sono principalmente un modo per rendere le cose esplicite e per prevenire alcuni errori. Ma sebbene le interfacce non siano strettamente necessarie, possono avere un certo valore. Rendono più espliciti alcuni dei nostri progetti e possono prevenire errori banali. Ma nel momento in cui scrivo, le interfacce in PHP esistono da troppo poco tempo per esprimere giudizi fermi su quando sono utili e quando non lo sono. Quello che possiamo fare è esplorare alcuni possibili vantaggi e svantaggi, alcune intuizioni e alcune insidie.

Un'interfaccia è praticamente la stessa cosa di una classe astratta senza metodi implementati, tranne per il fatto che una classe può avere solo una classe genitore, ma implementare un numero qualsiasi di interfacce. I linguaggi tipizzati dinamicamente non hanno tradizionalmente classi e interfacce astratte. Questo ha un senso pratico, ma concettualmente parlando lascia a desiderare. Quasi tutti i moderni linguaggi di programmazione hanno classi ed ereditarietà. Quindi puoi avere classi che hanno parte della loro implementazione in comune ed ereditarla dal loro genitore. Tipicamente, tali classi hanno sia questa relazione pratica, l'implementazione comune, sia una relazione concettuale. Ad esempio, se abbiamo una classe Document che è il genitore della classe Message e della classe NewsArticle, queste sono chiaramente correlate concettualmente. Se si rimuove l'implementazione comune (di solito ci sono modi diversi per farlo), la relazione concettuale rimane. Sembra ragionevole che un linguaggio ci permetta di esprimere questa relazione anche se non esiste un'implementazione comune. Se le classi astratte, anche senza metodi

concreti, e le interfacce vengono utilizzate per esprimere le somiglianze tra le classi e il modo in cui funzionano, rende espliciti questi aspetti concettuali o di progettazione e ciò può facilitare la comprensione di come è strutturato il codice. Proprio come con le classi padre, le interfacce possono essere utilizzate per suggerimenti sul tipo di classe. Ciò significa che il metodo seguente accetterà un oggetto appartenente a una classe che implementa l'interfaccia Template o che estende una classe denominata Template:

```
public function display(Template $template) {}
```

Se un suggerimento sul tipo fosse troppo specifico, potrebbe essere troppo restrittivo. Ad esempio, il metodo display() potrebbe essere in grado di utilizzare oggetti che non sono Template. Ad esempio, potrebbe essere in grado di accettare un oggetto Redirect che eseguirebbe un reindirizzamento HTTP invece di generare codice HTML come fanno abitualmente i Template. Può essere utile avere una certa libertà su quanto restrittivo diventa il suggerimento del tipo. Una classe non può estendere più di una classe, ma le interfacce sono illimitate; questo ci permette di esprimere relazioni astratte che esulano dalle gerarchie ereditarie. Per esprimere la somiglianza tra un oggetto Redirect e un oggetto Template, il fatto che sono intercambiabili in alcuni contesti, possiamo usare un'interfaccia chiamata qualcosa come Response:

```
interface Response {  
    public function execute();  
    public function display();  
}
```

Una classe template specifica potrebbe estendere una classe template e implementare l'interfaccia Response:

```
class FormTemplate extends Template implements Response {}
```

In PHP, abbiamo molto controllo sul grado di restrizione per i suggerimenti sui tipi. Ad un estremo, possiamo semplicemente lasciarli fuori e, a meno che l'oggetto che passiamo al metodo non sia incompatibile, non avremo problemi. All'altro estremo, possiamo basare il suggerimento di tipo su

un'interfaccia o su una classe genitore che ha gli stessi suggerimenti di tipo. Il modo meno restrittivo per utilizzarli senza saltarli del tutto è utilizzare un'interfaccia priva di metodi, un'interfaccia che comunica un certo aspetto o qualità di un tipo di dati. Avremmo potuto definire l'interfaccia Response in questo modo:

```
interface Response {}
```

Nella programmazione Java, questo tipo di interfaccia è talvolta chiamata interfaccia di tag. Ma c'è un'importante differenza tra Java e PHP: quando si nomina un'interfaccia come il tipo di argomento di un metodo in Java, il compilatore non consentirà di chiamare metodi su quell'oggetto che non sono definiti nell'interfaccia. Il suggerimento sul tipo PHP è solo un semplice controllo che l'oggetto sia effettivamente del tipo corretto; puoi chiamare qualsiasi metodo, per quanto inappropriato:

```
public function display(Template $template) {  
    $template->nonExistentMethod();  
}
```

Naturalmente, questo metodo fallirà non appena PHP tenterà di eseguire nonExistentMethod(), ma il suggerimento di tipo non ha nulla a che fare con questo. Non esiste una relazione diretta tra il suggerimento del tipo e la chiamata al metodo. Ce n'è solo uno debole e indiretto, poiché possiamo essere sicuri di evitare questo tipo di errore se ci assicuriamo di non utilizzare metodi che non sono definiti nell'interfaccia Template. Ad ogni modo, poiché non esiste un tale controllo in PHP, possiamo sempre utilizzare un'interfaccia di tag al posto di una "reale".

È già stata menzionata una notevole differenza tra le interfacce Java e PHP: i suggerimenti di tipo PHP non limitano i metodi che è consentito chiamare. Oltre a ciò, ci sono alcune piccole differenze nel modo in cui funzionano le interfacce in Java e PHP. Una differenza consiste nel fatto che una classe in PHP non può implementare due diverse interfacce se entrambe contengono lo stesso metodo. Java non ha questa restrizione. A differenza di Java, PHP consente di includere il costruttore nell'interfaccia. In genere non è utile definire il costruttore come parte dell'interfaccia, poiché la firma

del costruttore è spesso ciò che deve variare tra classi simili. Ma è importante sapere che puoi farlo e potrebbero esserci situazioni in cui sarebbe utile. Ci sono anche un paio di somiglianze che potrebbero non essere evidenti. Sia in Java che in PHP, un'interfaccia può estendere un'altra interfaccia utilizzando la parola chiave `extends`. E in entrambi, puoi aggiungere la parola chiave `abstract` ai metodi nell'interfaccia, ma non è richiesta, poiché tutte le dichiarazioni di metodo in un'interfaccia sono comunque astratte.

Dato che i concetti sono tanti, facciamo un breve riepilogo. La programmazione di base orientata agli oggetti fornisce un'organizzazione a due livelli: classi e metodi in cui possono convivere dati e diversi blocchi di codice che possono convivere. L'ereditarietà è un modo semplice per le classi di condividere il codice. Le restrizioni di visibilità definiscono ciò che è visibile al di fuori della classe e ciò che non lo è. I metodi e le variabili private e `protected` aiutano a rendere più leggibile il codice orientato agli oggetti e aiutano a incapsulare il contenuto di un oggetto. Interfacce, classi astratte e suggerimenti sui tipi non sono strettamente necessari in PHP, ma possono darci più spazio per esprimere tipi e astrazioni. Queste caratteristiche hanno lo scopo di semplificare lo sviluppo di strutture e design complessi.

I meriti relativi del codice orientato agli oggetti rispetto a quello procedurale sono talvolta dibattuti anche dai guru della programmazione orientata agli oggetti. La programmazione object-oriented è una buona idea, ma non sempre. Fatta eccezione per alcuni scenari, non c'è motivo per rendere "Hello world" più complesso di questo script PHP di base:

```
echo "Hello world!\n";
```

In Java, devi scrivere una classe per fare qualcosa di semplice come stampare "Hello world"; in PHP, sei libero di ignorare la programmazione orientata agli oggetti per tutto il tempo che desideri. In pratica, puoi farlo anche in Java e linguaggi simili, dal momento che sei libero di scrivere una classe che contiene solo blocchi di codice procedurali, o anche un lungo metodo procedurale. Non è orientato agli oggetti in alcun senso significativo della parola, ed è davvero l'equivalente di semplici script e funzioni PHP. Il codice extra che va nella scrittura di una classe solo per produrre "Hello world" è sprecato. È un bagaglio inutile. Ma l'orientamento

agli oggetti non è stato inventato per risolvere semplici problemi; la sua utilità sta nel facilitare la comprensione e la risoluzione di problemi complessi. Da qualche parte tra l'esempio "Hello world" e la vasta applicazione aziendale si trova una soglia alla quale la programmazione orientata agli oggetti diventa più efficace della programmazione procedurale a lungo termine. Dico "a lungo termine" perché il vantaggio principale dell'OOP è semplificare la manutenzione delle applicazioni. Anche se inizialmente richiede uno sforzo leggermente maggiore, può essere ripagato con meno lavoro quando si aggiungono nuove funzionalità e si correggono i bug. Tendo a pensare che tale soglia sia bassa. Trovo che scrivere classi mi aiuti anche quando il programma è relativamente piccolo e semplice. La programmazione orientata agli oggetti semplifica la scomposizione del programma in parti e poiché devi dare un nome alle parti (le classi), è più facile vedere cosa stanno facendo. Puoi farlo anche con funzioni semplici, ma man mano che le funzioni si moltiplicano, diventa più difficile tenerne traccia. Alla fine, devi scoprirlo da solo. Se ritieni che il design procedurale sia più semplice e facile da capire e che renderlo orientato agli oggetti lo complichino inutilmente, mantieni il design procedurale. Non renderlo orientato agli oggetti solo perché sembra interessante.

CONCLUSIONI

Sicuramente in questo ebook non abbiamo affrontato tutti i casi e tutte le possibili combinazioni che potresti trovare mentre programmi in PHP. Abbiamo analizzato le basi, partendo da una solida teoria sulla quale costruire alcuni esempi per facilitare la comprensione. Se hai imparato tutte le tecniche descritte qui, sei sulla buona strada per diventare un bravo sviluppatore PHP e con un po' più di sforzo, entrerai nel livello avanzato.

Non ti preoccupare se hai dovuto faticare a capire, ripassa i capitoli e vedrai che sarà tutto più chiaro. Non dimenticare che più ti alleni, più tutto diventa facile.

Se stai pensando: «*Come posso ricordare tutto questo?*», la risposta è che non è necessario ricordare tutto. Siamo nell'era del digitale quindi molti prima di te avranno avuto dubbi su come usare un loop ed è facile sbagliare se sei alle prime armi. Non vergognarti di cercare le cose. Ciò che rende semplice la programmazione per il Web non è una conoscenza enciclopedica delle funzioni e delle classi di PHP, ma una solida teoria su come funzionano le istruzioni condizionali, i loop e altre strutture che controllano il flusso di uno script.

Quando sei in grado di analizzare i tuoi progetti in termini di "se ciò accade, cosa dovrebbe succedere dopo?" sei già ad un livello successivo e sei tu che controlli il codice. Consulto frequentemente il manuale online di PHP e per me è come un dizionario, il più delle volte, voglio solo controllare di ricordare bene. Spesso trovo che qualcosa attira la mia attenzione e apre nuovi orizzonti, magari una nuova funzione che consente di risparmiare tempo e qualche riga di codice. Ti ringrazio per aver letto questo ebook e spero che questo sia solo il punto di partenza per costruire dei bellissimi siti Web, con molte funzioni in PHP, sfruttando tutte le sue potenzialità.