Guida a PHP 5

autore : Casula Francesco
Pubblicato il 23/06/2007 - 19:17

Avanti

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 23/06/2007 - 19:17

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 23/06/2007 - 19:17

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 23/06/2007 - 19:17

Indietro

Tags : Programmazione Guida Php 5
Copia link a questo articolo

Home

Guide Tutorials Download Curiosità Mappa del Sito

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 24/06/2007 - 18:04

Introduzione

Lo scopo di questa guida è quello di portare le conoscenze di un neofita del PHP, a livello professionale.

Anche chi non ha mai programmato in PHP può tranquillamente avvicinarsi a questo fantastico linguaggio, per la prima volta, con questa guida che illustrerà passo passo e in modo dettagliato tutte le caratteristiche di PHP 5.

Dalla struttura del linguaggio, alle nozioni necessarie per creare delle vere e proprie applicazioni web.

PHP è un linguaggio Server-Side che vi permetterà di creare delle pagine dinamiche, scrivendo il vostro codice anche in mezzo alla stessa pagina HTML.

Server-Side significa che il vostro codice risiederà solo sul server, e che non sarà pertanto possibile accedere ai sorgenti dal lato client, come invece accade con linguaggi Client-Side come JavaScript.

In pratica, quando un utente apre una vostra pagina PHP, il Web Server viene interrogato e restituisce al client niente altro che una semplice pagina HTML o XHTML ecc...

La pagina HTML che il client visualizzerà, sarà prodotta da PHP grazie alle vostre direttive.

Prima di iniziare a programmare, è necessario che vi procuriate il software necessario.

Per testare le vostre applicazioni, AppServ andrà più che bene.
E' un pacchetto autoinstallante per piattaforma Windows, che installerà e configurerà automaticamente tutti i pacchetti di cui avrete bisogno, nel vostro PC.

Al momento in cui scrivo è disponibile la versione **AppServ 2.5.9** :

- Apache 2.2.4
- PHP 5.2.3
- MySQL 5.0.45
- phpMyAdmin-2.10.2
- MySQL Root Password Reset

Una volta installato AppServ, aprite il vostro browser e digitate questo indirizzo per vedere se l'installazione ha avuto successo :

```
http://localhost/
```

Se vi appare una pagina rosa con la scritta **"The AppServ Open Project"** allora è tutto ok.

Ora vi serve solo un editor con cui scrivere e modificare il vostro codice, e al riguardo vi segnalo l'ottimo e

gratuito PSPad con cui ho realizzato questo sito, in concomitanza con Aptana.

Nella prossima pagina vedremo come configurare in modo ottimale PSPad, per programmare in PHP.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 25/06/2007 - 20:27

Configurare PSPad

Innanzitutto dobbiamo procurarci la documentazione ufficiale di PHP 5, che daremo in pasto a PSPad.
E' essenziale che prendiate proprio la versione linkata in formato **CHM**.

Vi tornerà molto utile, perchè contiene **tutte** le funzioni che PHP 5 ci mette a disposizione, con quasi sempre, la rispettiva descrizione in lingua italiana.

Una volta installato PSPad, ed impostato su **"Italiano"**, andiamo su **Configurazione** -> **Programma** -> **Server WEB** e ci assicuriamo che la voce "Server" sia impostata su **"localhost"** e che la "Cartella radice" punti alla path di AppServ o del Server Web che avete installato.
Se avete installato AppServ allora probabilmente sarà "C:\AppServ\www".

Per vedere una pagina su **localhost** con AppServ quindi, sarà sufficiente metterla dentro la cartella **www**.

Ancora su PSPad, nella stessa finestrella (Programma) andiamo sotto **Integrazione con il Sistema** -> **Tipi di file registrati**
e aggiungiamo l'estensione **".php"**.

> Ancora nella finestrella **Programma** andiamo su **File e cartelle** e impostiamo la voce **"Tipo nuovo file"** come PHP,
> spuntando **"Sempre questo tipo"**.
>
> Ora chiudete la finestrella **Programma** e andate su **Configurazione** -> **Evidenziazione sintassi**.
>
> Scegliete PHP nella lista dei linguaggi a sinistra e andate poi sulla scheda **"Caratteristiche"**.
>
> Nell'opzione **"File di Guida"** sfogliate per cartelle e dategli il file **CHM** scaricato in precedenza.
> In questo modo PSPad potrà evidenziare correttamente il nostro codice PHP, e potrà fornirci dei suggerimenti sulle funzioni.

Ora nella scheda **"Compilatore"** andate alla voce **"Compilatore"** e, sfogliando per cartelle, dategli il file **php.exe**.
Se il vostro Server Web è AppServ, probabilmente lo troverete in "C:\AppServ\php5\php.exe".

Ora dal menù **Visualizza** disabilitiamo **"Riga corrente"** e abilitiamo **"Numeri di riga"** (opzionale).

Scegliete **"Nuovo"** dal menù **File** e andate alla prossima pagina per scrivere il vostro primo script PHP.

Tags : Guida Php 5 Pspad
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 26/06/2007 - 22:51

Hello World!

Come consuetudine in tutti libri di programmazione, non potevo certo farvi mancare il capitolo"**Hello World!**".

Vediamo rapidamente il modo più semplice di scrivere una pagina PHP.
Questo metodo lo useremo per la prima parte di questa guida fino a quando introdurrò l'utilizzo dei file esterni.

Aprite il vostro editor preferito e iniziate a scrivere ...
Tutto quello che scriverete fra i tag ...

```
<?php      ?>
```

... verrà interpretato come codice PHP.

Ecco il codice della nostra prima pagina PHP :

```
<html>
     <head>
          <title>La mia prima pagina PHP</title>
     </head>

     <body>
          <h1>Intestazione della pagina</h1>
          <br />
          <?php
               echo "Hello World!\n";
               echo "\t\t<br />Questo messaggio viene da PHP\n";
          ?>
     </body>
</html>
```

Compilando la pagina col compilatore PHP (Ctrl+F9 in PSPad) otterete il sorgente della pagina HTML :

```
<html>
     <head>
          <title>La mia prima pagina PHP</title>
     </head>

     <body>
          <h1>Intestazione della pagina</h1>
          <br />
          Hello World!
```

```
        <br />Questo messaggio viene da PHP
    </body>
</html>

Process completed, Exit Code 0.
Execution time: 00.00.204
```
Il sorgente sopra riportato, produce questo risultato sul browser.

Ho usato l'istruzione **echo** per produrre del contenuto HTML all'interno della pagina.
I caratteri "**\n**" e "**\t**" sono detti **Caratteri di Escape**, che vi illustrerò nei prossimi capitoli.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 28/06/2007 - 1:14

Lessico

Introduciamo ora le basi del linguaggio.
PHP è un linguaggio sia "**Case Sensitive**" che "**Case Insensitive**".

Tutte i costrutti del linguaggio, le parole chiave, i nomi delle funzioni e delle classi sono**Case Insensitive**, di conseguenza le istruzioni che seguono sono equivalenti :

```
echo "ciao"; // stampa "ciao"
ECHO "ciao"; // stampa "ciao"
ecHO "ciao"; // stampa "ciao"
```
Al contrario, i nomi delle variabili sono**Case Sensitive**, di conseguenza due variabili con lo stesso nome ma con differenze di maiuscole e minuscole, vengono considerati due variabili distinte :

```
$x = 5;
$X = 10;

echo $x; // stampa "5"
```
Ogni volta che aprite una parentesi, PHP richiede la rispettiva chiusura.
Cicli ed istruzioni di controllo con una singola istruzione all'interno, non richiedono l'uso di parentesi graffe per delimitarne il frammento di codice, ma possono comunque essere messe.

Di seguito alcuni esempi :

```
if ($x == $y)
     echo "Sono uguali";
else
     echo "Sono diverse"; // corretto


if ($x == $y)
     echo "Sono ";
     echo "uguali";
else
     echo "Sono diverse"; // errore : sono necessarie le parentesi graffe


if ($x == $y)
{
     echo "Sono ";
```

```
     echo "uguali";
}
else
     echo "Sono diverse"; // corretto
```

Per semplici istruzioni, istruzioni di assegnazione e chiamate a funzioni, PHP richiede un carattere di**;** (punto e virgola) alla fine dell'istruzione, mentre per le condizioni, i tag del PHP e la fine di blocchi di codice (IF, FOR ...), il carattere di **;** deve essere omesso.

Di seguito alcuni esempi :

```
if ($x == $y) // corretto
     echo "Sono uguali"; // corretto

if ($x == $y); // errore
     echo "Sono uguali"; // corretto

echo "ciao" // errore
echo "ciao"; // corretto

<?php; // errore
?> // corretto

$x = 5; // corretto

listaImpiegati("gennaio") //errore
listaImpiegati("gennaio"); // corretto

if ($x == $y) { echo "Sono uguali"; } // corretto
```

Nella prossima pagina vedremo **Spazi** e **Commenti**

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 29/06/2007 - 3:37

Spazi e commenti

Spazi

In PHP è possibile andare a capo durante una stringa, nella lista di parametri di una funzione o di un array.
E' possibile inoltre andare a capo innumerevoli volte fra un'istruzione e l'altra senza per altro influenzare l'esecuzione del codice.

Di conseguenza, le linee di codice che seguono sono tutte valide :

```
echo "Hello
World!";

aggiungiImpiegato  (
                                "Mario" ,
                                "Rossi" ,
                                "Roma" ,
                                "01/04/1980"
                              );
```

Commenti

PHP ci mette inoltre a disposizione un utilissimo strumento : i**Commenti**.
I commenti servono per inserire righe di testo all'interno del codice, in formato testuale, che non verranno interpretate e considerate come codice, non influenzandone quindi l'esecuzione.

Ci sono 3 modi per inserire un commento in PHP :

- Commento a riga singola con doppio backslash//
- Commento a riga singola con cancelletto#
- Commento multiriga con backslash e asterisco/* */

Di seguito alcuni esempi :

```
$x = 5; // questo è un commento a riga singola
$y = 10; # anche questo è un commento a riga singola

echo "ciao"; /* commento multiriga su una riga singola */

/* Questo invece è un commento
esteso su due righe */
```

```
/* Infine questo è un commento annidato // perchè contiene due
tipi di commenti */
```
Nel prossimo capitolo vedremo come creare delle **Variabili**.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 30/06/2007 - 6:01

Variabili (Parte 1 di 2)

Le variabili sono dei contenitori che vengono usati per memorizzare tutti i tipi di dato che PHP ci mette a disposizione, e i tipi di dato che possiamo creare noi attraverso la programmazione ad oggetti che affronteremo più avanti.

I nomi delle variabili devono obbligatoriamente cominciare con il carattere del dollaro$, e possono contenere numeri e lettere sia maiuscole che minuscole, a patto che il primo carattere dopo il$ non sia un numero ma una lettera o un **underscore**.

L'unico simbolo ammesso è proprio l'**underscore** (trattino basso _).
Di seguito riporto alcuni esempi :

```
$variabile = 5; // corretto
$3variabile = "cinque"; // errore : inizia con un numero (3)
$variabile3 = "cinque"; // corretto
$_variabile5_ = 9.8; // corretto
$minnie&topolino = 0; // errore : simbolo non consentito
```
E' possibile memorizzare un valore in una variabile mediante l'operatore di assegnazione**"="**.
Se alla destra dell'operatore**=** c'è un'altra variabile invece di un valore costante, viene creata una copia del valore e inserita nella variabile a sinistra in questo modo :

```
$y = 5; // non vengono create copie, viene direttamente assegnato il valore
5 a $y
$x = $y; // viene creata una copia di $y e viene inserita in $x
```
E' possibile specificare di **non** creare una copia, ma di assegnare il valore per riferimento, anteponendo al nome della variabile, a destra dell'operatore **"="** di assegnazione, il simbolo della E commerciale (&).
Questo significa che se viene modificata la prima variabile, tutte le variabili assegnativi per riferimento punteranno al nuovo valore.

```
$x = 5;
echo $x; // stampa 5

$y = &$x;
echo $y; // stampa 5

$x = 8;
echo $y; // stampa 8
```

Le variabili possono assumere una visibilità (scope) diversa a seconda di dove vengano dichiarate o se vengono esplicitamente definite come **Globali** o **Statiche**.

Se dichiariamo una variabile all'interno di una funzione, ad esempio, essa non sarà visibile all'esterno di questa.
Se invece dichiariamo una variabile come **Globale** allora sarà accessibile in tutto il codice, anche se dichiarata in un file esterno, a patto che quest'ultimo ovviamente venga incluso.

Di seguito un esempio di visibilità con una funzione :

```
$x = 2;

function miaFunzione()
{
     $x = 5;
     echo $x;
} // stampa 5

echo $x; // stampa il valore 2
```

Non dichiarando la $x prima di "miaFunzione()", l'ultima istruzione non avrebbe stampato niente, dal momento che una variabile all'interno di "miaFunzione()", è visibile esclusivamente all'interno della funzione stessa.

Tags : Variabili Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 01/07/2007 - 8:24

Variabili (Parte 2 di 2)

Variabili statiche e globali

Vediamo ora un esempio di variabile **Globale** e un altro di variabile **Statica** :

```
/* Esempio di variabile globale */

$x = 100;
$y = $x + 5;

function miaFunzione()
{
     global $x;
     echo "x : $x - y : $y";
}

miaFunzione(); // stampa "x : 100 - y : " perchè non abbiamo specificato
anche $y come globale


/* Esempio di variabile statica */

function funzione()
{
     static $x = 0;
     $x++;
     echo "x : $x";
}

funzione(); // stampa 1
funzione(); // stampa 2
```

Nel primo esempio abbiamo visto un modo per accedere alle variabili che non ci sono visibli, perchè dichiarate all'esterno.
Un altro modo è attraverso l'array super globale **$GLOBALS** :

```
$x = 5;

function miaFunzione()
```

```
{ echo "x : " . $GLOBALS[x]; } // stampa 5
```
Nell'esempio della variabile **Statica**, invece, abbiamo visto che dichiarando una variabile in tal modo, PHP non la distrugge alla fine dell'esecuzione della funzione, ma rimane disponibile ad ogni chiamata della suddetta funzione fino al termine dell'intero script.

Funzioni standard per le nostre variabili

Per agire in modo sicuro sulle variabili, PHP ci mette a disposizione molte funzioni.
Illustrerò di seguito le 3 più importanti ed usate :

isset() - Restituisce **TRUE** se la variabile esiste o **FALSE** se non esiste.

```
echo isset($variabile); // stampa false
$variabile = 5;
echo isset($variabile); // stampa true
```
unset() - Distrugge la variabile specificata senza tornare alcun valore.

```
$variabile = 5;
echo $variabile; // stampa 5
unset($variabile);
echo $variabile; // non stampa nulla
```
empty() - Restituisce **FALSE** se la variabile non è vuota e diversa da zero.

```
echo empty($variabile); // stampa true
$variabile = 5;
echo empty($variabile); // stampa false
```


Guida a PHP 5

autore : Casula Francesco
Pubblicato il 02/07/2007 - 10:47

Tipi di dato

PHP non è un linguaggio fortemente tipizzato come il C++, ma è molto più flessibile come ad esempio un linguaggio di scripting, quali JavaScript, ActionScript ecc...

Questo significa che se noi assegniamo un valore intero ad una variabile, possiamo assegnargli subito dopo una stringa, o ancora un valore booleano, senza incorrere in alcun errore.

Il linguaggio ci fornisce diversi **Tipi di dato** molto utili.
Vediamo una lista di tutti i tipi di base per poi approfondirli singolarmente

- Interi - es. 10 / 50000 / -200 / 06333 / 0xFF
- Virgola mobile - es. 1.5 / 50.72 / 0.7e-5 / -100.3 / 61.2A14
- Stringhe - es. "stringa" / "ciao" / 'pippo' / "54746"
- Booleani - es. true / false
- Null - es. NULL
- Costanti - es. define("COSTANTE", 150); / COSTANTE
- Array - es. $array[0] / $podio["primo"] / $tabella[5]["cognome"]

Gli interi sono tutti i numeri non decimali come 1, 10, 3965, compresi fra **-2,147,483,648** e **2,147,483,647**.

I numeri a virgola mobile, invece, rappresentano i numeri decimali compresi fra **1.7E-308** e **1.7E+308**, il che significa che hanno una precisione di 15 cifre dopo la virgola.

Caratteri di escape

Le stringhe sono insiemi di caratteri racchiusi fra apici o doppi apici.
Di conseguenza "**minnie**" e '**paperino313**' sono da considerarsi stringhe.
Per far sì che le stringhe possano contenere, ad esempio, i caratteri speciali usati dalla sintassi del PHP, è necessario utilizzare i **Caratteri di escape**.

Di seguito un esempio :

```
$x = 5;
echo "Questo è il valore di $x : $x";
// stampa : Questo è il valore di 5 : 5
echo "Questo è il valore di \$x : $x";
// stampa : Questo è il valore di $x : 5
```
I caratteri di escape quindi, sono dei caratteri speciali preceduti da un Backslash\.
Di seguito una tabella di tutti i caratteri di escape che PHP ci fornisce :

Carattere di Escape	Rappresentazione del carattere
\"	Stampa i doppi apici "
\n	Va a capo e manda indietro il cursore **(newline)**
\r	Va a capo senza far tornare indietro il cursore **(Carriage return)**
\t	Carattere di tabulazione **(Tab)**
\\	Stampa un Backslash \
\$	Stampa il simbolo del Dollaro **$**
\{	Stampa la parentesi graffa **{**
\}	Stampa la parentesi graffa **}**
\[	Stampa la parentesi quadra **[**
\]	Stampa la parentesi quadra **]**
\0	Carattere ASCII che rappesenza i valori in base **ottale (8)**
\x0	Carattere ASCII che rappesenza i valori in base **esadecimale (16)**

I valori **Booleani** possono assumere solo due valori : **true** (ossia 1) o **false** (ossia 0).
Vengono usati spesso nelle strutture di controllo come la **IF** o come valori di ritorno delle funzioni.

```
$approvato = false;

if ($approvato)
     echo "Utente approvato!";
else
     echo "Utente non approvato!";

//stamperà : Utente non approvato!
```

Le variabili inoltre, possono assumere il valore NULL. La parola chiave non è **Case sensitive** e potete quindi scriverla come volete.
Il tipo di dato **NULL** serve a rappresentare una variabile vuota, senza valore.

```
$variabile = NULL;
if (empty($variabile))
     echo "Variabile vuota";
else
     echo "Variabile piena";

// stamperà : Variabile vuota
```

Nel prossimo capitolo vedremo le **Costanti**.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 03/07/2007 - 13:11

Costanti

Le **Costanti** sono un tipo di variabile che è possibile definire una volta sola.
Una volta che assegnate un valore ad una costante, essa avrà questo valore per tutta la durata dello script, senza la possibilità di modificarlo. Vediamo ora come è possibile definire una costante :

```
define("PI_GRECO", 3.14);

$x = PI_GRECO; // corretto
$x = pi_greco; // non corretto
```

I nomi delle costanti sono **Case Sensitive** di default, a meno che non venga esplicitamente impostato un terzo parametro col valore **false**, in questo modo :

```
define("PI_GRECO", 3.14, false);

$x = PI_GRECO; // corretto
$x = pi_greco; // anche questo ora è corretto
```

Nella prossima pagina affronteremo gli **Array**.

Tags : Php 5 Guida
Copia link a questo articolo

Home

Guide Tutorials Download Curiosità Mappa del Sito

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 04/07/2007 - 15:34

Array

Gli **Array** sono dei contenitori di variabili.
Possono contenere ogni tipo di variabile e possono essere multidimensionali.
Ci sono vari modi di dichiarare un Array. Vediamoli assieme :

```
$mioArray = array(1, 22.5, "ciao");

$altroArray[0] = 1;
$altroArray[1] = 22.5;
$altroArray[2] = "ciao";
```

I 2 array dell'esempio, **$mioArray** e **$altroArray** sono equivalenti.
In sostanza non c'è un limite specifico di quanti valori possa contenere un Array, in quanto dipende dalla memoria che la nostra applicazione ha a disposizione.

PHP inizia a contare gli elementi da zero, di conseguenza per accedere al primo elemento di un Array, è sufficiente specificare il nome dell'array preceduto dal dollaro $, e mettere il valore zero (0) fra parentesi quadre, in questo modo :

```
echo $mioArray[0];
```

Il valore fra le parentesi quadre è detto **Indice** dell'array, e non è consentito avere due indici uguali nello stesso array.

L'indice serve appunto per accedere a un determinato valore dell'array, e nel caso in cui si parli di array associativo, allora l'indice non sarà un numero intero ma bensì una stringa.

Anche qui vediamo i due modi per usare un array associativo :

```
$array1 = array("nome" => "John", "cognome" => "Doe");

$array2["nome"] = "Mario";
$array2["cognome"] = "Rossi";

echo $array1["nome"]; // stampa : John
echo $array2["cognome"]; // stampa : Rossi
```

L'array associativo si usa quindi allo stesso modo di un array classico, con la differenza che l'indice, essendo una stringa, non sarà più chiamato **Indice** ma bensì **Chiave** dell'array, e per accedere ai suoi valori servirà quindi una determinata stringa.

La differenza sostanziale fra questi due tipi di array, sta nel metodo per scorrerli, dal momento che difficilmente

quando useremo un array, sapremo a priori quanti valori conterrà e quali saranno esattamente gli indici o le chiavi.

Detto questo, vediamo alcune delle funzioni più importanti che PHP ci mette a disposizione per gestire gli array, e passiamole velocemente in rassegna :

- **count()** - Conta i valori presenti in un array
- **array_push()** - Aggiunge il valore specificato alla fine di un dato array
- **array_search()** - Cerca il valore specificato in un dato array
- **list()** - Assegna valori alle variabili specificate come se fossero un array
- **each()** - Restituisce la corrente coppia chiave/valore e incrementa il puntatore dell'array
- **reset()** - Reimposta il puntatore dell'array alla sua posizione iniziale

Vediamo ora un esempio di assegnazione e scorrimento di un array classico, mediante alcune delle funzioni sopracitate :

```
$mioarr = array();

array_push($mioarr, "minnie");
array_push($mioarr, "topolino");
array_push($mioarr, "pluto");

echo "L'array contiene " . count($mioarr) . " valori!\n\n"; // stampa :
L'array contiene 3 valori!

for ($i = 0; $i < count($mioarr); $i++)
     echo "Il valore " . ($i + 1) . " è " . $mioarr[$i] . "!\n";

/* la FOR stamperà :
Il valore 1 è minnie!
Il valore 2 è topolino!
Il valore 3 è pluto!
*/
```

Al contrario, ora vedremo come scorrere un array associativo in modo ordinato. Essendo associativo non possiamo incrementare la chiave come l'indice che è un numero intero, ma dovremo servirce di alcuni funzioni di PHP.

```
$arr = array();
$arr["nome"] = "John";
$arr["cognome"] = "Doe";
$arr["citta"] = "Sconosciuta";
$arr["nascita"] = "01/01/1980";

while (list($chiave, $valore) = each($arr))
     echo "$chiave : $valore<br />\n";
```

Abbiamo usato la funzione "**list()**" e la "**each()**" ottenendo questo risultato.

La funzione **list()**, non è esattamente una funzione ma piuttosto un costrutto del linguaggio, che nel nostro esempio, inizializza ed assegna le variabili **$chiave** e **$valore** passategli da **each()**.

each() la prima volta che viene chiamata, punta al primo valore dell'array e lo estrae passandolo a **list()**. Ogni volta che la funzione viene chiamata, ritorna il nuovo valore e incrementa di nuovo il puntatore al valore successivo, per prepararsi alla prossima chiamata. Quando l'array è finito, **each()** ritorna il valore **false** e il ciclo **while** si interrompe.

Per far tornare il puntatore di **each()** al valore iniziale in qualsiasi momento, è sufficiente chiamare la funzione **reset()**.

Tags : Array Contenitori Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 05/07/2007 - 17:57

Array Multidimensionali

Vediamo ora gli array multidimensionali e un metodo per gestirli.

Immaginate un array multidimensionale, come un array che contiene altri array.
Anche qui non c'è un limite preciso alla dimensione che può assumere un array multidimensionale, ma vediamo subito un esempio per farvi capire di che si tratta.

Immaginate di dover mettere in un unico array una serie di dati di questo genere :

ID	Nome	Cognome	Settore	Stipendio
1	Mario	Rossi	Design	1.250
2	Giacomo	Bernardini	Progettazione	1.550
3	Luca	Manfredi	Marketing	1.350

Un buon metodo per inserire questi dati in un array, è utilizzare un**indice** per la prima dimensione, ed una **chiave** per la seconda dimensione (N.B. : potete usare anche 3 dimensioni e vedere la base di dati come un cubo, ma non è questo il caso del nostro esempio).

```
$impiegati[1]["nome"] = "Mario";
$impiegati[1]["cognome"] = "Rossi";
$impiegati[1]["settore"] = "Design";
$impiegati[1]["stipendio"] = 1250;

$impiegati[2]["nome"] = "Giacomo";
$impiegati[2]["cognome"] = "Bernardini";
$impiegati[2]["settore"] = "Progettazione";
$impiegati[2]["stipendio"] = 1550;

$impiegati[3]["nome"] = "Luca";
$impiegati[3]["cognome"] = "Manfredi";
$impiegati[3]["settore"] = "Marketing";
$impiegati[3]["stipendio"] = 1350;

echo "<table width=\"600\" border=\"0\" cellspacing=\"5\" cellpadding=
\"1\">\n";

for ($i = 1; $i <= count($impiegati); $i++)
{
     echo "\t<tr>\r";

     while (list($chiave, $valore) = each($impiegati[$i]))
     {
          $chiave = mb_convert_case($chiave, MB_CASE_TITLE);
          // La funzione sopra converte la prima lettera di $chiave in
Maiuscola
          echo "\t\t<td>$chiave : $valore</td>\r";
     }

     echo "\t</tr>\r";
}

echo "</table>\n";
```

Il codice produce questo risultato.
I caratteri di escape "**\n**", "**\t**" e "**\r**", li ho usati per dare un indentazione e una formattazione ordinata al sorgente della pagina HTML che PHP produce, ottima abitudine per velocizzare le fasi di debug.

Infine vediamo la semplice ma efficace "**array_search**()".

```
$array = array();

array_push($array, "mela");
array_push($array, "fragola");
array_push($array, "limone");
array_push($array, "fragola");
array_push($array, "melone");
array_push($array, "fragola");
array_push($array, "limone");

echo array_search("fragola", $array) . "<br />\n";
echo array_search("limone", $array) . "<br />\n";
echo array_search("mela", $array) . "<br />\n";
echo array_search("anguria", $array) . "<br />\n";
```

Il codice produce come output "**1**", "**2**" e infine "**0**".
La funzione cerca la stringa e ritorna la posizione della**prima** occorrenza trovata, senza quindi tener conto di eventuali doppioni.
La quarta chiamata ad "**array_search**()", invece, non produce nessun output, in quanto la funzione torna il valore booleano "**false**" che **echo** non stampa.

Tags : Array Associativi Multidimensionali Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 06/07/2007 - 20:21

Stringhe (Parte 1 di 3)

In questo capitolo approfondiremo meglio le **stringhe** e il loro uso.
Potete racchiudere una stringa usando i doppi apici " o gli apici singoli '. Inoltre PHP 5 ci permette di gestire le stringhe come degli array, per mezzo delle parentesi graffe al posto delle quadre, in questo modo :

```
$stringa = "c";
$stringa{1} = 'i';
$stringa{2} = 'a';
echo $stringa . "o!"; // stampa : ciao!
```

Ci sono vari modi di stampare una stringa. Il più usato da me è **echo**, che essendo un costrutto del linguaggio e non una funzione, è possibile richiarmarlo senza parentesi, ma se preferite potete lasciarle.

Un altro modo è attraverso la funzione "**print()**", che ritorna il valore **true** quando riesce a stampare la stringa con successo, e il valore **false** se accade altrimenti.

Infine PHP ci fornisce un'altra funzione molto più avanzata : "**printf()**".
Questa funzione consente di inserire delle sottostringhe nella stringa, che consentono di dare una formattazione particolare a quest'ultima.

Le sottostringhe vengono identificate grazie al carattere percentuale/modulo "**%**", e vengono sostituite con i parametri che vengono passati dopo la stringa, a seconda del tipo di sottostringa specificata.

Di seguito un esempio chiarificatore :

```
printf('%.2f', 127.14457);
// stamperà : 127.14
```

Il carattere **%** indica dove deve essere stampato il secondo parametro passato a **printf()**, ossia il numero a virgola mobile 127.14457, e la sottostringa **.2f**, indica alla funzione che si tratta di un numero a virgola mobile (f = float) e che deve essere stampato con una precisione di 2 numeri dopo la virgola (.2).

Di seguito una tabella con tutti i caratteri per le sottostringhe, supportati da **printf()** :

Identificatore	Significato
%b	L'argomento è un numero intero e sarà visualizzato in base binaria
%c	L'argomento è un numero intero e sarà visualizzato come carattere ASCII corrispondente
%d	L'argomento è un numero intero e sarà visualizzato come un decimale con segno

%e	L'argomento sarà visualizzato con la notazione scientifica (es. 2.4e 3)
%u	L'argomento è un numero intero e sarà visualizzato come un decimale senza segno
%f	L'argomento sarà visualizzato come un numero a virgola mobile influenzato dalle impostazioni
%F	L'argomento sarà visualizzato come un numero a virgola mobile non influenzato dalle impostazioni
%o	L'argomento è un numero intero e sarà visualizzato in base ottale
%s	L'argomento è una stringa e sarà visualizzata come tale
%x	L'argomento è un numero intero e sarà visualizzato in base esadecimale (in minuscolo)
%X	L'argomento è un numero intero e sarà visualizzato in base esadecimale (in maiuscolo)

Per una documentazione completa sulla funzione **printf()**, vi rimando alla Documentazione Ufficiale da me segnalata nel capitolo **Configurare PSPad**.

Tags : Stringhe Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 07/07/2007 - 22:44

Stringhe (Parte 2 di 3)

Passeremo ora in rassegna le funzioni che useremo maggiormente quando costruiremo delle vere e proprie applicazioni, che dovranno manipolare dati prelevati da database o ancora peggio forniti dall'utente distratto.

Di seguito le funzioni usato per ripulire le stringhe dagli spazi inutili :

- **trim()** - elimina tutti gli spazi vuoti all'inizio e alla fine della stringa
- **ltrim()** - elimina tutti gli spazi vuoti all'inizio della stringa
- **rtrim()** - elimina tutti gli spazi vuoti alla fine della stringa

```
$stringa = "     ciao!  ";
echo trim($stringa); // stampa "ciao!" invece di "     ciao!  "
```

Ora vediamo le principali funzioni usate per rendere visibili i caratteri usati dal codice HTML come il segno< e il simbolo &.
Immaginiamo di dover prelevare un frammento di codice HTML dal database, e di doverlo inserire nella nostra pagina PHP e farlo visualizzare così come si presenta, facendo quindi in modo che il codice HTML non venga quindi interpretato come tale, ma bensì come un testo da stampare sulla pagina.

```
$stringa = "<a href=\"pagina.html\">Homepage</a>";
$conversione = htmlspecialchars($stringa);

echo $stringa; // stamperà "<a href="pagina.html">Homepage</a>"
echo "<br /><br />"; // va a capo 2 volte
echo $conversione; // stamperà "&lt;a
href=&quot;pagina.html&quot;&gt;Homepage&lt;/a&gt;"
```

La funzione **htmlspecialchars()** converte quindi i caratteri speciali in entità HTML.
In questo modo la stringa "**$stringa**" sarà interpretata come codice HTML e visualizzerà quindi il classico link, mentre la stringa "**$conversione**" produrrà in output una stringa normale, come potete vedere dal risultato ottenuto dal codice sopracitato.

La funzione "**html_entity_decode()**", invece, produce l'effetto contrario, stampando perciò "**html_entity_decode($conversione)**", ritorniamo ad ottenere la stringa iniziale.

Vediamo ora come confrontare due stringhe.
PHP ci mette a disposizione 2 operatori : **==** (Equal to) e **===** (Indentical to).

Esempio

```
$numero = 3;
```

```
$stringa = "3";

if ($numero == $stringa)
     echo "uguale"; // viene stampato

if ($numero === $stringa)
     echo "uguale"; // non viene stampato
```

Oltre questi 2 operatori, PHP ci fornisce svariate funzioni ... vediamo le 2 più importanti :

- **strcasecmp(stringa1, stringa2)** - Confronta 2 stringhe senza tener conto di maiuscole e minuscole. Restituisce un intero minore di zero se **stringa1** è minore di **stringa2**, oppure un intero maggiore di zero se **stringa1** è maggiore di **stringa2** e infine restituisce zero (0) se le due stringhe sono uguali.
- **strcmp(stringa1, stringa2)** - Confronta 2 stringhe consideranndo anche le differenze di maiuscole e minuscole. Restituisce un intero minore di zero se **stringa1** è minore di **stringa2**, oppure un intero maggiore di zero se **stringa1** è maggiore di **stringa2** e infine restituisce zero (0) se le due stringhe sono uguali.

```
$stringa1 = "ciao";
$stringa2 = "Zorro";

echo strcasecmp($stringa1, $stringa2); // stampa -23
echo strcmp($stringa1, $stringa2); // stampa 1
echo strcmp($stringa1, "ciao"); // stampa 0
```


Tags : Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 09/07/2007 - 1:07

Stringhe (Parte 3 di 3)

In questo ultimo capitolo sulle stringhe, vedremo altri metodo aggiuntivi per gestirle.
La funzione "**substr()**" ci permette di estrarre delle sottostringhe dalla stringa specificata, specificando un indice di partenza.

```
echo substr("Roma Caput Mundi", 5); // stampa : "Caput Mundi"
echo substr("Roma Caput Mundi", 5, 5); // stampa : "Caput"

// Il terzo parametro indica la lunghezza della sottostringa ed è
facoltativo
```

La funzione **substr_count()**, conta le occorrenze di una sottostringa nella stringa specificata.
str_replace() invece, ci permette di sostituire una parte di stringa con la sottostringa specificata.
Per ultima, **strlen()** che ci fornisce la lunghezza esatta della stringa specificata.
Vediamo alcuni esempi :

```
$stringa = "ciao ciao! ancora un ultimo saluto ... ciao";

echo str_replace("ciao", "bye", $stringa, $occorrenze); // stampa : bye
bye! ancora un ultimo saluto ... bye
echo $occorrenze; //stampa : 3

echo substr_count($stringa, "ciao"); // stampa : 3

echo strlen("ciao"); // stampa : 4
```

Continuiamo con altre funzioni, che ci permettono di trasformare le stringhe in array e viceversa.
Sono "**explode()**" e "**implode()**" che prendono come primo parametro un **separatore** e per secondo una **stringa** (explode) o un **array** (implode) su cui lavorare :

```
$stringa = "mela;banana;anguria;melone;kiwi";

$frutti = explode(";", $stringa);

for ($i = 0; $i < count($frutti); $i++)
     echo $frutti[$i] . "<br />\n";

echo implode(" - ", $frutti);
```

L'esempio produce il seguente output.

Infine vediamo le funzioni per eseguire delle ricerche nelle stringhe.

- **strpos($stringa, $ricerca [, $partenza])** - Restituisce un intero corrispondente alla posizione della prima occorrenza della stringa **$ricerca** nella stringa **$stringa**, oppure **false** se non viene trovata.
  Il terzo parametro è facoltativo ed indica da quale posizione deve iniziare la ricerca.
- **strstr($stringa, $ricerca)** - Trova la prima occorrenza della stringa **$ricerca** nella stringa **$stringa**, e restituisce la parte della stringa a partire dall'occorrenza di **$ricerca**.
- **stristr($stringa, $ricerca)** - Versione di "**strstr()**" insensibile alle maiuscole/minuscole.

```
$stringa = "posta@realizzazione-sito.info";

echo strpos($stringa, "@"); // stampa : 5
echo strstr($stringa, "@"); // stampa : @realizzazione-sito.info
```


Tags : Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 10/07/2007 - 3:31

Operatori (Parte 1 di 2)

Gli operatori sono usati principalmente per costruire espressioni.
Solitamente, essi vengono posizionati fra due valori per produrne un terzo, creando così un'espressione.

PHP ci mette a disposizione tre tipi di operatori :

- **Unari** - Richiedono un solo valore (es. $i++)
- **Binari** - Richiedono due valori (es. $x $y)
- **Terzari** - Esiste un solo operatore ternario e richiede tre valori (es. $x = ($y == 0) ? 5 : 10)

Vediamo ora la tabella degli operatori ordinati per precedenza in ordine decrescente :

Posizione	Operatore	Ambito	Descrizione
	new	new	Istanzia un nuovo oggetto
Destra	**[ ]**	Array	Array subscript
Destra	**!**	Operatore Logico	NOT Logico
Destra	**~**	Operatore sui bit	NOT sui bit
Destra	**++**	Incremento \ decremento	Operatore di incremento
Destra	**--**	Incremento \ decremento	Operatore di decremento
Destra	**(int), (double), (string), (array), (object)**	Tipi	Operatori di **Casting**
Destra	**@**	Errori	Inibisce gli errori
Sinistra	*	Operatore aritmetico	Moltiplicazione
Sinistra	/	Operatore aritmetico	Divisione
Sinistra	**%**	Operatore aritmetico	Modulo (resto divisione ecc...)

Sinistra	**+**	Operatore aritmetico	Addizione
Sinistra	**-**	Operatore aritmetico	Sottrazione
Sinistra	**.**	Operatore su stringa	Concatenazione stringhe
Sinistra	**<< ; >>**	Operatori sui bit	Scorrimento bit a sinistra e destra
	< ; <=	Operatori di confronto	Minore ; Minore uguale
	> ; >=	Operatori di confronto	Maggiore ; Maggiore uguale
	== ; ===	Operatori di confronto	Uguaglianza nel valore ; Uguaglianza nel tipo e nel valore
	!= ; <> ; !==	Operatori di confronto	Disuguaglianza ; Disuguaglianza ; Disuguaglianza nel tipo e nel valore
Sinistra	**&**	Operatore sui bit	AND sui bit o passaggio parametri/variabili per riferimento (es. $x = &$y)
Sinistra	**^**	Operatori sui bit	XOR sui bit
Sinistra	**|**	Operatore sui bit	OR sui bit
Sinistra	**&& ; ||**	Operatori Logici	AND Logico ; OR Logico
Sinistra	**?:**	Operatore Ternario	Operatore condizionale
Sinistra	**=**	Operatore d'assegnazione	Operatore d'assegnazione
Sinistra	**+= ; -= ; *= ; /= ; %= ; &= ; |= ; ^= ; ~= ; <<= ; >>=**	Operatori d'assegnazione	Operatori d'assegnazione con operazione
Sinistra	**.=**	Operatore su stringa	Assegnazione con concatenazione stringa
Sinistra	**and ; xor ; or**	Operatori Logici	AND Logico ; XOR Logico ; OR Logico
Sinistra	**,**	Diversi usi	Separatore liste

Vediamo ora un esempio sulla precedenza degli operatori, per capire anche come sfruttare al meglio questa tabella.
L'esempio consiste in una semplice espressione, con una addizione e una moltiplicazione. L'operatore*, come potete vedere nella tabella, precede l'operatore+, di conseguenza ha la precedenza nella seguente espressione :

```
echo 1 + 5 * 2; // stampa : 11 e non 12
echo (1 + 5) * 2; // stampa : 12
```

Di seguito altri esempi commentati sulla precedenza degli operatori :

```
$x = 5;
$y = 10;

echo $x > $y ? 2 * 4 : 10 / 2; // stampa : 5
echo $x < $y ? 2 * 4 : 10 / 2; // stampa : 8
```


Tags : Operatori Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 11/07/2007 - 5:54

Operatori (Parte 2 di 2)

Riporto altri esempi sull'utilizzo di vari operatori :

```
$x = $a + $b; // somma $a e $b e memorizza il valore in $x

$x = 5;
echo $x *= 14 % 6; // stampa : 10 (14 diviso 6 da 2 di resto, moltiplica 2
ad $x e lo memorizza in quest'ultima

echo $x++; // stampa : 10 (stampa la $x e poi la incrementa)
echo $x; // stampa : 11
echo ++$x; // stampa : 12 (prima incrementa $x e poi la stampa)
echo $x; // stampa : 12

$stringa = "Roma";
$stringa .= " Caput Mundi"; // $stringa diventa : "Roma Caput Mundi"
```

Vediamo ora a cosa serve l'operatore Ternario **?:** e come si usa.
Questo operatore ci consente di valutare una espressione **(condizione)**, e in caso venga soddisfatta viene eseguita una determinata operazione, altrimenti viene eseguita l'operazione alternativa.

Bisogna specificare prima la condizione seguita dal punto interrogativo **?**, poi la prima operazione che sarà eseguita nel caso in cui la condizione viene soddisfatta, a seguire i due punti :, e infine la seconda operazione che verrà eseguita se la condizione **NON** viene soddisfatta.

```
$stringa = "At vero eos et accusam et justo duo dolores et ea rebum.";

$messaggio = strlen($stringa) >= 100 ? "Stringa di 100 caratteri o più!" :
"Stringa con meno di 100 caratteri!";

echo $messaggio; // stampa : "Stringa con meno di 100 caratteri!"
```

Parliamo ora del **Casting.**
Il casting è un operazione il cui scopo è quello di convertire il tipo di un dato in un altro tipo.

E' possibile effettuare due tipi di casting : uno implicito impostato automaticamente da PHP, ed uno esplicito impostato da noi tramite gli operatori di **casting**. Vediamone due esempi.

```
echo "44 gatti " + 1; // stampa : 45 - Casting implicito
echo "44 gatti " . 1; // stampa : "44 gatti 1" - Casting implicito
echo 1.28 + 2; // stampa : 3.28 - Nessun casting
echo (int) 1.28 + 2; // stampa : 3 - Casting esplicito
```

Nella prima **echo** possiamo osservare un casting di tipo implicito, infatti PHP esegue un casting **(int)** sulla stringa "44 gatti" perchè viene seguita dall'operatore aritmetico **+**.

Nella seconda **echo** abbiamo ancora un casting di tipo implicito, ma stavolta del tipo **(string)** sul valore 1, perchè viene usato l'operatore di concatenazione stringhe.

Nella terza **echo** non viene effettuato nessun casting, ma serve a rendere più chiaro il casting esplicito della quarta **echo**, dove tramite l'operatore di casting **(int)**, viene detto a PHP di considerare il numero come intero, e non come numero a virgola mobile.

Prima di chiudere il capitolo, rivediamo gli **Operatori sui bit**.

Questi operatori sono particolari perchè convertono prima gli operandi dell'espressione nella loro rappresentazione binaria.
Ecco un esempio di come lavora l'operatore logico sui bit & :

```
// Operatore sui bit &

111101101 &
110111001
-------------
110101001
```

In questo caso quindi, per ogni binario viene effettuata la **AND** che fallisce quando i bit non sono entrambi 1.

Per capire meglio l'aritmetica binaria, potete usare le funzioni **bindec()** e **decbin()**, per riconvertire i numeri da binario a decimale e viceversa, e vi consiglio anche di dare un'occhiata al capitolo **"Operatori bitwise"** della Documentazione Ufficiale di PHP.net costantemente aggiornata.

Tags : Operatori Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 12/07/2007 - 8:17

Strutture di controllo (Parte 1 di 4)

PHP ci mette a disposizione diverse strutture di controllo, che possiamo dividere principalmente in due grandi categorie : le strutture condizionali e quelle iterative.

Le strutture condizionali, ci consentono di eseguire percorsi diversi in casi specifici in fase di esecuzione, mentre le strutture iterative, ci permettono di eseguire lo stesso pezzo di codice per un determinato numero di volte, dipendente dalla condizione specificata.

Vediamo ora le strutture di controllo **condizionali** :

- **IF e ELSE** - Struttura più comune presente in molti linguaggi
- **SWITCH** - Costrutto molto simile a **IF e ELSE**, utile per creare strutture condizionali molto lunghe

E' possibile inserire una semplice **IF**, specificando il pezzo di codice da eseguire nel caso in cui la condizione venga soddisfatta, fra parentesi graffe. Le parentesi non sono necessarie qualora il codice da eseguire consista in una singola riga.

Se volete specificare del codice da eseguire, nel caso in cui la condizione della **IF** fallisca, è sufficiente accodare a quest'ultima un blocco **ELSE**. Se due condizioni non sono sufficienti, allora potete inserire fra la **IF** ed un'eventuale **ELSE**, dei blocchi **ELSE IF**, che al contrario del blocco **ELSE**, richiedono una condizione precisa.

Le istruzioni nel blocco **ELSE** quindi, verranno eseguite solo se falliscono le condizioni della **IF** e di tutte le **ELSE IF** che precedono il blocco **ELSE**. E' anche possibile omettere i blocchi **ELSE** e **ELSE IF**, a patto che ci sia sempre e per primo uno o più blocchi **IF**.
Vediamo qualche esempio :

```
$x = 5;

if ($x == 5) // Se $x è uguale a 5
     $x = 1;
else // altrimenti
     $x = 0;

echo $x; // stampa : 1

$stringa = "5";

if ($stringa === 5) // Se è uguale a 5 ed è un numero intero
     $stringa += 5;
else if ($stringa === "5") // Se è uguale a 5 ed è una stringa
     $stringa .= " gatti";
else if ($stringa == "pippo") // Se il valore è uguale a "pippo"
{
     // Qui usiamo le graffe perchè la "else if" deve ospitare più di una
istruzione
     $stringa .= " e pluto ";
     $stringa .= strlen($stringa);
}
else
     $stringa = "errore";

echo $stringa; // Stampa : "5 gatti"
```

Ovviamente mettendo in fila solo blocchi **IF**, le condizioni verranno tutte controllate e se soddisfatte, saranno eseguiti tutti i blocchi di istruzioni corrispondenti, come nell'esempio seguente :

```
$a = 5;
$stringa = "x";

if ($a == 5)
     $stringa .= "5";
if ($a > 4)
     $stringa .= "x";

echo $stringa; // Stampa : "x5x"
```


Tags : Strutture Di Controllo If Else Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 13/07/2007 - 10:41

Strutture di controllo (Parte 2 di 4)

Altra struttura di controllo condizionale, è la **SWITCH**, che specificato un parametro iniziale, ci consente di confrontarlo con una serie di valori, mediante la parola chiave **case** seguita dai due punti.
Si può separare un blocco **case** da un altro grazie all'istruzione **break**.

Anche qui non abbiamo un limite al numero di condizioni, in quanto dipende dalla memoria che la nostra applicazione ha a disposizione.
Oltre alla parola chiave **case**, è possibile usare il comando **default**, per specificare un blocco di codice da eseguire nel caso in cui tutte le **case** falliscano (equivalente di **ELSE**).

Una particolarità della struttura **SWITCH**, consiste nel confronto che verrà sempre e solo effettuato per valore, come con l'operatore **==**, rendendola quindi inutile qualora sia necessario fare dei confronti di equivalenza (corrispondente operatore **===**), che tengano conto sia del valore che del tipo di dato.

E' possibile però specificare, nella condizione dei singoli **case**, di utilizzare una comparazione di equivalenza, inserendo variabile ed operatore nella condizione, come per una **IF** o per una **ELSE IF**.

Un altra particolarità che rende versatile la **SWITCH**, è il cosiddetto metodo a **cascata**, che ci consente di scrivere un unico blocco di codice da eseguire al verificarsi di più condizioni.

Vediamo un esempio che includa tutto ciò che è stato detto finora sulla **SWITCH** :

```
$stringa = "50";

switch ($stringa)
{
     case 50: echo "Uguale a 50<br />";
     case $stringa === "50": echo "Equivalente a 50<br />";
          break;
     case $stringa > 10: echo "Maggiore di 10<br />";
          break;
     default: echo "Diverso da 50<br />";
          break;
}
```

L'esempio produce questo risultato.
Nei primi due **case** possiamo osservare l'effetto a cascata. Essendo stato omesso il **break** nel primo **case**, il controllo continua e finisce nel secondo **case**, dove viene esplicitamente utilizzato un controllo sull'equivalenza di **$stringa**, e infine la **SWITCH** si conclude.

Il terzo **case** non produce la stringa "Maggiore di 10", in quanto viene soddisfatto già il secondo **case**, che contenendo un **break**, pone fine alla serie di condizioni.

Tags : Strutture Di Controllo Switch Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 14/07/2007 - 13:04

Strutture di controllo (Parte 3 di 4)

Parliamo ora delle Strutture di controllo **iterative** (o **Cicli**) che, come abbiamo detto prima, ci consentono di eseguire un determinato pezzo di codice un preciso numero di volte, o fino al fallimento di una data condizione.

PHP ci mette a disposizione 4 diversi tipi di Cicli :

- **FOR** - Prende tre parametri : un indice, una condizione e un incremento.
- **WHILE** - Prende un solo parametro : una condizione.
- **DO/WHILE** - Prende un solo parametro : una condizione.
- **FOREACH** - Prende due parametri : un array e un valore o chiave/valore.

Il ciclo **FOR** viene solitamente usato per gestire uno o più indici interi, spesso per scorrere uno o più array contemporaneamente ed effettuare particolari ricerche o ordinamenti in essi.

Non è necessario specificare tutti e 3 i parametri della **FOR**, ma omettendo la condizione si da vita ad un ciclo infinito, terminabile solo con un'istruzione **break** all'interno del ciclo.

Il primo parametro consiste nella dichiarazione ed inizializzazione di uno o più indici, o omettendolo in un punto e virgola.

Il secondo parametro invece è la condizione, che come per la **IF** può essere un'espressione più o meno articolata.

Infine il terzo parametro, deve essere l'incremento o decremento di uno o più indici, o anche questo come gli altri parametri, un semplice punto e virgola per la dichiarata omissione.

Vediamo subito qualche esempio di **FOR** :

```
for ($i = 0; $i < 10; $i++) echo $i; // Stampa : 0123456789

for ( ; ; )     echo "ciao"; // Stampa "ciao" all'infinito
```

Il ciclo **WHILE**, come già accennato, prende un solo parametro ossia una condizione, che farà eseguire il ciclo fino al suo fallimento.
Questo significa che possiamo riprodurre la **FOR** dell'esempio precedente, anteponendo al ciclo l'inizializzazione degli indici e mettendo a fine ciclo i vari incrementi/decrementi.

Il ciclo **DO/WHILE** invece, è una sorta di **WHILE** invertita, ossia esegue prima il codice all'interno e poi controlla la sua condizione. Vediamo qualche esempio :

```
$i = 0;

while ($i < 10)
{
     echo $i;
     $i++;
} // Stampa : 0123456789

$condizione = false;

$i = 0;

do {
     echo $i++;
} while ($condizione) // Stampa 0 e poi si ferma
```

Infine vediamo l'ultimo dei 4 cicli, la **FOREACH**.
Lo scopo principale di questo ciclo, è quello di eseguire un blocco di codice per ogni elemento contenuto nell'array specificato.

Ci sono due modi di usare una **FOREACH**, vediamone subito un esempio :

```
$array = array("uno", "due", "tre", "quattro", "cinque");

foreach ($array as $valore)
     echo $valore . "<br />";

foreach ($array as $chiave => $valore)
     echo "$chiave : $valore<br />";
```

La prima **FOREACH** produce questo risultato, mentre la seconda produce quest altro.

Il primo metodo viene solitamente usato nel caso di array normali, mentre il secondo per array associativi, ma potete usarli come preferite.

Tags : Strutture Di Controllo Cicli Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 15/07/2007 - 15:27

Strutture di controllo (Parte 4 di 4)

PHP ci fornisce altri due comandi per aumentare il controllo sui cicli :**break** e **continue**.

L'istruzione **break**, semplicemente interrompe il ciclo quando viene eseguita, mentre l'istruzione**continue**, obbliga il ciclo a saltare all'iterazione successiva senza completare le istruzioni presenti nell'iterazione corrente.

```
for ($i = 0; $i < 10; $i++)
{
     if ($i % 2)
     {
          echo "$i ... continue<br />";
          continue; // Istruzione CONTINUE
     }

     echo "$i ... non continue<br />";
}
```

Nell'esempio vediamo la prima **IF** controllare se l'indice **$i** è dispari, attraverso l'operatore modulo **%** (resto della divisione), e in tal caso richiama l'istruzione **continue** che fa saltare al ciclo tutte le istruzioni successive ad essa, procedendo con l'iterazione successiva.

Avremmo anche potuto costruire del codice equivalente, servendoci di un blocco**IF ELSE**, producendo il medesimo risultato.

Vediamo ora il codice senza l'istruzione **continue**, che produrrà invece sul browser questo risultato :

```
for ($i = 0; $i < 10; $i++)
{
     if ($i % 2)
          echo "$i ... continue<br />";

     echo "$i ... non continue<br />";
}
```

Vediamo ora un esempio veloce per illustrare meglio l'istruzione**break**.
Poniamo di voler interrompere il ciclo a metà della sua vita, impostando la preferenza attraverso una variabile esterna al ciclo.

```
$esterna = true;
$iterazioni = 10;
```

```
for ($i = 0; $i < $iterazioni; $i++)
{
     if ($esterna && ($iterazioni / 2) == $i)
          break;

     echo "$i<br />";
}
```
Il ciclo dell'esempio stamperà "**0 1 2 3 4**" e poi verrà interrotto dal **break**.

Vediamo infine le ultime strutture di controllo : **exit** e **die**.
Le due istruzioni in questione sono equivalenti, ed hanno lo scopo di interrompere l'intero script da qualsiasi posizione vengano chiamate, con un parametro aggiuntivo opzionale, che consiste in un messaggio da stampare sul browser prima della terminazione dello script.

Questi due comandi, non solo terminano lo script, ma interrompono l'intero flusso **HTML**, rendendole quindi particolarmente inappropriate per una gestione pulita degli errori, argomento che tratteremo più avanti in un capitolo interamente dedicato a questo.

Di seguito un esempio con l'istruzione **die** :

```
<html>
     <head>
          <title>Test</title>
     </head>

     <body>
          <h3>Test</h3>
<?php

     die("Errore irreversibile!");

?>
          <p>Alcuni diritti riservati - by pippo</p>
     </body>
</html>
```
L'esempio produce questo risultato. Il piè di pagina con i diritti d'autore non viene infatti stampato, in quanto **die** interrompe completamente il flusso dati (Sorgente pagina).

Tags : Strutture Di Controllo Continue Die
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 16/07/2007 - 17:51

Funzioni (Parte 1 di 4)

Definizione e dichiarazione di una funzione

Le **funzioni** sono un blocco di codice che assolvono un determinato compito, a cui viene dato un nome specifico ed unico.

Esse possono ricevere o meno dei valori in entrata chiamati **parametri**, per produrre e restituire o meno un singolo valore.

> Il loro principale compito è quello di rendere il codice non solo più leggibile, ma anche più pulito, elastico e flessibile.
> Questo è possibile proprio perchè, racchiudendo una porzione di codice all'interno delle nostre funzioni, sarà possibile richiamare quel codice senza doverlo riscrivere, e lo possiamo inoltre rendere adattabile a seconda dei parametri ricevuti in entrata.
>
> Un altro punto di forza si riscontra in fase di **Debug** (correzione degli errori), in quanto per il medesimo motivo sopracitato, è sufficiente correggere l'errore nella nostra funzione, per far sì che automaticamente abbia effetto ogni volta che viene usato il codice della funzione attraverso una chiamata ad essa.

Vediamo ora come è possibile dichiarare e definire una nuova funzione.
Innanzitutto è necessaria la parola chiave **function**, a seguire il nome che abbiamo scelto per la nostra funzione, inoltre una lista di parametri racchiusi fra parentesi tonde e separati da una virgola, e infine il corpo della funzione (codice) racchiuso fra parentesi graffe.

```
function nomefunzione($parametro1, $parametro2) // Da zero a N parametri
{
     // corpo della funzione
}
```

Restituzione di valori

Molto spesso torna utile che la nuova **funzione** restituisca un valore per fare in modo che questo venga letto ed utilizzato, come ad esempio il risultato di una operazione matematica o ancora il successo/fallimento di una determinata operazione.

Per restituire un valore, è sufficiente usare la parola chiave **return** seguita dal valore che deve essere restituito (variabili, costanti, espressioni booleane ...) :

```
function addizione($valore1, $valore2)
{ return $valore1 + $valore2; }

function inviaEmail($mittente, $destinatario, $oggetto, $messaggio)
{
     // codice per l'invio dell'email
     // return true; (Se l'email viene inviata con successo)
     // return false; (Se l'email non viene inviata con successo)
}
```

Visibilità delle funzioni

E' possibile richiamare le funzioni in ogni parte del codice, all'interno di altre funzioni, nel corpo principale dello script ecc ...
Per farlo è necessario scrivere il nome della funzione seguita dalle parentesi tonde. Se la funzione riceve dei parametri per cui non sono stati specificati dei valori di **default**, allora è obbligatorio inserirli all'interno delle parentesi tonde separandoli anche qui da una virgola e chiudendo la riga con un punto e virgola.

```
echo addizione(5, 4); // Stampa : 9
```

Come potete notare dall'esempio, è possibile utilizzare i valori ritornati da una funzione proprio come una variabile, stampandoli direttamente con echo, passandoli ad altre funzioni, assegnandoli a variabili e così via. Di seguito alcuni esempi :

```
function add($n1, $n2) { return intval($n1 + $n2); } // Somma i 2 valori e
ritorna il risultato come numero intero
function sub($n1, $n2) { return intval($n1 - $n2); } // Sottrae i 2 valori
e ritorna il risultato come numero intero

function pariDispari($num)
{ return (!($num % 2)) ? "pari" : "dispari"; }

$n = sub(17, 12);
echo sub(add(3, 4), $n); // Stampa : 2

$value = pariDispari(3) or die("E' richiesto un numero pari!");
echo "<br />" . $value; // Va a capo nell'output HTML e stampa : "dispari"
```

Le funzioni in **PHP** hanno visibilità globale, possono quindi essere richiamate in ogni parte del codice usando il nome della funzione che è **Case-Insensitive**, anche se vi consiglio di richiamare le funzioni con lo stesso identico nome con cui le avete dichiarate.

PHP fra l'altro, non supporta l'overloading delle funzioni come ad esempio il C++, questo significa che non potete ridefinire una funzione già dichiarata.

Tags : Php 5 Guida
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 17/07/2007 - 20:14

Funzioni (Parte 2 di 4)

Funzioni dentro funzioni

Tornando al discorso della visibilità delle funzioni, è possibile precludere l'utilizzo di alcune funzioni fino all'avvenuta esecuzione di un'altra determinata funzione, dichiarando all'interno di quest'ultima, le funzioni che dovranno essere visibili all'esecuzione di questa.

```
function func1()
{
     function func2()
     { echo "Sono la func2()!"; }
}

func1();
func2(); // Stampa : "Sono la func2()!"
```

Omettendo la chiamata alla funzione **func1()** si ottiene il seguente errore :
Fatal error: Call to undefined function func2() in **C:\AppServ\www\test.php** on line **X**.

Funzioni condizionali

Un altro modo per rendere disponibile una funzione solo in determinati casi, è quello di costruire una **funzione condizionale**, ossia una **funzione** dichiarata e definita all'interno di una struttura di controllo come la **IF** :

```
$creafunzione = true;

if ($creafunzione)
{
     function funzione()
     { return "ciao"; }
}

if ($creafunzione) funzione(); // Stampa : "ciao" nel caso in cui
$creafunzione sia true
```

In questi casi è necessario che la dichiarazione della funzione condizionale, appaia nello script prima di una qualunque eventuale chiamata, prerequisito non necessario per le funzioni normali.

Parametri di default

Approfondiremo ora il discorso dei parametri (o argomenti) che le nostre funzioni prenderanno in input.
E' possibile specificare un valore di **default**, che i parametri della nostra funzione dovranno assumere qualora non venissero specificati.

Per farlo è sufficiente assegnare un valore al parametro durante la dichiarazione della funzione, mediante l'operatore di assegnazione **=**.
Fate attenzione però all'ordine con cui disponete i parametri, avendo cura di lasciare per ultimi tutti quelli per cui avete previsto un valore di **default**.

```
function funzione($par1, $par2, $par3 = "par3", $par4 = "par4")
{
     static $call = 0;

     echo "Chiamata : " . ++$call . "\n<br />\n";

     echo $par1 . "<br />";
     echo $par2 . "<br />";
     echo $par3 . "<br />";
     echo $par4 . "<br /><br />";
}

funzione("pippo", "pluto", "paperino");
funzione("minnie", "topolino");
funzione("crono", "athena", "poseidone", "zeus");
funzione("gastone");
```

La nostra **funzione** nell'esempio soprastante, prevede quattro parametri, due dei quali hanno un valore di **default**.
Nella prima chiamata a **funzione()** risulta mancante il quarto parametro, la funzione lo sostituisce quindi con la stringa di default "par4".
Nella seconda chiamata mancano il terzo e il quarto parametro e vengono rispettivamente rimpiazzati coi valori "par3" e "par4".
Nella terza chiamata a **funzione()** invece, vengono specificati tutti e quattro i parametri previsti.
Infine nella quarta ed ultima chiamata viene generato un **"Warning"** (avvertenza) che segnala la mancanza del secondo parametro, producendo in output solo i parametri uno, tre e quattro.

Questo è il risultato.

Tags : Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 18/07/2007 - 22:37

Funzioni (Parte 3 di 4)

Passaggio di parametri per riferimento

Di default PHP passa i parametri alle funzioni **per valore**, ossia ogni volta che una funzione riceve un parametro, riceve in realtà una copia di esso. Vedremo ora come è possibile invece passare un **riferimento** alla variabile che contiene il suddetto valore.

Questo metodo si può applicare anche ai **valori restituiti** dalla funzione, e può tornare utile per svariati motivi come ad esempio il risparmio di memoria o altre metodologie di programmazione più sofisticate che non sono argomento di questa guida.

```
function funzione(&$stringa, $carattere, $ripetizioni)
{
     $tmp = "";

     for ($i = 0; $i < intval($ripetizioni); $i++)
         $tmp .= $carattere{0};

     $tmp .= $stringa;

    for ($i = 0; $i < intval($ripetizioni); $i++)
         $tmp .= $carattere{0};

     $stringa = $tmp;
}

$stringa = "Argomenti per riferimento";
funzione($stringa, "*", 10);
echo $stringa; // Stampa : **********Argomenti per riferimento**********
```

Nell'esempio soprastante, **funzione()** prende il parametro $stringa per riferimento, grazie alla E commerciale (&) anteposta al nome del parametro, permettendogli di modificare **direttamente** $stringa, senza necessità di restituire il valore prodotto.

Restituzione di valori per riferimento

E' possibile inoltre far si che una funzione restituisca non una copia di un valore ma ancora un riferimento, anteponendo stavolta la E commerciale (&) al nome della funzione, e avendo cura di usare l'operatore di assegnazione per riferimento =& in questo modo :

```
function &funzione()
{
     static $contatore = 0;
     return ++$contatore;
}

echo funzione(); // Stampa : 1
echo funzione(); // Stampa : 2
echo funzione(); // Stampa : 3

$value =& funzione();
$value = 0;

echo funzione(); // Stampa : 1
```

Questo metodo torna spesso utile in quanto, le variabili dichiarate all'interno della funzione, hanno una visibilità confinata alla sola funzione, rendendole quindi inaccessibili all'esterno.

Variabili statiche

Normalmente le variabili dichiarate nelle funzioni, vengono create ad ogni chiamata e distrutte alla fine di ogni esecuzione della funzione, ad eccezione delle variabili **statiche**.

> Anteponendo la parola chiave **static** al nome della variabile, si indica a **PHP** di non distruggere la variabile ogni volta che termina una chiamata alla funzione, ma bensì di renderla sempre disponibile fino alla terminazione dell'intero script.
>
> La visibilità delle variabili statiche resta comunque limitata alla funzione, non accessibile quindi a livello globale a meno di restituirla per riferimento come nell'esempio precedente.
>
> Le funzioni possono restituire solo un valore, ma vi capiterà senz'altro di costruire applicazioni con funzioni che dovranno restituire risultati più elaborati, composti da insiemi più complessi di valori.

In questi casi sarà buona cosa progettare le vostre funzioni per organizzare le collezioni di valori da restituire in **Array** o **Oggetti** in questo modo :

```
function ribalta($par1, $par2, $par3)
{ return array($par3, $par2, $par1); }

$array = ribalta("uno", "due", "tre");

for ($i = 0; $i < count($array); $i++)
     echo $array[$i] . " ";
```

L'esempio produce questo risultato.

Tags : Funzioni Static Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 20/07/2007 - 1:01

Funzioni (Parte 4 di 4)

Numero di argomenti variabile

Vi potrà capitare di progettare una funzione, senza sapere effettivamente quanti parametri prenderà in input, non potendo quindi ricorrere allo stratagemma dei valori di default descritto precedentemente.

PHP ci mette a disposizione tre **funzioni** per far fronte a questo tipo di esigenza :

- **func_num_args()** - Restituisce un numero intero corrispondente al numero di parametri ricevuto dalla funzione.
- **func_get_arg()** - Restituisce il valore che risiede all'indice specificato nell'array dei parametri.
- **func_get_args()** - Restituisce un array contenente tutti i parametri ricevuti in ingresso.

```
function funzione()
{
     $num = func_num_args();
     $parametri = func_get_args();

     $tmp = ($num > 0) ? func_get_arg(0) : 0;

     for ($i = 1; $i < $num; $i++)
          $tmp += $parametri[$i];

     return $tmp;
}

echo funzione(5, 3, 44, 12); // Stampa : 64
```

Funzioni Variabili

Vedremo ora il concetto delle **funzioni variabili**, che ci consentirà di chiamare più funzioni usando una sola variabile come riferimento, rendendo il nostro codice non solo più flessibile e di facile modifica, ma permettendoci inoltre di chiamare funzioni di cui non conosciamo a priori il nome.

Il funzionamento è piuttosto semplice, è sufficiente assegnare il nome della funzione che si intende chiamare ad una variabile, ed usare quest'ultima per effettuare la chiamata :

```
$language = "italiano";

function ilSaluto()
```

```
{ return "ciao"; }

function itGreets()
{ return "hello"; }

function ilSalue()
{ return "bonjour "; }

function error()
{ return "error"; }

switch ($language)
{
     case "italiano" : $funzione = "ilSaluto"; break;
     case "english" : $funzione = "itGreets"; break;
     case "french" : $funzione = "ilSalue"; break;
     default : $funzione = "error";
}

echo $funzione(); // Stampa : "ciao"
```

L'esempio non rispecchia un caso pratico, ma vuole solo mostrare la sintassi per un corretto utilizzo delle **funzioni variabili**.
Ricordatevi che con questo metodo non potrete effettuare chiamate ai costrutti del linguaggio come **echo**.

Conclusione

Finiamo il capitolo sulle **funzioni**, con la segnalazione del metodo **function_exists()**, che ci consentirà di verificare in fase di runtime, se una funzione è stata dichiarata e quindi esiste :

```
function funzione() { return "ciao"; }

if (function_exists("funzione")) echo "funzione esiste";
else echo "funzione non esiste";
// Verrà stampato "funzione esiste"
```


Tags : Php 5 Guida
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 21/07/2007 - 3:24

Librerie e file esterni (Parte 1 di 2)

Costruendo delle vere applicazioni web in **PHP**, vi accorgerete che sarà impossibile sviluppare del codice ordinato e flessibile senza suddividere l'applicazione in più file o librerie.

Vedremo ora come è possibile creare una libreria esterna all'applicazione includendola in quest'ultima.
Per farlo è necessario creare un nuovo file, dove sarà possibile inserire variabili, funzioni, oggetti e tutto il resto come avviene in una normale pagina **PHP**.

> Solitamente quando si crea un nuovo file di libreria, gli si assegna una estensione .inc che sta per **include**, ma in questo modo se non viene configurato correttamente il server, il codice diventa leggibile dall'esterno, creando problemi di sicurezza.
>
> Vi potrà capitare di produrre del codice che non installerete voi, perciò vi consiglio in ogni caso, di dare l'estensione .php anche ai file di libreria, in modo che vengano sempre interpretati come tali dal server rendendo illegibile il codice dal lato client.

Include & Require

PHP ci mette a disposizione quattro costrutti del linguaggio, non richiamabili quindi con il metodo delle **variabili funzione** :

- **include()** - Include il file specificato e solleva un **Warning** se non viene trovato.
- **require()** - Include il file specificato e solleva un **Fatal error** se non viene trovato interrompendo l'esecuzione dello script.
- **include_once()** - Include il file specificato durante l'esecuzione dello script e solleva un **Warning** se non viene trovato.
  Se il file è stato già incluso, **include_once** non lo includerà nuovamente.
- **require_once()** - Include il file specificato durante l'esecuzione e solleva un **Fatal error** se non viene trovato interrompendo l'esecuzione dello script. Se il file è stato già incluso, **require_once** non lo includerà nuovamente.

L'unica differenza fra i metodi **include** e **require**, è la gestione degli errori.

Include infatti solleva un **Warning** che non interrompe l'esecuzione dello script, mentre **require** produce un **Fatal error**, non esitate quindi ad usare **require** se il file incluso è di vitale importanza per la corretta esecuzione dello script.

Esempio di file esterno

Creeremo ora un nuovo file con estensione .php, per contenere alcune variabili e funzioni che useremo nell'applicazione principale, e che chiameremo **"libreria.php"** (**N.B. :** fate attenzione all'estensione del file, se siete sotto Windows assicuratevi di avere disattivato l'opzione **"Nascondi le estensioni per i tipi di file conosciuti"** sotto **"Strumenti"** -> **"Opzioni cartella..."** -> **"Visualizzazione"**).

```
<?php
	// libreria.php

	$settimana = array();
	array_push($settimana, "Domenica");
	array_push($settimana, "Lunedì");
	array_push($settimana, "Martedì");
	array_push($settimana, "Mercoledì");
	array_push($settimana, "Giovedì");
	array_push($settimana, "Venerdì");
	array_push($settimana, "Sabato");

	function stampaData()
	{
		global $settimana;

		$data = "Oggi è " . $settimana[date("w" ,time())];
		$data .= " " . date("d/m/Y - G:i", time());

		return $data;
	}
?>
```

Ora includiamo la nostra **libreria** tramite la direttiva **require_once** per richiamare la nuova funzione **"stampaData()"** :

```
<?php require_once("libreria.php"); ?>
<html>
	<head>
		<title>Test</title>
	</head>

	<body>
	<?php echo stampaData(); ?>
	</body>
</html>
```

Questo è il risultato ottenuto.

Tags : Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 22/07/2007 - 5:47

Librerie e file esterni (Parte 2 di 2)

Ci sono due sintassi corrette per utilizzare questi costrutti :

- require_once("libreria.php");
- require_once "libreria.php";

Potete usarli all'interno dei blocchi condizionali come **IF ELSE**, con l'accorgimento di racchiuderli sempre fra parentesi graffe, anche quando c'è solo un'istruzione, o potreste avere effetti indesiderati :

```
// Esempio corretto
if ($condizione)
{ include "libreria1.php"; }
else
{ include "libreria2.php"; }
```

Inoltre, essendo dei costrutti speciali che non richiedono parentesi tonde, bisogna prestare particolare attenzione quando si effettuano dei controlli sui valori che restituiscono, avendo cura di usare la seguente sintassi :

```
// Non corretto
if (include('libreria.php') == 'success')
{ echo 'OK'; }

// Corretto
if ((include 'libreria.php') == 'success')
{ echo 'OK'; }
```

Come si intuisce dall'esempio precedente, è possibile gestire i valori restituiti da **include** e **require**, semplicemente utilizzando il costrutto **return** nel file che si desidera includere.
Per capire meglio riporto direttamente l'esempio della Documentazione Ufficiale di **PHP** :

```
<?php
     // file "return.php"

     $var = 'PHP';
     return $var;
?>

<?php
     // file "noreturn.php"

     $var = 'PHP';
?>
```

```
<?php
     // file "testreturns.php"

     $temp = include 'return.php';
     echo $temp; // Stampa : 'PHP'

     $temp = include 'noreturn.php';
     echo $temp; // Stampa : 1
?>
```

Se includete un file all'interno di una funzione, tutto il codice all'interno del file sarà come se fosse stato dichiarato e definito all'interno di questa funzione, di conseguenza le variabili non saranno accesibili dall'esterno a meno di non essere dichiarate come globali.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 23/07/2007 - 8:11

Oggetti (Parte 1 di 2)

Introdurrò ora uno dei concetti più importanti di **PHP 5** : gli **Oggetti**.

La programmazione orientata agli **oggetti** è alla base di ogni applicazione solida e potente, e ci consentirà di scrivere del codice estremamente flessibile ed elastico, facendoci risparmiare talvolta anche ore ed ore di lavoro.

Possiamo pensare ad un **oggetto** come ad un tipo di dato più complesso e personalizzato, non esistente fra i tipi tradizionali di **PHP**, ma bensì ideato e creato da noi.

Lo scopo è quello di creare un tipo di dato più complesso, per soddisfare richieste che potremo comunque adempiere con i tipi tradizionali, ma con molta meno difficoltà e tempo.

Gli oggetti sono formati principalmente da **attributi** e **metodi**.

Gli **attributi** sono delle variabili proprietarie dell'oggetto, semplici o complesse, quindi anche **Array** o **Oggetti**.

I **metodi** invece, sono delle funzioni proprietarie dell'oggetto, e fra questi ce ne sono due molto importanti : i **Costruttori** e i **Distruttori**.

Se non definirete un **costruttore** e un **distruttore** per la vostra **classe** (oggetto), PHP li rimpiazzerà con dei metodi propri di default, in quanto questi due metodi sono essenziali per il corretto fuzionamento della classe.

Il **costruttore** verrà chiamato automaticamente da PHP, ogni volta che verrà creato un oggetto (istanza), mentre il **distruttore** sarà chiamato quando l'oggetto verrà distrutto, ossia quando non esiste più alcun riferimento all'oggetto oppure alla fine dello script.

Per dichiarare un costruttore, è sufficiente creare una funzione all'interno della classe che abbia lo stesso nome di quest'ultima, oppure potete usare la parola chiave **__construct()**.

Nell'esempio che segue creeremo un nuovo tipo di dato, l'oggetto "Persona", con gli attributi "nome", "cognome" e "data di nascita" e il metodo "stampaPersona()" per visualizzare i dati sul browser. Per creare la nuova classe utilizzeremo il costrutto del linguaggio **class** :

Persona.php

```
<?php

     class Persona
     {
```

```
        public $nome = ""; // attributo
        public $cognome = ""; // attributo
        public $datanascita = ""; // attributo

        public function Persona($n_nome, $n_cognome, $n_data) //
costruttore
        {
             $this->nome = $n_nome;
             $this->cognome = $n_cognome;
             $this->datanascita = $n_data;
        }

        public function stampaPersona() // metodo
        {
             echo "Nome : " . $this->nome . "<br />\n";
             echo "Cognome : " . $this->cognome . "<br />\n";
             echo "Data di nascita : " . $this->datanascita . "<br />\n";
        }
    }

?>
```

Come potete notare il nome del file è uguale a quello della classe, così come lo è il nome del costruttore.
Il costruttore prende in input tre parametri ($n_nome, $n_cognome, $n_data) che andrà a memorizzare rispettivamente nei propri attributi ($nome, $cognome, $datanascita), a cui accederà tramite la parola chiave **this**.

Con **this** l'oggetto può richiamare i suoi attributi e metodi, in quanto **this** indica l'oggetto stesso.
Subito dopo **this** segue l'operatore di selezione **->** che punta ad un determinato attributo o metodo alla sua destra, appartenente all'oggetto alla sua sinistra (this).

Ricordate sempre che in questi casi, il simbolo del dollaro **$**, va solo su **this** e non sul nome dell'attributo/metodo, a meno che non si stia usando il metodo delle variabili funzione, di cui discuteremo la variante per oggetti nei prossimi capitoli.

Infine troviamo il metodo **stampaPersona()** che semplicemente stampa gli attributi dell'oggetto tramite il costrutto **echo**.

La parola chiave **public** sarà illustrata nel prossimo capitolo assieme a **protected** e **private**.

Tags : Oggetti Classi This Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 24/07/2007 - 10:34

Oggetti (Parte 2 di 2)

Ora vedremo come creare ed utilizzare un'istanza dell'oggetto **"Persona"** dichiarato nella pagina precedente.

test.php

```
<?php require_once("Persona.php"); ?>
<html>
	<head>
		<title>Test</title>
	</head>

	<body>
	<?php

		$utente = new Persona("Mario", "Rossi", "10-01-1980");
		$utente->stampaPersona();

	?>
	</body>
</html>
```

Il codice soprastante produce questo risultato.
Per creare una nuova istanza della classe **"Persona"**, abbiamo usato la parola chiave **new** seguita dal costruttore della classe con i rispettivi parametri.
A seguire richiamiamo il metodo **"stampaPersona()"** con l'operatore di selezione **->** descritto in precedenza.

Osservando la classe nell'esempio della pagina precedente **"Persona.php"**, possiamo notare che tutti gli attributi, il costruttore e il metodo, sono dichiarati con la parola chiave **public**.

Questo ci consente di usarli e richiamarli liberamente nello script attraverso un'istanza della suddetta classe.
La parola chiave **public** è un modificatore di accesso, che serve a stabilire che tipo di restrizioni si devono avere lavorando su quel dato.

Public, Protected e Private

Di seguito una breve descrizione di **public** e le sue varianti **protected** e **private** :

- **public** - I membri (attributi o metodi) della classe dichiarati **public**, possono essere utilizzati sia all'interno che all'esterno della classe madre e di quelle derivate da essa (**ereditarietà**) (**$this->membro** oppure **$oggetto->membro**)
- **protected** - I membri dichiarati **protected**, possono essere utilizzati solo all'interno delle classi madri e derivate (**$this->membro**)
- **private** - I membri dichiarati **private**, possono essere utilizzati solo all'interno della classe madre (**$this->membro**)

Tornando alla classe **"Persona"**, poichè abbiamo dichiarato gli attributi come **public**, il linguaggio ci consente di agire direttamente su di essi, senza dover richiamare il costruttore ogni volta che dobbiamo modificare un attributo dell'istanza (**$utente**) :

```
$utente = new Persona("John", "Doe", "1-1-1970");

$utente->nome = "Santiago";
$utente->cognome = "Arnavisca";

$utente->stampaPersona(); // output = Nome : Santiago / Cognome :
Arnavisca / Data di nascita : 1-1-1970
```

Se invece avessimo usato la restrizione **private** per gli attributi della classe **"Persona"**, avremmo ricevuto dal server il seguente errore sollevato dalla riga $utente->nome = "Santiago";
Fatal error: Cannot access private property Persona::$nome in **C:\AppServ\www\test\test.php**

Avremmo ottenuto un errore equivalente anche utilizzando il modificatore di accesso **protected**, in quanto **protected** stabilisce che l'attributo o metodo così dichiarato, può essere utilizzato solo all'interno della classe, ma a differenza di **private**, ne consente l'utilizzo anche nelle classi derivate, argomento che tratteremo più avanti nel capitolo della **Ereditarietà**.

Tags : Oggetti Classi Public Private Protected Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 25/07/2007 - 12:57

Gli oggetti e i Membri statici

Prima di introdurre il concetto dei **membri statici**, è necessario sapere che per utilizzare un membro di una **classe**, un **attributo** o un **metodo**, è obbligatorio creare un'istanza di tale classe attraverso la parola chiave **new**, come illustrato nel capitolo precedente.

```
$utente = new Persona("John", "Doe", "1-1-1970");
```

In questo modo, ogni istanza della classe avrà la propria copia di ogni attributo e ogni metodo.

Talvolta può risultare utile voler rendere accessibile un membro di una classe senza doverne inizializzare un'istanza, e questo è appunto possibile dichiarando il membro come **statico**.

Per capire meglio, prendiamo un esame un'ipotetica classe **"Colore"**, che avrà una serie di colori di base che sarà utile rendere accessibili sempre, anche quando effettivamente non ci sarà utile creare un'istanza della suddetta classe.

Colore.php

```
<?php

     class Colore
     {
          static $rosso = "#FF0000";
          static $verde = "#00FF00";
          static $blu = "#0000FF";

          /*
               ... altri attributi
          */

          public function Colore()
          { /* codice costruttore */ }

          /*
               ... altri metodi
          */

          static public function stampaColore($colore)
          {
               echo "<font color=\"" . Colore::${$colore} . "\">Il valore
esadecimale del colore $colore è : ";
                    echo Colore::${$colore} . "</font><br />\n";
               }
          }

?>
```

Ora che abbiamo dichiarato la nostra nuova classe **Colore**, vediamo come è possibile utilizzare i suoi 3 attributi statici.
E' sufficiente specificare il nome della classe seguito dai doppi due punti e il nome del membro :

test.php

```
<?php

     require_once("Colore.php");

     echo "<font color=\"" . Colore::$rosso . "\">Il valore esadecimale del
colore rosso è : ";
     echo Colore::$rosso . "</font><br />\n";

     echo "<font color=\"" . Colore::$verde . "\">Il valore esadecimale del
colore verde è : ";
     echo Colore::$verde . "</font><br />\n";

     echo "<font color=\"" . Colore::$blu . "\">Il valore esadecimale del
colore blu è : ";
     echo Colore::$blu . "</font><br />\n";

?>
```

Richiamando la pagina **test.php**, otterremo questo risultato.

Se in un secondo momento decidiamo che per il nostro sito è più adatto un rosso scuro, sarà sufficiente assegnarvi tale valore all'inizio dello script, rendendo così effettiva la modifica in tutto il codice, senza peraltro modificare la classe che potrebbe essere condivisa con altre applicazioni anche esterne al server.

Nell'esempio che segue, stavolta utilizzeremo il metodo statico **"stampaColore()"** per visualizzare i colori sul browser.

test.php

```
<?php

     require_once("Colore.php");

     Colore::stampaColore("rosso");

     Colore::$rosso = "#C11C1C"; // Rosso più scuro

     Colore::stampaColore("rosso");

?>
```

L'esempio soprastante genera questo risultato.

Per accedere ai membri statici dall'interno della classe,**$this** non è adeguato e genera il seguente errore :
Fatal error: Using $this when not in object context in **C:\AppServ\www\test\Colore.php**

Il modo corretto per accedervi dall'interno è mediante il costrutto**self** :

```
<?php

     class Colore
     {
          static $rosso = "#FF0000";

          // ...

          static public function stampaColore($colore)
          {
               self::$rosso = "#FF0000";

               echo "<font color=\"" . Colore::${$colore} . "\">Il valore
esadecimale del colore $colore è : ";
               echo Colore::${$colore} . "</font><br />\n";
          }
     }

?>
```

Concludo il capitolo con una nota per chi è già avvezzo con il concetto dell'ereditarietà.
Se accedete a un membro statico della classe madre all'interno della figlia, ricordatevi di sostituire **self** con **parent**.

Tags : Oggetti Membri Statici Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 26/07/2007 - 15:21

Oggetti e Costanti

PHP 5 ci consente di definire delle costanti all'interno delle classi.
L'utilizzo è simile alle **costanti** globali descritte nel nono capitolo di questa guida.

Potete pensare alle **costanti** delle classi, come a degli attributi statici che non possono però essere modificati una volta dichiarati e definiti.

Prendiamo nuovamente in esempio la vecchia classe **"Colore"** esaminandone un utilizzo con le costanti :

Colore.php

```
<?php

     class Colore
     {
          const ROSSO = "#FF0000";
          const VERDE = "#00FF00";
          const BLU = "#0000FF";

          static public function stampaRosso()
          {
               echo "<font color=\"" . self::ROSSO . "\">Il valore
esadecimale del colore rosso è : ";
               echo self::ROSSO . "</font><br />\n";
          }
     }

?>
```

Come con le costanti globali è necessario omettere il simbolo del dollaro**$** durante la dichiarazione.
I nomi delle costanti in PHP 5 sono sempre Case Sensitive, ed è buona norma scriverle tutte in maiuscolo per distinguerle immediatamente come costanti, anche se non è obbligatorio.

test.php

```
<?php

     require_once("Colore.php");
```

```
     echo Colore::ROSSO . "<br />\n";
     Colore::stampaRosso();

?>
```
test.php produce questo risultato.

Tags : Oggetti Costanti Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 27/07/2007 - 17:44

Funzioni variabili applicate agli Oggetti

Nell'ultima parte del capitolo sulle **Funzioni**, abbiamo parlato delle funzioni variabili.

Vedremo ora come è possibile applicare il medesimo concetto ai metodi di una**classe**.

```
<?php

     class OggettoInventato
     {
          function metodo1()
          { echo "Ciao sono il primo metodo!<br />\n"; }

          function metodo2()
          { echo "Ciao sono il secondo metodo!<br />\n"; }

          function metodo3()
          { echo "Ciao sono il terzo metodo!<br />\n"; }
     }

     $obj = new OggettoInventato();

     $num = 3;
     $prefisso = "metodo";

     for ($i = 1; $i <= $num; $i++)
     {
          $funzione = $prefisso . $i;
          $obj->$funzione();
     }

?>
```
Il codice soprastante produce questo risultato.

Per richiamare i metodi con le variabili dall'interno della classe, è sufficiente sostituire **$obj** con **$this.**

```
<?php

     class OggettoInventato
     {
```

```
        function metodo1()
        { echo "Ciao sono il primo metodo!<br />\n"; }

        function metodo2()
        { echo "Ciao sono il secondo metodo!<br />\n"; }

        function metodo3()
        { echo "Ciao sono il terzo metodo!<br />\n"; }

        function richiamaMetodi()
        {
             $num = 3;
             $prefisso = "metodo";

             for ($i = 1; $i <= $num; $i++)
             {
                  $funzione = $prefisso . $i;
                  $this->$funzione();
             }
        }
   }

   $obj = new OggettoInventato();
   $obj->richiamaMetodi();

?>
```

Il risultato ottenuto è il medesimo.

Tags : Funzioni Variabili Oggetti Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 28/07/2007 - 20:07

Gli Oggetti e l'Ereditarietà

Il concetto dell'**ereditarietà**, è uno dei più importanti della programmazione orientata agli**oggetti**, a cui si appoggiano altri metodi avanzati di programmazione come ad esempio il**Polimorfismo** o le **Classi Astratte** che tratteremo nei capitoli seguenti.

L'**ereditarietà** ci consente di creare delle classi (**classi derivate** o **sottoclassi**) basate su classi già esistenti (**classi base** o **superclassi**).

Un grande vantaggio è quello di poter riutilizzare il codice di una**classe di base** senza doverlo modificare.

L'**ereditarietà** ci consente quindi di scrivere del codice molto più flessibile, in quanto permette una generalizzazione molto più forte di un concetto, rendendo più facile descrivere una situazione di vita reale.

Pensate alla **classe base** come ad un oggetto che descrive un concetto generale, e pensate invece alle**sottoclassi** come ad una specializzazione di tale concetto, esteso mediante proprietà e metodi aggiuntivi.

Imparare a programmare utilizzando questi concetti è un passo fondamentale per costruire applicazioni di un certo livello.

Vediamo ora un esempio dove definiremo una classe base e di seguito due derivate (sottoclassi) attraverso la parola chiave **extends**.
Segue la definizione della **classe madre** (base) :

Animale.php

```
<?php

     class Animale // Classe Base
     {
          public $zampe;
          public $ordine;
          public $nome;

          public function Animale($z, $o, $n) // Costruttore
          {
               $this->zampe = $z;
               $this->ordine = $o;
               $this->nome = $n;
          }
```

```
          protected function stampaDati() // Metodo protetto : può essere
richiamato solo dalle classi derivate
          { echo $this->nome . " : Zampe = " . $this->zampe . " / Ordine =
" . $this->ordine; }
     }

?>
```

Cane.php

```
<?php

     require_once("Animale.php");

     class Cane extends Animale // "Sottoclasse" o "Classe Derivata"
     {
          public function Cane() // Costruttore classe derivata
          {
               parent::Animale(4, "Vertebrati", "Cane"); // Chiamata al
costruttore della classe madre

               /* oppure più generalizzato

               parent::__construct(4, "Vertebrati", "Cane"); */
          }

          public function stampaDati($suono)
          {
               echo "Faccio $suono perchè sono ";
               parent::stampaDati(); // Chiamata al metodo protetto della
classe madre
          }
     }

?>
```

class="importante">Gallina.php

```
<?php

     require_once("Animale.php");

     class Gallina extends Animale // "Sottoclasse" o "Classe Derivata"
     {
          public function Gallina() // Costruttore classe derivata
          {
               parent::Animale(2, "Vertebrati", "Gallina"); // Chiamata al
costruttore della classe madre
          }

          public function stampaDati($suono)
          {
               echo "Faccio $suono perchè sono ";
               parent::stampaDati(); // Chiamata al metodo protetto della
classe madre
          }
     }

?>
```

Nel codice soprastante abbiamo dichiarato una classe base "**Animale**" e due classi derivate da questa : "**Cane**" e "**Gallina**".

Il metodo **stampaDati** di **Animale** è dichiarato con il modificatore di accesso **protected**, ossia può essere utilizzato solo all'interno della classe madre **Animale** e all'interno di tutte le classi da essa derivate, quindi anche **Cane** e **Gallina**.

Per richiamare il costruttore della classe madre dalle figlie è sufficiente utilizzare **parent::** invece di **this**, in questo modo accediamo direttamente alla classe base, allo stesso modo con cui **self::** viene usato al posto di **this**.

Ora vediamo un frammento di codice dove utilizziamo le classi appena dichiarate :

test.php

```
<?php

     require_once("Cane.php");
     require_once("Gallina.php");

     $cane = new Cane();
     $cane->stampaDati("bau");

     echo "\n\n<br /><br />\n\n";

     $gallina = new Gallina();
     $gallina->stampaDati("chicchirichì");

?>
```

Il risultato ottenuto.

In questo modo possiamo ampliare e modificare "**Animale**" senza effettivamente modificarne la classe e quindi il file **Animale.php**.

Sarà sufficiente fare le modifiche necessarie direttamente alla classi derivate e, qualora si presentassero nuove esigenze, sarà possibile creare una nuova sottoclasse di **Animale** senza intaccare il funzionamento delle due derivate **Cane** e **Gallina**, o addirittura potremmo creare una nuova sottoclasse derivata da una delle sottoclassi di **Animale** come nell'esempio che segue.

Alano.php

```
<?php

     require_once("Cane.php");
```

```
class Alano extends Cane
{
	private $suono;

	public function Alano()
	{
		parent::Cane();
		$this->suono = "wuoff";
	}

	public function stampaDati()
	{
		parent::stampaDati($this->suono);
		echo " / Razza : Alano";
	}
}

?>
```

test.php

```
<?php

	require_once("Alano.php");

	$cane = new Alano();
	$cane->stampaDati();

?>
```

Il nuovo risultato ottenuto.

Nota : PHP non supporta l'ereditarietà multipla, offrendoci come alternativa le**Interfacce**.

Tags : Oggetti Ereditarietà Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 29/07/2007 - 22:31

Gli Oggetti e il Polimorfismo

PHP ci consente di sfruttare le potenzialità del **Polimorfismo**, una tecnica di programmazione orientata agli **oggetti** e basata sull'**ereditarietà**.

Il **Polimorfismo** è la capacità di utilizzare un unico metodo in grado di comportarsi in modo specifico quando applicato a tipi di dato differenti.

Questo è reso possibile grazie all'**ereditarietà**, che consente alle **sottoclassi** di ridefinire i metodi ereditati dalla classe base.

Il linguaggio può identificare una qualunque istanza della sottoclasse, come un'istanza della classe base, poichè per il principio dell'**ereditarietà**, ogni sottoclasse possiede per natura tutte le proprietà della classe base (attributi e metodi).

Vedremo ora due classi normali, senza l'utilizzo di ereditarietà e polimorfismo, che implementeremo successivamente :

Esempio senza Polimorfismo

```
<?php

     class Cane
     {
          function zampeCane()
          {
               echo "Zampe : 4";
          }
     }

     class Gallina
     {
          function zampeGallina()
          {
               echo "Zampe : 2";
          }
     }

     function numeroZampe($oggetto)
     {
          if ($oggetto instanceof Cane) // Condizione = Se oggetto è un
istanza della classe Cane
          {
               $oggetto->zampeCane();
          }
          else if ($oggetto instanceof Gallina)
          {
               $oggetto->zampeGallina();
          }
          else
          {
               echo "Tipo di oggetto non riconosciuto!!";
          }
     }

     numeroZampe(new Cane()); // Stampa : "Zampe : 4"
     numeroZampe(new Gallina()); // Stampa : "Zampe : 2"

?>
```

Ora riscriverò il codice dell'esempio soprastante utilizzando il **polimorfismo**

Esempio con Polimorfismo

```
<?php

     class Animale
     {
          /* Rendiamo protetto il metodo per obbligare una redefinizione in
una
          sottoclasse necessaria a sbloccare l'utilizzo del metodo
dall'esterno.
          Più avanti per ottenere un risultato migliore useremo le Classi
Astratte. */

          protected function zampe()
          {
               echo "Errore : la funzione va obbligatoriamente ridefinita
da una sottoclasse!";
          }
     }

     class Cane extends Animale
     {
          public function zampe()
          {
               echo "Zampe : 4";
          }
     }

     class Gallina extends Animale
     {
```

```
          public function zampe()
          {
               echo "Zampe : 2";
          }
     }

     function numeroZampe($oggetto)
     {
          if ($oggetto instanceof Animale) // Condizione = Se oggetto è un
istanza di Animale o derivata da essa
          {
               $oggetto->zampe();
          }
          else
          {
               echo "Tipo di oggetto non riconosciuto!!";
          }
     }

     numeroZampe(new Cane()); // Stampa : "Zampe : 4"
     numeroZampe(new Gallina()); // Stampa : "Zampe : 2"

?>
```

Nel secondo esempio possiamo notare innanzitutto che abbiamo stabilito un nome univoco per il nostro metodo (**"zampe()"**), ereditato direttamente dalla classe base **Animale**.

Altro punto di forza è la flessibiità ottenuta nella funzione esterna **numeroZampe()**, che sarà in grado di gestire anche altri classi oltre a **Cane** e **Gallina**, che potremo definire anche in un secondo momento senza necessità alcuna di apportare modifiche alla funzione **numeroZampe()** o alle tre classi (Animale, Cane e Gallina).

Nota : l'operatore **instanceof** si comporta come un normale operatore logico binario, e viene impiegato per individuare se un'istanza è della classe specificata o derivata da essa.

Tags : Polimorfismo Oggetti Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 31/07/2007 - 0:54

Clonare gli Oggetti

Ho deciso di scrivere questo capitolo vista la novità introdotta con **PHP 5** in merito alla creazione di istanze.

In PHP 4, durante la creazione di un oggetto attraverso la parola chiave **new**, veniva restituito l'oggetto stesso e questo veniva memorizzato nella variabile specificata.

In **PHP 5** invece, quando creiamo una nuova istanza (**$oggetto = new MiaClasse();**), **new** ci restituisce non il nuovo oggetto ma bensì un riferimento ad esso.

Se quindi non fate attenzione manipolando le istanze dei vari oggetti, potreste facilmente incappare in un errore tanto facile da risolvere quanto difficile da trovare.

Ricapitolando, in **PHP 5** se assegnate ad una variabile l'istanza di un oggetto, l'assegnazione avverrà per riferimento poichè l'istanza stessa contiene solo un riferimento all'oggetto creato.

Se volete quindi creare una copia di un'istanza, dovrete clonarla.
Per clonare un oggetto è sufficiente mettere la parola chiave **clone** dopo l'operatore di assegnazione "**=**".

Esempio clonazione Oggetti

```
<?php

     class Oggetto
     {
          public $valore;

          public function Oggetto($v)
          {
               $this->valore = $v;
          }
     }

     $istanza1 = new Oggetto(5);
     $istanza2 = $istanza1; // Assegnazione per riferimento
     $istanza3 = clone $istanza1; // Clonazione oggetto

     $istanza2->valore = 7; // Modifica anche $istanza1
     $istanza3->valore = 13;

     echo $istanza1->valore; // Non stampa 5 ma 7
     echo $istanza2->valore; // Stampa 7
     echo $istanza3->valore; // Stampa 13

?>
```

Volendo è possibile controllare il processo di clonazione.
Per farlo **PHP** ci consente di definire all'interno delle nostre classi il metodo **__clone()**.

```
<?php

     class Oggetto
     {
          public $valore;

          public function Oggetto($v)
          {
               $this->valore = $v;
          }

          public function __clone()
          {
               $this->valore = "Sono stato clonato!";
          }
     }

     $istanza1 = new Oggetto(5);
     $istanza2 = clone $istanza1;

     echo $istanza1->valore; // Stampa 5
     echo $istanza2->valore; // Stampa "Sono stato clonato!"

?>
```

Clonando un oggetto, per definizione si crea una copia esatta di tale oggetto, quindi i riferimenti contenuti in esso saranno comunque copiati come tali, e dopo la clonazione conterranno ancora il riferimento alla stessa risorsa di prima.

Tags : Clonare Oggetti Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 01/08/2007 - 3:17

Classi Astratte

Le **classi astratte** ci permettono di specificare con esattezza quali classi e quali metodi devono obbligatoriamente essere ridefiniti da una sottoclasse per poter essere utilizzati.

La sottoclasse derivata da una classe astratta, dovrà per forza definire tali metodi astratti per far sì che l'ereditarietà venga accettata.

Se quindi decidete di costruire una classe con il solo scopo di fornire una base per altri classi derivate da essa, allora le classi astratte fanno al caso vostro.

Esempio Classe astratta

```
<?php

     abstract class Animale
     {
          protected $zampe;

          protected function Animale($z)
          {
               $this->zampe = $z;
          }

          public function numeroZampe()
          {
               echo $this->zampe;
          }

          abstract protected function suono();
     }

     class Cane extends Animale
     {
          public function Cane()
          {
               parent::Animale(4);
          }

          public function suono()
```

```
        { echo "bau!"; }
    }

    $rex = new Cane();
    $rex->numeroZampe(); // Stampa 4
    $rex->suono(); // Stampa "bau!"

?>
```

Come potete notare dall'esempio soprastante, il **metodo astratto "suono()"** della classe astratta **"Animale"** è solamente dichiarato, mentre la definizione avviene direttamente nella classe derivata.

Alcuni linguaggi **OOP** consentono sia di dichiarare che di definire il metodo astratto nella classe base, per fornire un implementazione di default al metodo astratto, ma **PHP** non lo permette, obbligando a lasciare nella classe astratta solo la dichiarazione della funzione.

Se non avessimo fornito una definizione per il metodo astratto **suono()**, ossia dichiarando **Cane** in questo modo ...

Esempio senza definizione del metodo astratto

```
class Cane extends Animale
{
    public function Cane()
    {
        parent::Animale(4);
    }
}
```

... avremmo ottenuto il seguente errore :
Fatal error: Class Cane contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (Animale::suono) in **C:\AppServ\www\test\test.php**

Tags : Guida Php 5 Classi Oggetti
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 02/08/2007 - 5:41

Interfacce

Teoria sulle Interfacce

Come già accennato nel capitolo "**Gli Oggetti e l'Ereditarietà**", **PHP** non supporta l'**ereditarietà multipla** fra classi, ma solo fra interfacce o fra classi e interfacce, se quindi vi sarà necessario creare una sottoclasse che erediti le proprietà da più classi, allora dovrete creare delle **Interfacce** che gliele forniscano.

Lo scopo delle **interfacce** è quello di fornire un preciso set di metodi base per le classi, mediante la dichiarazione di metodi astratti, che andranno poi definiti nelle sottoclassi che li erediteranno implementando l'interfaccia.

Le **interfacce** possono avere solo metodi che saranno di default astratti e costanti.
Non sono pertanto ammessi :

- Attributi di nessun tipo (Variabili membro)
  Fatal error: Interfaces may not include member variables
- Definizioni di metodi, è consentita solo la dichiarazione poichè ripeto, i metodi delle **interfacce** devono essere astratti
  Fatal error: Interface function INTERFACCIA::FUNZIONE() cannot contain body
- La parola chiave **abstract** non è necessaria, pertanto deve essere omessa
  Fatal error: Access type for interface method INTERFACCIA::FUNZIONE() must be omitted
- Modificatori di accesso sui metodi (**public**, **protected** e **private**)
  Fatal error: Access type for interface method INTERFACCIA::FUNZIONE() must be omitted

Dichiarazione Interfaccia

Per dichiarare un'**interfaccia** si usa la parola chiave **interface** in questo modo :

```
<?php

     interface NomeInterfaccia
     {
          const COSTANTE = "valore";

          function metodoAstratto();
          function altroMetodo(Oggetto $obj);
     }

?>
```

Implementare ed estenere Interfacce

Per implementare un'interfaccia in una classe usate il comando **implements**, per estendere invece un'interfaccia in un'altra interfaccia dovete usare il comando **extends** come per le classi derivate :

```
<?php

     interface Interfaccia1
     {
          const COSTANTE = "valore";

          function metodo1();
          function metodo2();
     }

     interface Interfaccia2
     {
          function altroMetodo1();
          function altroMetodo2();
     }

     interface Interfaccia3 extends Interfaccia1, Interfaccia2
     {
          function metodoSpeciale1();
          function metodoSpeciale2();
     }

     class MiaClasse1 implements Interfaccia3
     {
          public function metodo1() { /* Corpo del Metodo */ }
          public function metodo2() { /* Corpo del Metodo */ }
          public function altroMetodo1() { /* Corpo del Metodo */ }
          public function altroMetodo2() { /* Corpo del Metodo */ }
          public function metodoSpeciale1() { /* Corpo del Metodo */ }
          public function metodoSpeciale2() { /* Corpo del Metodo */ }
     }

     class MiaClasse2 implements Interfaccia2
     {
          public function altroMetodo1() { /* Corpo del Metodo */ }
          public function altroMetodo2() { /* Corpo del Metodo */ }
     }

?>
```

Una classe o un'interfaccia può implementare / estendere più di un'interfaccia, solo se tali interfacce non contengono la dichiarazione degli stessi metodi o costanti. Per generare un conflitto è sufficiente anche solo una costante o un metodo in comune.

```
<?php

     interface Interfaccia1
```

```
    {
        const COSTANTE = "valore";

        function metodo1();
        function metodo2();
    }

    interface Interfaccia2
    {
        function metodo1(); // Genererà un errore nella dichiarazione di
MiaClasse1
        function altroMetodo1();
        function altroMetodo2();
    }

    class MiaClasse1 implements Interfaccia1, Interfaccia2
    {
        public function metodo1() { /* Corpo del Metodo */ }
        public function metodo2() { /* Corpo del Metodo */ }
        public function altroMetodo1() { /* Corpo del Metodo */ }
        public function altroMetodo2() { /* Corpo del Metodo */ }
    }

?>
```

L'esempio soprastante genera il seguente errore :
Fatal error: Can't inherit abstract function Interfaccia2::metodo1() (previously declared abstract in Interfaccia1)

Nella prossima pagina un'esempio molto pratico per farvi familiarizzare maggiormente con questo potente strumento.

Tags : Interfacce Ereditarietà Multipla Oggetti Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 03/08/2007 - 8:04

Esempio pratico per l'uso di Interfacce

Introduzione

In questo capitolo scriverò delle classi e delle interfacce per la creazione e la gestione di alcuni account per un sito web.

Ovviamente il codice non sarà utilizzabile, in quanto solo parzialmente scritto. Lo scopo è quello di far capire meglio le potenzialità delle **interfacce**, mediante un esempio pratico facilmente intuibile e più vicino ad un'applicazione reale.

Definizione del concetto base

Questa pseudo-libreria, fornirà gli strumenti per creare e gestire 3 tipi di account :

- Utente - Utente normale senza privilegi particolari
- Premium - Utente premium con la possibilità di gestire una propria casella messaggi
- Amministratore - Utente con privilegi amministrativi ed una casella messaggi personale

Per questo scopo scriveremo 1 **classe astratta**, 2 **interfacce** e 3 **sottoclassi** elencate di seguito :

- **Account** - **Classe Astratta**
  Fornirà gli attributi di base per gli altri account e un metodo astratto per registrare gli account nell'archivio del sito (es. Database MySQL o file esterno)
- **Amministrazione** - **Interfaccia**
  Fornirà dei metodi astratti di base per amministrare l'utenza, inserendo, modificando e cancellando utenti dall'archivio del sito
- **CasellaMessaggi** - **Interfaccia**
  Fornirà dei metodi astratti per la gestione di una casella messaggi personale interna al sito
- **Utente** - **Sottoclasse derivata da Account**
  Oggetto per la memorizzazione dei dati su un utente con metodo per la registrazione in archivio
- **Premium** - **Sottoclasse derivata da Account che implementa l'interfaccia CasellaMessaggi**
  Oggetto per memorizzare un utente con privilegi premium e metodi per registrare l'utente in archivio e gestire una casella messaggi personale
- **Amministratore** - **Sottoclasse derivata da Account che implementa l'interfaccia Amministrazione e CasellaMessaggi**
  Oggetto per la memorizzazione di un account di amministrazione, con metodi per la registrazione dell'account in archivio, la gestione di altri utenti (inserimento, modifica e cancellazione) e la gestione di una casella messaggi personale

Ecco un diagramma che mostra tutti gli oggetti e le loro relazioni.

Se state usando un browser aggiornato (non Internet Explorer 6) e avete una risoluzione bassa (1024x768) potreste vedere il diagramma rimpicciolito, cliccate allora sul primo bottone in altro a destra () per ingrandirlo in un'altra finestra.

Di seguito il codice testato con la corretta sintassi :

```
<?php

     // Classe Astratta di Base
     abstract class Account
     {
          protected $nome_utente;
          protected $pass_utente;
          protected $data_iscrizione = NULL;

          abstract protected function registraAccount();
     }

     // Inizio INTERFACCE
     interface Amministrazione
     {
          function aggiungiUtente(Utente $utente);
          function modificaUtente(Utente $utente);
          function cancellaUtente(Utente $utente);
     }

     interface CasellaMessaggi
     {
          const MAX_MESSAGGI = 100;

          function controllaCasella();
          function leggiMessaggio($id_messaggio);
          function cancellaMessaggio($id_messaggio);
     }
     // Fine INTERFACCE

     // Classi per la definizione dei vari tipi di Account

     class Utente extends Account
     {
          public function Utente($n, $p, $d)
          {
               $this->nome_utente = $n;
               $this->pass_utente = $p;
               $this->data_iscrizione = $d;
          }

          public function registraAccount()
          { /* Codice per la registrazione dell'utente */ }
     }

     class Premium extends Account implements CasellaMessaggi
     {
          public function Premium($n, $p, $d)
          {
               $this->nome_utente = $n;
               $this->pass_utente = $p;
               $this->data_iscrizione = $d;
          }

          public function registraAccount()
          { /* Codice per la registrazione dell'utente */ }

          public function controllaCasella()
          { /* Codice per il controllo della Casella Messaggi */ }

          public function leggiMessaggio($id_messaggio)
          { /* Codice per la lettura di un messaggio della casella */ }

          public function cancellaMessaggio($id_messaggio)
          { /* Codice per la cancellazione di un messaggio dalla casella
*/ }
     }

     class Amministratore extends Account implements Amministrazione,
CasellaMessaggi
     {
          public function Amministratore($n, $p)
          {
               $this->nome_utente = $n;
               $this->pass_utente = $p;
          }

          public function registraAccount()
          { /* Codice per la registrazione dell'utente */ }

          public function aggiungiUtente(Utente $utente)
          {
               echo "Stai aggiungendo l'utente " . $utente->nome_utente .
"<br />\n";
               // Codice per aggiungere un utente all'archivio ...
               echo "Utente " . $utente->nome_utente . " aggiunto con
successo!<br />\n";
          }

          public function modificaUtente(Utente $utente)
          {
               echo "Stai modificando l'utente " . $utente->nome_utente .
"<br />\n";
               // Codice per modificare un utente dell'archivio ...
               echo "Utente " . $utente->nome_utente . " modificato con
```

```
successo!<br />\n";
          }

          public function cancellaUtente(Utente $utente)
          {
               echo "Stai cancellando l'utente " . $utente->nome_utente .
"<br />\n";
               // Codice per cancellare un utente dall'archivio ...
               echo "Utente " . $utente->nome_utente . " cancellato con
successo!<br />\n";
          }

          public function controllaCasella()
          { /* Codice per il controllo della Casella Messaggi
dell'Amministratore */ }

          public function leggiMessaggio($id_messaggio)
          { /* Codice per la lettura di un messaggio della casella
dell'Amministratore */ }

          public function cancellaMessaggio($id_messaggio)
          { /* Codice per la cancellazione di un messaggio dalla casella
dell'Amministratore */ }
     }

     // Inizio codice di esempio

     $utente = new Utente("mario", "miapassword", 1185456501);
     $admin = new Amministratore("francesco", "altrapass");

     $admin->aggiungiUtente($utente);

?>
```

Le ultime tre righe di codice, puramente a scopo illustrativo, genereranno il seguente output :

Stai aggiungendo l'utente mario
Utente mario aggiunto con successo!

Ricordatevi che l'**ereditarietà multipla** è concessa solo fra interfacce con **extends**, mentre fra classi è necessario ricorrere all'implementazione di **interfacce** con **implements**.

Nota : le interfacce non possono ereditare dalle classi.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 04/08/2007 - 10:27

Classi e Metodi final

Come abbiamo visto nei capitoli precedenti, una sottoclasse ha la possibilità di ridefinire i metodi della classe madre.

E' possibile impedire la ridefinizione di determinati **metodi** e o la derivazione da determinate **classi**, usando la direttiva **final**.

Se quindi desiderate che non vengano fornite ulteriori implementazioni di un metodo della vostra classe, allora è sufficiente far precedere alla parola chiave **function**, la direttiva **final** in questo modo :

Metodo final

```
<?php

     class ClasseBase
     {
          public $attributo;

          public final function metodo()
          {
               echo "Sono un metodo finale!";
          }
     }

     class SottoClasse extends ClasseBase
     {
          public function metodo() // Genera un errore fatale
          {
               echo "Proviamo a ridefinire il metodo finale!";
          }
     }

?>
```

Il codice soprastante genera il seguente errore poichè **SottoClasse** ha tentato di ridefinire un metodo finale di **ClasseBase** :
Fatal error: Cannot override final method ClasseBase::metodo()

Se invece volete impedire che una **sottoclasse** derivi da una classe specifica, applicate a quest'ultima la direttiva **final** seguita dalla parola chiave **class** e dal nome classe in questo modo :

Classe final

```
<?php

     final class ClasseBase
     {
          public $attributo;

          public function metodo()
          { /* ... */ }
     }

     class SottoClasse extends ClasseBase // Genera un errore fatale
     {
         /* ... */
     }

?>
```

Il codice soprastante genera il seguente errore fatale :
Fatal error: Class SottoClasse may not inherit from final class (ClasseBase)

Tags : Classi Metodi Final Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 05/08/2007 - 12:51

Gestione degli Errori

Teoria sulle Eccezioni

Come abbiamo visto nel capitolo **Strutture di controllo** (Parte 4 di 4), al verificarsi di una situazione inaspettata, quindi di un errore dal momento che l'applicazione non è stata programmata per produrre quello stato, è possibile interrompere lo script con le istruzioni **exit** e **die**.

Così facendo però non si ha una gestione dell'errore intelligente ma una drastica terminazione dello script.

Per essere precisi, il termine esatto per definire questi eventi inaspettati che si verificano durante l'esecuzione dello script, è **Eccezioni**.

Un evento imprevisto, un baco nello script, la perdita di una connessione al Database, sono tutte considerate **eccezioni** che **PHP** ci consente di gestire attraverso tre comandi molto semplici :

- **throw** - Solleva un'eccezione
- **try** - Se si verifica un'eccezione nel codice del blocco **try**, quest ultimo fa saltare l'esecuzione del codice al blocco **catch**
- **catch** - Contiene il codice alternativo che viene eseguito al verificarsi dell'eccezione

Il sistema è molto semplice ed è paragonabile ad un normale blocco **IF ELSE**.

Il codice per cui intendete gestire le **eccezioni** va all'interno del blocco **try**, che obbligatoriamente dovrà avere un blocco antagonista **catch** che conterrà il codice alternativo da eseguire nel caso in cui si verificasse un'**eccezione**.

Se all'interno del blocco **try** si verifica una condizione per cui viene chiamato il comando **throw**, allora l'esecuzione salta al blocco **catch**.

Non è necessario che l'istruzione **throw** si trovi esattamente nel blocco **try**, ma può essere richiamata anche da una funzione o da un metodo di un oggetto all'interno di **try**.

Esempio di gestione errori

Nell'esempio che segue vedremo del codice che ha come compito il controllo dei dati di un utente.
Se i dati vengono ritenuti adeguati si procede alla registrazione nel database, altrimenti si notifica l'errore all'utente, mettendogli a disposizione un link per tornare indietro a reinserire i dati.

Il codice essendo solo dimostrativo, considera i dati non validi se superano una certa lunghezza, e rimanda l'utente ad una ipotetica pagina di registrazione chiamata registrazione.php :

```
<?php

     define("MAX_LUNGHEZZA", 20);

     function registraUtente()
     {
          // ... codice per registrare l'utente nel database

          $registrato = true; // Simuliamo una registrazione avvenuta con
successo

          if ($registrato)
              echo "Utente registrato con successo!";
          else
              throw new Exception("Impossibile registrare l'utente!");
     }

     /* --- Inizio dati Utente --- */

     $nome_utente = "Agamennone";
     $pass_utente = "supercalifragilistichespiralidoso";

     /* --- Fine dati Utente --- */

     try
     {
          if (strlen($nome_utente) > MAX_LUNGHEZZA)
              throw new Exception("Nome utente troppo lungo!");

          if (strlen($pass_utente) > MAX_LUNGHEZZA)
              throw new Exception("Password troppo lunga!");

          registraUtente();
     }
     catch (Exception $exc)
     {
          echo "Errore : " . $exc->getMessage() . "<br />\n\n";
          echo "<br /><a href=\"registrazione.php\">Torna indietro</a> per
reinserire i dati.";
     }

?>
```

Il codice soprastante produce questo risultato.

un fallimento nella registrazione e viene sollevata una nuova eccezione ottenendo il messaggio :
Errore : Impossibile registrare l'utente!

Ovviamente per far verificare l'ultimo caso descritto, è necessario dare alle due variabili $nome_utente e $pass_utente, una lunghezza minore o uguale a 20, perchè vengono controllate prima dell'esecuzione della funzione "registraUtente()."

Osservazioni sulle eccezioni

Come avrete notato dall'ultimo esempio, quando **throw** solleva un'eccezione, manda al dovuto blocco **catch** un'istanza della classe **Exception**, nativa di **PHP**.

Throw quindi, utilizza **new** per creare una nuova istanza, e a seguire il costruttore di **Exception** con un messaggio opzionale per inizializzarne l'istanza.

La suddetta istanza viene poi catturata da **catch**, che si serve del metodo "getMessage()" per leggere il messaggio inviatogli da **throw** e gestire l'eccezione come programmato.

Nota : Se viene sollevata un'eccezione al di fuori di un blocco **try / catch**, questa interromperà lo script con un errore fatale :
Fatal error: Uncaught exception ...

Tags : Gestione Errori Eccezioni Throw Try Catch Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 06/08/2007 - 15:14

La Classe Exception

La classe **Exception** è una classe **built-in**, ossia nativa del linguaggio **PHP**.

E' possibile estendere questa classe, creandone delle proprie derivate da essa, ma prima di creare le nostre **Sottoclassi** personalizzate è necessario conoscere **Exception** e sapere che opportunità ci offre.

Vediamo subito la dichiarazione della classe :

```
<?php

     class Exception
     {
          protected $message = 'Unknown exception'; // exception message
          protected $code = 0; // user defined exception code
          protected $file; // source filename of exception
          protected $line; // source line of exception

          function __construct($message = null, $code = 0);

          final function getMessage(); // message of exception
          final function getCode(); // code of exception
          final function getFile(); // source filename
          final function getLine(); // source line
          final function getTrace(); // an array of the backtrace()
          final function getTraceAsString(); // formated string of trace

          /* Overrideable */
          function __toString(); // formated string for display
     }

?>
```

Diamo un occhiata più da vicino alle proprietà della suddetta classe :

Attributi protetti

- **$message** - Il messaggio dell'eccezione
- **$code** - Il codice errore definito dall'utente
- **$file** - Il file sorgente dove è stata sollevata l'eccezione
- **$line** - La riga del sorgente dove è stata sollevata l'eccezione

Costruttore

- **__construct()** - Costruttore della classe

Metodi finali

- **getMessage()** - Restituisce $message
- **getCode()** - Restituisce $code
- **getFile()** - Restituisce $file
- **getLine()** - Restituisce $line
- **getTrace()** - Restituisce un array di backtrace()
- **getTraceAsString()** - Restituisce una stringa formattata del trace

Metodi ridefinibili

- **__toString()** - Restituisce una stringa formattata dell'oggetto

Nel prossimo capitolo approfondiremo l'argomento creando delle sottoclassi di**Exception**, per una gestione delle eccezioni maggiormente personalizzata.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 07/08/2007 - 17:37

Gestione Avanzata delle Eccezioni

Abbiamo visto nei capitoli precedenti, che l'istruzione**throw** quando solleva un'eccezione solleva in realtà un oggetto della classe **Exception**.

PHP ci consente inoltre di sollevare delle eccezioni proprie, attraverso la creazione di**sottoclassi** specifiche che devono obbligatoriamente essere derivate dalla built-in**Exception**.

A questo proposito vedremo come sarà possibile annidare più blocchi**catch** ad un blocco **try**, similmente ad una struttura di controllo classica come la**IF - ELSE IF**, che ci consentirà di controllare che tipo di eccezione è stata sollevata e agire di conseguenza.

Illustrerò ora un esempio analogo a quello spiegato nel capitolo**Gestione degli Errori**, inserendo però una classe aggiuntiva per gestire le eccezioni sugli input utente, lasciando la classe**Exception** per le altre eccezioni non previste.

Esempio eccezioni personalizzate

Per questo esempio ho preferito creare una classe**Utente**, che controlla automaticamente la correttezza dei dati, e se necessario solleva un'eccezione del tipo personalizzato**ErroriUtente**.

Quest'ultima eredita come abbiamo detto prima, dalla classe**Exception**, fornendo un metodo aggiuntivo per l'inserimento opzionale di un link che invita l'utente a tornare indietro per reinserire i dati.

La funzione "registraUtente()" invece è rimasta invariata, quindi in caso di errore solleva un'eccezione normale che sarà catturata dal secondo blocco**catch**.

```
<?php

     class Utente
     {
          public $nome_utente;
          public $pass_utente;

          public function Utente($n, $p)
          {
               if (strlen($n) > ErroriUtente::MAX_LUNGHEZZA)
                   throw new ErroriUtente($n);
```

```
            if (strlen($p) > ErroriUtente::MAX_LUNGHEZZA)
                throw new ErroriUtente($p);

            $this->nome_utente = $n;
            $this->pass_utente = $p;
        }
    }

    class ErroriUtente extends Exception
    {
        const MAX_LUNGHEZZA = 20;

        private $errore;

        public function ErroriUtente($stringa)
        {
            $this->errore = "Errore : La lunghezza massima consentita è
" . self::MAX_LUNGHEZZA;
            $this->errore .= "\n<br />\"$stringa\" è di " .
strlen($stringa) . " caratteri!";
        }

        private function stampaLink($pagina = "registrazione.php")
        {
            echo "<a href=\"$pagina\">Torna indietro</a> per reinserire
i dati.";
        }

        public function stampaMessaggio($link = false)
        {
            echo $this->errore . "<br /><br />\n\n";

            if ($link) $this->stampaLink();
        }
    }

    function registraUtente(Utente $utente)
    {
        // ... codice per registrare l'utente nel database

        $registrato = true; // Simuliamo una registrazione avvenuta con
successo

        if ($registrato)
            echo "Utente registrato con successo!";
        else
            throw new Exception("Impossibile registrare l'utente,
riprova più tardi!");
    }

    try
    {
        $utente = new Utente("Amircare",
"supercalifragilistichespiralidoso");

        registraUtente($utente);
    }
    catch (ErroriUtente $e)
    {
        $e->stampaMessaggio(true);
    }
    catch (Exception $e)
    {
        echo $e->getMessage();
    }

?>
```

Il codice produce questo risultato.

Analisi del codice

Nello script viene sollevata un'eccezione **ErroriUtente** dal costruttore di **Utente** nella prima riga del blocco **try**, che viene catturata dal primo blocco **catch**.

Se impostate una stringa più corta di "supercalifragilistichespiralidoso" (mannaggia Mary Poppins), e assegnate $registrato a **false** dentro registraUtente(), allora otterrete il seguente messaggio senza link : Impossibile registrare l'utente, riprova più tardi!

Quest'ultima eccezione viene invece gestita dal secondo blocco **catch**.

Suggerimenti sulle eccezioni

Ricordatevi che non sempre è necessario definire un **costruttore** per la vostra **sottoclasse** di **Exception**, se **PHP** non lo troverà effettuerà automaticamente una chiamata al costruttore della classe madre, ossia **Exception**.

Inoltre usate questo sistema solo per gestire le eccezioni, per risolvere quindi quei problemi imprevisti che si presume non accadano poi così spesso.

Non utilizzate assolutamente la gestione delle eccezioni come una sruttura di controllo, non solo per un discorso di performance (le eccezioni sono più lente), ma anche per mantenere il codice il più robusto e leggibile possibile, evitando problemi di manutenzione futuri.

Infine prestate attenzione al tipo di eccezioni che gestite in determinati blocchi **try** / **catch**, poichè con questo sistema se catturate un eccezione anche molto grave, non interromperete lo script, e il codice successivo sarà quindi eseguito :

```
<?php

    function sollevaEccezione()
    { throw new Exception("Eccezione grave!"); }

    function registraUtente()
```

```
     { /* Codice per registrare un'utente */ }

     try
     {
          sollevaEccezione();
     }
     catch (exception $e)
     {
          echo $e->getMessage() . "<br />\n";
     }

     try
     {
          echo "Codice d'esempio che non andrebbe eseguito in caso di
eccezioni gravi!";
          registraUtente();
     }
     catch (exception $e)
     {
          echo $e->getMessage();
     }

?>
```

L'esempio produrrà in output :

Eccezione grave!
Codice d'esempio che non andrebbe eseguito in caso di eccezioni gravi!

E' buona norma quindi minimizzare la quantità di blocchi **try** / **catch** e se necessario annidateli.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 08/08/2007 - 20:01

Serializzare gli Oggetti

In questo capitolo vedremo come **PHP** ci consente di serializzare i nostri oggetti.

Serializzare un oggetto significa essenzialmente trasformarlo in una stringa (byte-stream di tutti i valori dell'oggetto) che rappresenterà l'istanza dell'oggetto al suo stato corrente, che potremo poi memorizzare dove preferiamo (es. Database MySQL, file di testo esterno ...).

Per fare questo useremo due funzioni native di **PHP 5** : **serialize()** e la sua antagonista **unserialize()**.

unserialize() ci consentirà di ricostruire l'oggetto che abbiamo precedentemente serializzato, ponendolo all'interno dell'istanza specificata.

Nell'esempio che segue definirò una classe d'esempio **ContatoreAccessi**, la cui istanza sarà serializzata in un file esterno "**statistiche.txt**" che conterrà l'oggetto in questione, il cui compito sarà quello di contare quante volte è stata aperta la pagina.

```
<?php

     class ContatoreAccessi
     {
          private $cont;

          public function MiaClasse($v)
          { $this->cont = $v; }

          public function stampa()
          { echo "Accessi : " . $this->cont; }

          public function aumenta()
          { $this->cont++; }
     }

     $nomefile = "statistiche.txt";
     $dimensione = filesize($nomefile);

     if ($dimensione) // Se il file esiste e contiene più di un byte
     {
          $file = fopen($nomefile, "r"); // Apre il file in sola lettura
          $content = file_get_contents($nomefile); // Legge il contenuto
```

```
del file e lo memorizza in $content

          $obj = unserialize($content); // Ricostruisce l'istanza
deserializzando $content
          $obj->aumenta(); // Incrementa il contatore di accessi
     }
     else
          $obj = new ContatoreAccessi(1);

     $file = fopen($nomefile, "w+"); // Riapre il file in scrittura
azzerandone il contenuto

     $ser = serialize($obj); // Serializza l'istanza
     fwrite($file, $ser); // Memorizza l'istanza serializzata
     fclose($file);

     echo $obj->stampa();

?>
```

Eseguendo il codice soprastante noterete che il contatore di accessi cresce ogni volta che aprite la pagina. Aprendo "**statistiche.txt**" col Blocco Note e cancellandone il contenuto, azzererete il contatore di accessi.

Potete memorizzare gli oggetti serializzati ovunque come ad esempio un Database MySQL, mantenendo il preciso stato in cui si trova la vostra applicazione o se preferite parte di essa, notando subito il vantaggio che una volta effettuata la deserializzazione, otterrete immediatamente la vecchia istanza pronta da utilizzare, con i suoi attributi e i suoi metodi.

Abbiate cura che la definizione della **classe** dell'istanza che andrete a serializzare / deserializzare, sia sempre accessibile dai sorgenti che effettuano questo tipo di operazioni, o otterrete un oggetto inutile.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 09/08/2007 - 22:24

Metodi Magici

PHP 5 ci fornisce un supporto aggiuntivo per potenziare le nostre classi attraverso la definizione di determinati metodi detti **Metodi Magici**, in quanto sono nativi di **PHP** e non è possibile dichiararne di propri con lo stesso nome (riservati).

Vediamo subito una lista di questi metodi e i servizi che offrono :

- **__toString()** - Metodo per restituire l'oggetto sotto forma di stringa
- **__set()** - Permette l'overloading per assegnare un valore ad un attributo
- **__get()** - Consente l'overloading delle istruzioni di lettura di un attributo
- **__isset()** - Overloading del metodo **isset()** per gli attributi di una classe (disponibile da **PHP** 5.1.0)
- **__unset()** - Overloading del metodo **unset()** per gli attributi di una classe (disponibile da **PHP** 5.1.0)
- **__call()** - Permette l'overloading dei metodi di una classe
- **__clone()** - Consente l'overloading per la **clonazione degli oggetti**
- **__autoload()** - Funzione per l'auto-inclusione di file per classi non ancora definite
- **__sleep()** - Richiamato subito prima di una serializzazione di un oggetto
- **__wakeup()** - Richiamato subito dopo una serializzazione di un oggetto
- **__set_state()** - Metodo statico richiamato per le classi esportate con **var_export()** (disponibile da **PHP** 5.1.0)

Overload di __toString()

Prototipo della funzione :

```
string __toString()
```

Effettuando l'overloading di questa funzione, consentiremo a **PHP 5** di trattare le istanze della nostra classe come stringhe, qualora venissero fatte su di esse operazioni come ad esempio la stampa su schermo con **echo** o ancora la concatenazione fra stringhe ecc...

Non essendo supportato un tipo di **casting** implicito da **Oggetto** a **Stringa**, questo risulta un ottimo metodo per aggiungere tale supporto alle nostre classi :

```
<?php

     class Persona
     {
          private $nome;
          private $cognome;
```

```
        public function Persona($n, $c)
        {
             $this->nome = $n;
             $this->cognome = $c;
        }

        public function __toString()
        { return $this->nome . " " . $this->cognome; }
    }

    $obj = new Persona("John", "Doe");
    echo $obj; // Stampa "John Doe"

    $stringa = "Ciao sono " . $obj;
    echo $stringa; // Stampa "Ciao sono John Doe"

?>
```

Nota : Prima di **PHP 5.2.0**, **__toString()** veniva chiamato solo in combinazione con **echo** e **print**.

Nei capitoli a seguire vedremo nel dettaglio l'overload degli altri **Metodi Magici** sopra elencati.

Tags : Metodi Magici Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 11/08/2007 - 0:47

Il metodo Magico __set()

Overload di __set()

Prototipo della funzione :

```
void __set (string name, mixed value)
```

> Questo metodo ci consente di effettuare delle operazioni di assegnazione tradizionali sugli attributi della classe.
>
> Il primo parametro è il nome dell'attributo mentre il secondo parametro rappresenta il valore da assegnare a tale attributo.
>
> Se ad esempio abbiamo un'istanza di una ipotetica classe '**Oggetto**" che ha definito il metodo **__set()**, questo sarà richiamato quando verrà effettuata un'operazione di assegnazione ad un attributo della classe non esistente o non accessibile (**private** o **protected**), passando alla funzione i soliti due parametri, il nome dell'attributo sotto forma di stringa e il valore da assegnargli.

Esempi di overloading

```
<?php

     class Oggetto
     {
          public function __set($attributo, $valore)
          { /* Corpo funzione */ }
     }

     $obj = new Oggetto();
     $obj->pippo = 32;

?>
```

L'ultima riga di codice quindi ($obj->pippo = 32), corrisponde ad una chiamata al metodo **__set()** con questi parametri : __set("pippo", 32);
Questo accade perchè nella definizione della **classe** "**Oggetto**", non abbiamo dichiarato nessun attributo di nome "pippo".

Vediamo un esempio più complesso :

```php
<?php

     class Stringa
     {
          private $stringa;
          const MAX_CHARS = 15;

          public function Stringa($valore)
          {
               if (strlen($valore) > self::MAX_CHARS)
               { throw new Exception("Stringa troppo lunga (max " .
self::MAX_CHARS . " car.)"); }
               else
               { $this->stringa = $valore; }
          }

          public function __set($attributo, $valore)
          {
               if ($attributo == "stringa")
               { self::Stringa($valore); }
               else
               { throw new Exception("Attributo non esistente!"); }
          }

          public function __toString()
          { return $this->stringa; }
     }

     try
     {
          $obj = new Stringa("Ciao ciao!!!");
          echo "$obj<br>\n"; // Stampa "Ciao ciao!!!"

          $obj->stringa = "Ancora ciao!";
          echo "$obj<br>\n"; // Stampa "Ancora ciao!"

          $obj->stringa = "Un ultimo saluto!";
          echo "$obj<br>\n"; // Solleva un'eccezione con il messaggio :
"Stringa troppo lunga (max 15 car.)"
     }
     catch (Exception $e)
     {
          echo $e->getMessage();
     }

?>
```

no usato per simulare un accesso ad un attributo di tipo **public**.

Questo mi consente di effettuare dei controlli precisi sui valori assegnati all'attributo $stringa, senza comunque perdere la trasparenza e la comodità della classica operazione di assegnamento, tipica degli attributi **public**.

A seguire un ultimo esempio, dove gestirò gli attributi della **classe** "**Coordinata**", come se fossero degli attributi normali memorizzandoli invece in un unico attributo di tipo **Array**.

Il questo modo avremo la sicurezza che venga sempre effettuato un **casting** esplicito di tipo **float** durante l'assegnazione dei valori, e risparmieremo inoltre righe di codice durante la gestione degli pseudo-attributi potendo utilizzare una semplice **foreach** per scorrerli tutti, creando del codice rapido e più flessibile.

```php
<?php

     // Inizio dichiarazione Classe
     class Coordinata
     {
          private $xyz;

          public function Coordinata($x, $y, $z)
          {
               $this->xyz = array();

               $this->xyz["x"] = (float) $x;
               $this->xyz["y"] = (float) $y;
               $this->xyz["z"] = (float) $z;
          }

          public function __set($attributo, $valore)
          {
               switch ($attributo)
               {
                    case "x" :
                    case "y" :
                    case "z" :
                         $this->xyz[$attributo] = (float) $valore;
                         break;
                    default :
                         throw new Exception("Attributo non valido!");
                         break;
               }
          }

          public function __toString()
          {
               $return = "";

               foreach ($this->xyz as $chiave => $valore)
                    $return .= "$chiave = $valore<br />\n";

               $return .= "<br>\n";
```

```
            return $return;
        }
    }
    // Fine dichiarazione Classe

    try
    {
        $punto = new Coordinata(15, 17.3, 22.4);

        echo $punto;

        $punto->x = 20.3;
        $punto->z = 11.2;

        echo $punto;

        $punto->a = 22.1; // L'attributo "a" non esiste e sarà sollevata
un'eccezione
    }
    catch (Exception $e)
    { echo $e->getMessage(); }

?>
```

Il codice soprastante produrrà questo risultato.

Tags : Overloading Metodi Magici __set() Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 12/08/2007 - 3:11

Il metodo Magico __get()

Overload di __get()

Prototipo della funzione :

```
mixed __get(string name)
```

Al contrario di **__set()**, l'overload di questo metodo ci consente di effettuare operazioni sulla lettura di un attributo.

Il metodo **__get** viene richiamato quando si tenta di accedere ad attributi con restrizioni (**private** o **protected**) o ad attributi non dichiarati.

Vediamo un semplice esempio :

```
<?php

    class Utente
    {
        private $nickname;
        private $password;
        public $dataiscrizione;

        public function Utente($n, $p)
        {
            $this->nickname = $n;
            $this->password = $p;
            $this->dataiscrizione = "01-01-1970";
        }

        public function __get($attributo)
        {
            switch ($attributo)
            {
                case "nickname" :
                    return "Nickname : " . $this->nickname;
                    break;

                case "password" :
                    return "Password : " . $this->password;
```

```
                    break;

                default :
                    return "Attributo inesistente";
                    break;
            }
        }
    }

    $utente = new Utente("johndoe", "mypassword");

    echo $utente->nickname . "<br />";
    echo $utente->password . "<br />";
    echo $utente->dataiscrizione . "<br />";
    echo $utente->nonesisto;

?>
```

L'esempio produce questo risultato :

Nickname : johndoe
Password : mypassword
01-01-1970
Attributo inesistente

Le prime due **echo** richiamano **__get()** e stampano correttamente i due attributi **private** ossia $nickname e $password, a cui **__get()** applica anche un prefisso col nome dell'attributo, come si evince dai primi due **case** della **switch**.

La terza **echo** invece stampa direttamente l'attributo $dataiscrizione senza passare da **__get()**, poichè tale attributo è esistente ed accessibile essendo dichiarato come **public**.

Infine l'ultima **echo** passa nuovamente per **__get()** stampando la stringa "Attributo inesistente" come programmato nell'opzione di **default** della **switch**.

Tags : Overloading Metodi Magici __get() Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 13/08/2007 - 5:34

Overload di __isset() e __unset()

Prototipi di __isset() e __unset()

```
bool __isset (string name)
void __unset (string name)
```

Effettuando un **overloading** su questi due **metodi magici**, abbiamo la possibilità di controllare quando vengono effettuate delle chiamate alle funzioni **isset()** e **unset()** (descritte qui) su un attributo della classe.

Anche in questo caso, le chiamate vengono effettuate solo per attributi non accessibili o non dichiarati.

Il metodo **__isset()** viene richiamato anche per l'overload su **empty()** :

```
<?php

     class Oggetto
     {
          private $privato;
          public $pubblico;

          public function __isset($attributo)
          {
               echo "<br />Chiamata a __isset() da \"$attributo\" = ";

               switch ($attributo)
               {
                    case "privato" :
                         return isset($this->privato);
                         break;

                    case "pubblico" :
                         return isset($this->pubblico);
                         break;

                    default :
                         throw new Exception("Attributo non esistente");
                         break;
               }
          }

          public function __unset($attributo)
          {
               echo "<br />Chiamata a __unset() da \"$attributo\"";
               unset($attributo);
          }
     }

     try
     {
          $obj = new Oggetto();

          echo (int) isset($obj->pubblico);
          echo (int) isset($obj->privato);
          echo (int) empty($obj->privato);

          unset($obj->bho);
          echo (int) isset($obj->nonesisto);
     }
     catch (Exception $e)
     { echo $e->getMessage(); }

?>
```

Il codice produce il seguente risultato :

```
0
Chiamata a __isset() da "privato" = 0
Chiamata a __isset() da "privato" = 1
Chiamata a __unset() da "bho"
Chiamata a __isset() da "nonesisto" = Attributo non esistente
```

Analizziamo il codice nel blocco **try** :

- Prima **echo** : non provoca chiamate al metodo magico **__isset()** ma direttamente alla funzione standard **isset()** perchè l'attributo **Oggetto->pubblico** è accessibile (**public**)
- Seconda **echo** : provoca una chiamata a **__isset()** che ritorna **false** da isset($this->privato); poichè $privato non è stato inizializzato
- Terza **echo** : provoca una chiamata a **__isset()** che ritorna **true** perchè $privato non è stato inizializzato (quindi è vuoto)
- **unset()** : provoca una chiamata a **__unset()** che stampa ("Chiamata a __unset() da "bho"")
- Quarta **echo** : provoca una chiamata a **__isset()** che solleva un'eccezione ("Attributo non esistente")

Nota : nelle **echo** viene effettuato il **casting** esplicito al tipo **int**, perchè se **echo** riceve il valore booleano **false** non stampa nulla.

Inoltre fate attenzione alla versione di **PHP** che utilizzate perchè l'overloading di queste funzioni è disponibile solo da **PHP 5.1.0**

Tags : Overloading Metodi Magici __isset() __unset() Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 14/08/2007 - 7:57

Il metodo Magico __call()

Overload di __call()

Prototipo della funzione :

```
mixed __call (string method, array arguments)
```

Mediante l'overloading del metodo magico **__call()**, abbiamo la possibilità di catturare e gestire tutte le chiamate a metodi non esistenti ossia non dichiarati nella classe.

L'overload di questo metodo torna particolarmente utile qualora si utilizzi una strategia di programmazione come la **Delegation Pattern**.

```
<?php

     class Oggetto
     {
          private $arr;

          public function Oggetto()
          {
               $this->arr = array();
          }

          private function aggiungiAllaFine($valore)
          { array_push($this->arr, $valore); }

          private function aggiungiAllInizio($valore)
          { array_unshift($this->arr, $valore); }

          public function __call($metodo, $parametri)
          {
               if ($metodo == "aggiungi")
               {
                    if ($parametri[1] == "inizio")
                    { $this->aggiungiAllInizio($parametri[0]); }
                    else if ($parametri[1] == "fine")
                    { $this->aggiungiAllaFine($parametri[0]); }
                    else
                    { throw new Exception("Parametri non validi"); }
```

```
            }
            else
            { throw new Exception("Metodo inesistente"); }
      }

      public function mostra()
      {
            foreach ($this->arr as $valore)
            {
                  echo "$valore<br />\n";
            }
      }
}

try
{
      $obj = new Oggetto();

      $obj->aggiungi(3, "inizio");
      $obj->aggiungi("pippo", "inizio");
      $obj->aggiungi(44.5, "fine");
      $obj->aggiungi("ciao", "inizio");

      $obj->mostra();

      $obj->aggiungi(5);
}
catch (Exception $e)
{ echo $e->getMessage(); }

?>
```

Il codice produce questo output sul browser :

ciao
pippo
3
44.5
Parametri non validi

Nota : __call() può restituire dei valori proprio come un metodo tradizionale, e come tale può chiamare anche funzioni esterne e metodi di altre classi.

Nell'articolo Wikipedia da me segnalato sulla **Delegation Pattern** non viene usato **__call()**, ma viene spiegato semplicemente l'utilizzo di questa strategia di programmazione, a cui potrete poi facilmente applicare l'overload di **__call()**.

Tags : Overloading Metodi Magici Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 15/08/2007 - 10:21

La funzione Magica __autoload()

E' buona pratica nella programmazione orientata agli oggetti, avere un file per la dichiarazione di ogni classe, ognuno nominato con lo stesso identico **case** usato per il nome della classe.

Questo ci consente di trovare immediatamente il codice di una classe, e di apportarvi modifiche immediatamente disponibili in tutti i sorgenti che utilizzano tale classe, facendoci risparmiare molto tempo durante la manutenzione del codice.

Questo metodo ci consentirà anche di farci risparmiare molte righe di codice perchè ci da la possibilità di includere esclusivamente le classi che realmente sono necessarie per la corretta esecuzione dello script.

In applicazioni reali però, diventa problematico inserire in ogni file dello script decine e decine di righe di inclusioni di file esterni, e a questo proposito **PHP** ci mette a disposizione la funzione Magica **__autoload**().

Esempio con __autoload()

ClasseUno.php

```
<?php

     class ClasseUno
     {
          public $attributo = "Attributo di Classe Uno";
     }

?>
```

ClasseDue.php

```
<?php

     class ClasseDue
     {
          public $attributo = "Attributo di Classe Due";
     }

?>
```

test.php

```
<?php

     function __autoload($classe)
     {
          require_once("$classe.php");
     }

     $obj = new ClasseUno();
     echo $obj->attributo . "<br />\n";

     $obj = new ClasseDue();
     echo $obj->attributo;

?>
```

L'output prodotto è il seguente :

Attributo di Classe Uno
Attributo di Classe Due

Come potete notare dalla pagina **test.php**, non ho esplicitamente incluso i file esterni **ClasseUno.php** e **ClasseDue.php**, ma ci ha pensato la funzione **__autoload**() richiamata automaticamente da **PHP** che non ha trovato subito le dichiarazioni delle classi.

Nota : fate attenzione a non specificare più di una funzione **__autoload**() per script o riceverete un errore fatale.

Fatal error : Cannot redeclare __autoload() (previously declared in C:\AppServ\www\test\test.php:3)

Inoltre i nomi delle classi in **PHP** sono **case-insensitive**, mentre **__autoload**() li gestisce come **sensitive**, perciò prestate molta attenzione a quando nominate i file, che come dicevo all'inizio è buona norma che siano identici al nome della classe.

Tags : Overloading Metodi Magici Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 16/08/2007 - 12:44

I metodi magici __sleep() __wakeup e __set_state()

I primi due **metodi magici** che illustrerò in questo capitolo sono **__sleep()** e **__wakeup()**, che hanno in comune la **Serializzazione degli Oggetti** descritta in questa guida prima della serie sui **metodi magici**.

PHP richiamerà il metodo **__sleep()** subito prima di una serializzazione di una istanza della nostra classe. **__wakeup()** invece sarà chiamato subito dopo una deserializzazione dell'oggetto, rendendolo utile ad esempio per ristabilire una connessione al database chiusa prima della serializzazione.

Altra differenza importante fra i due metodi è che **__sleep()** deve ritornare un'array contenente la lista degli attributi da serializzare, e questo ci consente di avere un controllo maggiore sulle serializzazioni della nostra classe.

Se omettete il valore di ritorno di **__sleep()** la serializzazione fallirà restituendo un valore **NULL**.

Vediamo un esempio teorico :

```
<?php

     class Oggetto
     {
          public $attributo = "stringa";

          public function __sleep()
          {
               echo "L'oggetto sta per essere serializzato!<br>";
               return array("attributo");
          }

          public function __wakeup()
          {
               echo "L'oggetto è appena stato deserializzato!<br>";
          }
     }

     $obj = new Oggetto();

     $s = serialize($obj);
     echo "Oggetto serializzato = $s<br>";
```

```
     $u = unserialize($s);
     echo $u->attributo;

?>
```

Il codice produce il seguente risultato :

```
L'oggetto sta per essere serializzato!
Oggetto serializzato = O:7:"Oggetto":1:{s:9:"attributo";s:7:"stringa";}
L'oggetto è appena stato deserializzato!
stringa
```

Overload di __set_state()

__set_state() è un metodo statico che viene richiamato per l'esportazione delle classi tramite **var_export()**.

L'unico parametro di questo metodo è un **array** contenente le proprietà esportate nella forma array ("proprietà" => valore, ...)

Il metodo è disponibile da **PHP 5.1.0**

E con questo capitolo abbiamo definitivamente concluso la serie sui **metodi magici**.

Tags : Overloading Metodi Magici Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 17/08/2007 - 15:07

Overload dell'operatore di accesso degli Array

Un'altra caratteristica che rende molto versatile **PHP 5** è l'overloading dell'operatore di accesso degli **Array** : **[]**

Per effettuare questo **overload** è necessario far sì che le nostre classi implementino l'interfaccia nativa **ArrayAccess**.
Quest'ultima fornisce i metodi astratti che ci permetteranno di utilizzare le istanze delle nostre classi come se fossero degli **array** omettendo quindi il classico operatore di selezione **->** con i vantaggi che ne derivano dalla gestione di un **array** :

```
<?php

     class Oggetto implements ArrayAccess
     {
          public $attributo1 = "ciao";
          public $attributo2 = "pippo";

          /* Ridefinizione dei metodi di ArrayAccess

          ...

          */
     }

     $obj = new Oggetto();

     echo $obj["attributo1"] . " " . $obj["attributo2"]; // Stampa "ciao
pippo"

?>
```

Vediamo cosa contiene l'interfaccia **ArrayAccess** prima di illustrare l'overload vero e proprio e le sue potenzialità.

interface ArrayAccess

- **bool** offsetExists ($index)
- **mixed** offsetGet ($index)
- **void** offsetSet ($index, $new_value)
- **void** offsetUnset ($index)

offsetExists viene richiamato per verificare se esiste l'indice specificato nel nostro pseudo-array.

offsetGet è invocato quando si tenta di leggere un valore all'indice specificato.

offsetSet è chiamato quando si tenta di scrivere un valore all'indice specificato.

offsetUnset è invocato quando si tenta di cancellare un valore all'indice specificato.

Esempio ArrayAccess

Di seguito un esempio dove mostrerò come unire le potenzialità di un oggetto al pratico sistema di gestione degli **array**.

Il codice nell'esempio ovviamente è solo parzialmente scritto in quanto serve solo a far capire il funzionamento di questo **overload**.

Ho creato una classe **Impiegati** che rappresenta una lista di impiegati e i lori stipendi gestita come se fosse un **array**, ma essendo anche un **oggetto** potrà vantare altri metodi per avviare operazioni più complesse come la stampa su schermo dell'intera lista degli impiegati o il salvataggio di quest'ultima su un file esterno.

```
<?php

     class Impiegati implements ArrayAccess
     {
          private $lista;

          public function Impiegati()
          {
               $this->lista = array();
          }

          private function convertiChiave($chiave)
          { return ucwords(strtolower($chiave)); }

          function offsetExists($nome)
          {
               /* Converto la chiave dell'array assicurandomi che sia
sempre minuscola
               con la sola iniziale maiuscola in modo da rendere gli
accessi case-insensitive

               NOTA : le chiavi degli array nel linguaggio nativo sono
invece case-sensitive */
               $nome = $this->convertiChiave($nome);

               foreach ($this->lista as $chiave => $valore)
                    if ($chiave == $nome)
                         return true;

               return false;
          }

          function offsetGet($nome)
          {
               $nome = $this->convertiChiave($nome);

               if ($this->offsetExists($nome))
                    return $this->lista[$nome];
               else
                    return NULL;
          }

          function offsetSet($nome, $valore)
          {
               $nome = $this->convertiChiave($nome);
               $this->lista[$nome] = $valore;
          }

          function offsetUnset($nome)
          {
               $nome = $this->convertiChiave($nome);

               if ($this->offsetExists($nome))
                    $this->lista[$nome] = "Licenziato";
          }

          public function mostraStipendi()
          {
               foreach ($this->lista as $chiave => $valore)
               {
                    if ($valore == "Licenziato")
                    { echo "$chiave è stato $valore<br />\n"; }
                    else
                    { echo "$chiave ha uno stipendio di $valore euro<br />
\n"; }
               }
          }

          public function salvaSuFile($nomefile)
          {
              /* Codice per salvare la lista degli impiegati su file

              ...
```

```
            */
        }
    }

    $impiegati = new Impiegati();

    $impiegati["Mario"] = 1250;
    $impiegati["Luca"] = 2300;
    $impiegati["Giovanni"] = 1570;
    $impiegati["MARIO"] = 3740; // Sovrascrive "Mario" perchè converto
sempre la chiave : ucwords(strtolower($chiave))

    // Licenzia Luca
    unset($impiegati["Luca"]);

    $impiegati->mostraStipendi();

?>
```

L'esempio produce il seguente risultato :

```
Mario ha uno stipendio di 3740 euro
Luca è stato Licenziato
Giovanni ha uno stipendio di 1570 euro
```


Tags : Overload Array Classi Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 18/08/2007 - 17:31

Overload dell'interfaccia di Iterazione

Illustrerò in questo capitolo l'ultimo **overload** che vedrà come protagonista l'interfaccia nativa **Iterator**.

Implementando questa **interfaccia** nelle nostre classi, avremo modo di fornire gli strumenti a **PHP** per gestire tali oggetti nei cicli di iterazione.

L'**overload** di questa interfaccia abbinata ad ArrayAccess vi consentirà di costruire degli strumenti veloci e flessibili per interagire più rapidamente con le vostre classi.

Gli indici per effettuare le iterazioni sugli oggetti dovranno essere attributi interni della classe.

Vediamo come è costituita l'interfaccia **Iterator** :

interface Iterator

- **void** rewind()
- **mixed** current()
- **mixed** key()
- **void** next()
- **bool** valid()

- **rewind()** - Azzera l'indice del ciclo facendolo tornare all'inizio
- **current()** - Ritorna il valore della posizione corrente
- **key()** - Restituisce l'indice / chiave della posizione corrente
- **next()** - Aumenta l'indice per puntare all'elemento successivo
- **valid()** - Ritorna **true** o **false** per controllare se l'array è finito (viene interrogato prima di una chiamata a **current()** o **key()**)

Esempio overload Iterator

Per creare un esempio più semplice possibile non ho volutamente utilizzato l'ereditarietà, ma ho creato due semplici classi base.

La prima classe è **Unità** e rappresenta un unità presente in un magazzino, con il nome, prezzo e codice.

La seconda e ultima classe (**Magazzino**) è un semplice contenitore di **Unità**, che memorizzerà utilizzando un **array** classico monodimensionale, dove l'elemento singolo non sarà un semplice valore ma l'oggetto più

complesso **Unità**.

```
<?php

     class Unita
     {
          public $nome;
          public $prezzo;
          public $codice;

          public function Unita($n, $p, $c)
          {
               $this->nome = $n;
               $this->prezzo = $p;
               $this->codice = $c;
          }
     }

     class Magazzino implements Iterator
     {
          private $pezzi;
          private $current; // Indice interno per gestire l'iterazione

          public function Magazzino()
          {
               $this->pezzi = array();
          }

          /* Inizio Interfaccia Iterator */
          function rewind()
          {
               $this->current = 0;
          }

          function current()
          {
               return $this->pezzi[$this->current];
          }

          function key()
          {
               return $this->current;
          }

          function next()
          {
               return $this->current  ;
          }

          function valid()
          {
               return isset($this->pezzi[$this->current]);
          }
          /* Fine Interfaccia Iterator */

          public function aggiungi(Unita $unita)
          {
               array_push($this->pezzi, $unita);
          }
     }

     $store = new Magazzino();

     $store->aggiungi(new Unita("Le avventure dello zio Pasquale", 15.5,
"A096H"));
     $store->aggiungi(new Unita("I delitti di Grottaferrata", 12.0,
"B076X"));
     $store->aggiungi(new Unita("Quando dico Basta!!", 23.5, "KJ87F2"));

     foreach ($store as $unita)
     {
          echo "Nome = $unita->nome<br />\n";
          echo "Prezzo = $unita->prezzo<br />\n";
          echo "Codice = $unita->codice<br /><br />\n\n";
     }

?>
```

Il codice produce il seguente risultato :

```
Nome = Le avventure dello zio Pasquale
Prezzo = 15.5
Codice = A096H

Nome = I delitti di Grottaferrata
Prezzo = 12
Codice = B076X

Nome = Quando dico Basta!!
Prezzo = 23.5
Codice = KJ87F2
```


Tags : Guida Php 5 Overload Interfacce
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 19/08/2007 - 19:54

Osservazioni finali sugli Oggetti

Con questo capitolo concludo tutta la parte che riguarda la struttura e la sintassi del linguaggio, per avvicinarmi a un livello più avanzato di questa **guida**, dove inizierò ad illustrare i metodi e gli strumenti che **PHP 5** ci fornisce per realizzare delle **vere applicazioni web**, per interagire con l'utente e scambiare dati e molto altro.

Prima di proseguire volevo darvi qualche consiglio su come usare al meglio quello che avete imparato finora.

Se non avete mai programmato ad **Oggetti** e siete ancora un pò restii a farlo, vi assicuro che è solo questione di pratica e vi troverete poi dannatamente male scontrandovi con altri linguaggi che non sono invece **Object Oriented**.

Questa strategia di programmazione è alla base di molti linguaggi evoluti odierni, e vi permetterà di risparmiare tonnellate di codice e di tempo ... provare per credere.

Vorrei aggiungere inoltre, che molti troveranno scomoda la decisione di **PHP** di non essere molto tipizzato come ad esempio il **C++**.

Per molti altri invece potrà essere un vantaggio, io in ogni caso vi consiglio di fare molta attenzione quando sviluppate i vostri script, facendo largo uso dell'operatore **instanceof** e specificando ogni volta che vi sarà possibile il tipo di **Classe** usata nei parametri delle funzioni, sfruttando il più possibile il Polimorfismo, per essere sempre certi che la vostra applicazione stia lavorando sulle classi giuste.

```
<?php

     $obj = new MioOggetto();

     if ($obj instanceof MioOggetto)
          // continua è tutto ok
     else throw new Exception("Tipo di dato non corretto");

?>
```

Usate sempre **try** e **catch** per gestire le eccezioni, ma usateli solo per le eccezioni appunto e mai come struttura di controllo.

Se siete capitati qui per caso, perchè avete cercato la frase sbagliata su Google vi consiglio di dare una letta anche al resto :-P

Non avete idea (o magari si LOL) di quanta gente che si dice esperta di IT non sa neanche fare una ricerca corretta su Internet.
Perdonate il fuori tema, ma visto che ci sono vi informo che le ricerche migliori e più rilevanti sono proprio

quando beccate le frasi in mezzo a un testo in tema, che difficilmente non saranno articolate visto che nessuno parla come un cazzo di robot.

Un esempio stupido ... è più probabile ottenere dei risultati rilevanti cercando con varie frasi articolate tipo "marmellata di fragole" piuttosto che scrivere in stile telegramma "marmellata fragole".

E qui non vi metto neanche la pubblicità in mezzo al testo ... tanto nei 40 euro di tenuta server annui non ci rientro comunque.

Buon proseguimento e buone ricerche.

Tags : Classi Oggetti Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 20/08/2007 - 22:17

Codice PHP nelle pagine HTML

In questo capitolo vedremo come è possibile inserire del codice **PHP** in mezzo a del codice **HTML**.

Abbiamo già visto come fare il contrario, semplicemente stampando i tag HTML come una stringa direttamente con **echo**, avendo cura di utilizzare gli appropriati caratteri di escape per, ad esempio, il doppio apice ".

Per inserire del codice **PHP** è sufficiente racchiuderlo fra questi due tag :

- Tag di apertura : **<?php**
- Tag di chiusura : **?>**

```
<html>
	<head>
		<title>Codice PHP nelle pagine HTML</title>
	</head>
	<body>
		<?php echo "Ciao da PHP"; ?>
	</body>
</html>
```

PHP ci mette a disposizione altri due tipi di tag, disponibili se nel file di configurazione del vostro server (php.ini) la variabile **short_open_tag** è settata a "On".

In questo modo abbiamo a disposizione la variante più corta del tag descritto precedentemente, che omette **php** dopo il punto interrogativo :

```
<html>
	<head>
		<title>Codice PHP nelle pagine HTML</title>
	</head>
	<body>
		<? echo "Ciao da PHP"; ?>
	</body>
</html>
```

Abbiamo infine a disposizione un'ultima variante di **Short Tag**, che manda direttamente in output il contenuto senza bisogno dei costrutti **echo** o **print**. Per utilizzarlo è sufficiente aggiungere il simbolo **=** dopo il punto interrogativo :

```
<?php

	$href = "http://www.realizzazione-sito.info/";
```

```
     $title = "Guide e Tutorial gratuiti";
     $target = "_blank";

?>
<html>
     <head>
          <title>Codice PHP nelle pagine HTML</title>
     </head>
     <body>
          <a href="<?=$href?>" title="<?=$title?>" target="<?=$target?
>">Realizzazione Sito . info</a>
     </body>
</html>
```

Nota : se nel file **php.ini** avete attivo **asp_tags** potete usare i tag in stile ASP **<% %>**
Se non avete modo di accedere al file **php.ini**, il modo più veloce per verificare la disponibilità di questi tag, è ovviamente creare una pagina di prova che li utilizza per verificarne il corretto funzionamento.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 22/08/2007 - 0:41

Inviare dati coi metodi GET e POST

Come ogni linguaggio **server-side** che si rispetti, anche **PHP** ci consente lo scambio di dati fra pagine con i metodi classici che **HTML** ci mette a disposizione : **GET** e **POST**.

Con entrambi i metodi l'invio dei dati avviene tramite un normale **form** HTML, e **PHP** ci consente di prelevarli attraverso i due array globali **$_GET[]** e **$_POST[]**.

Al form HTML sono necessari due paramentri principali : **action** e **method**.

A seconda del **DOCTYPE** che utilizzate nella vostra pagina e dagli scopi del vostro script potrebbero servire anche i parametri **id** e **name** (enctype, target ...) che io ormai specifico sempre perchè abituato ad usare il **DOCTYPE XHTML 1.0** che richiede fra l'altro che i valori dei parametri siano racchiusi fra doppi apici, ma non è scopo di questa guida che illustrerà solo il funzionamento della parte **PHP**, vi consiglio pertanto lo studio delle ultime versioni di **HTML** e **XHTML** prima di cominciare a sviluppare delle applicazioni a livello commerciale.

Tornando al nostro form HTML vediamo a cosa corrispondono i parametri **action** e **method** :

- **action** - Indica la pagina a cui inviare i dati. Se lasciato vuoto punterà di default alla pagina stessa.
- **method** - Indica il metodo in cui avviene lo scambio di dati : "**get**" o "**post**".

L'attributo "**name**" degli elementi "**input**" usati all'interno del form HTML è molto importante, poichè rappresenta la chiave che useremo per accedere all'array globale **$_GET[]** e che ci consentirà di leggere il valore del dato inviato.

Esempio invio di dati

Iniziamo con un esempio semplice composto da due pagine : una in **HTML** che invia i dati e una in **PHP** che li riceve.

inviodati.html

```
<html>
     <head>
          <title>Pagina HTML per l'invio dei dati</title>
     </head>
     <body>
          <form action="ricezionedati.php" method="get">
               <input type="text" name="miotesto" />
```

```
            <input type="submit" value="Invia dati" />
        </form>
    </body>
</html>
```

Osserviamo i due attributi del **form**, **action** assegnato a "ricezionedati.php", che sarà la pagina incaricata di ricevere i dati inviati, e **method** assegnato a **get**, per specificare che stiamo usando appunto il metodo di invio dati **GET**.

Infine notiamo i due elementi "**input**" del **form**, uno di tipo **text** (campo di testo) e uno di tipo **submit** (bottone per inviare i dati).
L'attributo **name** del campo di testo è settato al valore "**miotesto**", ossia la stringa che useremo per accedere al valore del campo di testo dalla pagina **PHP**.

ricezionedati.php

```
<html>
    <head>
        <title>Pagina PHP per la ricezione dei dati</title>
    </head>
    <body>
        <?php

            if (strlen($_GET["miotesto"]))
                echo "Mi hai inviato la stringa \"" .
$_GET["miotesto"] . "\"";
            else
                echo "Non mi hai inviato nulla!!";

        ?>
    </body>
</html>
```

Niente di più semplice, usiamo il nome dell'attributo **name** per accedere all'array globale **$_GET[]** e il gioco è fatto.

Con la funzione **strlen()** effettuo un semplice controllo per vedere se è stato inviato almeno un carattere.

Una volta inviati i dati e caricata la pagina **ricezionedati.php**, potrete notare nella barra degli indirizzi del vostro browser che i dati inviati sono in chiaro e perfettamente leggibili dall'utente finale :

`http://localhost/test/ricezionedati.php?miotesto=testo+da+inviare`
Il metodo **GET** infatti, consiste semplicemente nell'accodare all'URL della pagina un punto interrogativo a cui seguiranno tutte le variabili inviate con la sintassi **variabile=valore**, separate da una E commerciale & :

`pagina.php?variabile1=valore&variabile2=valore&variabile3=altro+valore`
Il metodo **POST** è equivalente al **GET** con la differenza che **POST** quando invia i dati non li mette in chiaro nell'URL.

Un metodo è migliore di un altro a seconda del caso ma mai a prescindere.
Come forse avrete notato, in questo stesso sito vengono usati entrambi i metodi.

Ho usato **POST** per inviare dati sensibili preservando un minimo di sicurezza, mentre ho usato **GET** per far sapere a **guide.php** quale pagina deve caricare, mettendo in chiaro il titolo della pagina nell'URL, favorendo il posizionamento sui motori di ricerca per quelle determinate parole chiave (Web Marketing).

Scrivendo applicazioni e creando siti vi ritroverete senzaltro in situazioni in cui preferirete **POST** e altre in cui è preferibile **GET**.

Esempio invio dati su pagina singola

Vedremo ora un ultimo esempio in cui inviamo e riceviamo i dati nella stessa pagina, stavolta col metodo **POST**.

invio_ricezione.php

```
<html>
    <head>
        <title>Invio e ricezione in unica pagina</title>
    </head>
    <body>
        <?php

            if (strlen($_POST["miotesto"]))
            {
                echo "Dato inviato = " . $_POST["miotesto"];
            }
            else
            {
                ?>

                    <form action="" method="post">
                        <input type="text" name="miotesto" />
                        <input type="submit" value="Invia dato" />
                    </form>

                <?php
            }

        ?>
    </body>
</html>
```

Per inviare e ricevere i dati nella stessa pagina, avrei potuto specificare quest'ultima nell'attributo **action**, ottenendo il medesimo risultato ma garantendo una maggiore compatibilità coi vecchi browser (vi consiglio se possibile di specificarlo sempre).

Tornando ai metodi **GET** e **POST**, abbiate cura di usare sempre l'array globale corretto a seconda del metodo che avete scelto.

E' possibile inoltre accedere ai dati inviati come se fossero delle variabili, indipendentemente dal metodo di invio dati utilizzato, ma è necessario avere la direttiva **register_globals** impostata a ON nei settaggi di **PHP**.
Per maggiori informazioni vi rimando alla Documentazione ufficiale.

Nota : l'estensione delle pagine contenenti sia **HTML** che **PHP** dovrà essere "**.php**" o lo script verrà interpretato come semplice testo.

Tags : Inviare Dati Get Post Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 23/08/2007 - 3:04

I Cookie

I **cookie** sono un metodo veloce ed efficace per "ricordarsi" di un utente.

Il sistema è molto semplice e permette di memorizzare dei dati nel browser del navigatore. Questi dati vengono scritti in un file che prende il nome di **cookie**, che sarà possibile creare se il browser dell'utente è configurato per accettarli.

Di norma i **cookie** sono attivi nella stragrande maggioranza dei browser, e molti siti richiedono che siano obbligatoriamente abilitati per poter usufruire dei servizi da loro offerti.

I **cookie** si possono impostare con due funzioni che **PHP** ci fornisce : **setcookie()** e **setrawcookie()**.

In **PHP** i **cookie** fanno parte dell'intestazione HTTP, motivo per cui è necessario chiamare **setcookie()** prima di mandare qualsiasi dato in output al browser dell'utente.

Se non è possibile evitare degli output prima di settare il **cookie**, interviene in nostro aiuto la funzione **ob_start()**, che posticiperà automaticamente gli output dello script, bufferizzandoli finchè non saranno impostati i **cookie** o l'intestazione.

Una volta impostati i **cookie**, sarà possibile accedervi attraverso l'array globale **$_COOKIE[]**.

Vediamo ora come è possibile impostare i **cookie** attraverso la funzione **setcookie()**.

Prototipo della funzione

```
bool setcookie ( string name [, string value [, int expire [, string path
[, string domain [, bool secure [, bool httponly]]]]]] )
```

Lista dei parametri di **setcookie()** :

1. **name** - Obbligatorio - Definisce un nome per il **cookie**
2. **value** - Facoltativo - Il valore da memorizzare nel **cookie**
3. **expire** - Facoltativo - Il tempo di scadenza del **cookie** in formato timestamp Unix
4. **path** - Facoltativo - La directory del server da cui sarà accessibile il **cookie**
5. **domain** - Facoltativo - Il dominio web da cui sarà accessibile il **cookie**
6. **secure** - Facoltativo - Per impostare il **cookie** solo su connessioni sicure **HTTPS**
7. **httponly** - Facoltativo - Per rendere accessibile il **cookie** solo attraverso il protocollo HTTP
   Nota : httponly è disponibile da **PHP 5.2.0** e non è supportato da tutti i browser ma è molto utile per ridurre gli attacchi di tipo **XSS**, impedendo l'accesso al **cookie** da linguaggi client-side come **Javascript**

Esempio Cookie

Vedremo ora un semplice esempio in una pagina singola, attraverso cui creeremo e cancelleremo un **cookie** servendoci di due form HTML separate in modo da poter inviare due comandi differenti.

Stavolta dividerò il codice in due pezzi per poterlo spiegare meglio ai neofiti.

test.php

Vediamo la prima parte della pagina, costituita da solo codice **PHP** e priva di output in modo da non inviare l'header della pagina prima di aver operato sul **cookie**.

Non uso volontariamente la funzione **ob_start()** che ci permette di bufferizzare l'output, per non diminuire inutilmente le prestazioni dello script, anche se in questo caso lo script è talmente leggero che non si noterebbe alcuna differenza, ma vi consiglio comunque di non usare mai **ob_start()** inutilmente.

```
<?php

     function cancellaCookie($nomeCookie)
     {
          unset($_COOKIE[$nomeCookie]);
          setcookie($nomeCookie, "", time() - 86400); // Imposta la
scadenza al giorno prima
     }

     if ($_POST["comando"] == "cancella")
     {
          cancellaCookie("dato");
     }
     else if ($_POST["comando"] == "imposta")
     {
          $scadenza = (int) $_POST["expire"] + time();
          setcookie("dato", $_POST["dato"], $scadenza, "/test/");

          /* Scriviamo direttamente nell'array globale per non far fallire
la prossima IF
          rendendo così il cookie disponibile da subito senza ricaricare la
pagina */
          $_COOKIE["dato"] = $_POST["dato"];
     }

?>
```

In questa prima parte di codice ho creato la funzione **cancellaCookie()**, che per cancellare i **cookie** ne imposta la data di scadenza al giorno prima. Prima di chiamare **setcookie()** però effettua una chiamata ad **unset()**.

Questo per evitare che si tenti di accedere al **cookie** cancellato prima che la pagina venga ricaricata.

I browser infatti cancellano il **cookie** (file) dall'hard disk se verificano che il cookie è effettivamente scaduto, ma appena cancellato rimane comunque disponibile nell'array globale **$_COOKIE[]** fino ad un nuovo caricamento pagina che forzerà l'aggiornamento dell'array.

Dopo la definizione della funzione **cancellaCookie()** troviamo un blocco **IF - ELSE IF** dove agiamo di conseguenza al comando inviato da una delle due form HTML che vedremo nel secondo pezzo della pagina.

Nella **else if** assegniamo la data di scadenza del **cookie**, sommando al valore **$_POST["expire"]** inviatoci da una form, la **timestamp** corrente restituita da **time()**.

Infine ho richiamato **setcookie()** impostando "/test/" come **path** valida per il **cookie**, poichè la mia pagina test.php si trova al momento sotto il seguente URL : http://localhost/test/test.php.

Vediamo il secondo pezzo di **test.php**

```
<?php /* Incollate qui il primo pezzo di test.php per fare le prove */ ?>
<html>
     <head>
          <title>Impostiamo e cancelliamo i Cookie</title>
     </head>
     <body>


          <?php

               if (isset($_COOKIE["dato"]))
               {
                    echo "Cookie impostato per ricordare il seguente dato =
\"" . $_COOKIE["dato"] . "\"";

                    ?>

                    <br /><br />

                    <form name="deletecookie" action="test.php"
method="post">
                         <input type="hidden" name="comando"
value="cancella" />
                         <input type="submit" value="Cancella cookie" />
                    </form>

                    <?php
               }
               else
               {
                    ?>

                    <form name="writecookie" action="test.php"
method="post">
                         <table width="400" border="0" cellspacing="5"
cellpadding="5">
                              <tr>
                                   <td>Dato da memorizzare</td>
                                   <td><input type="text" name="dato"
value="" /></td>
                              </tr>
                              <tr>
```

```
                                        <td>Scadenza del cookie</td>
                                        <td><input type="text" name="expire"
value="" /></td>
                                   </tr>
                                   <tr>
                                        <td><input type="submit" value="Imposta
cookie" /></td>
                                        <td> </td>
                                   </tr>
                              </table>
                              <input type="hidden" name="comando"
value="imposta" />
                         </form>

                         <?php
                    }

               ?>

          </body>
</html>
```

Questo secondo pezzo di codice è un misto di **PHP** e **HTML**, diviso sostanzialmente in due blocchi principali, una **IF** e la sua corrispondente **ELSE**.

Con la **IF** controllo se il **cookie** è impostato mediante un controllo con **isset()** sull'array globale **$_COOKIE[]**, motivo per cui nel primo pezzo di codice settiamo il **cookie** sia manualmente che con la funzione **setcookie()**.

Se il **cookie** è impostato viene stampato il suo valore e di seguito una form HTML per eliminarlo dal browser del client, che invierà una sola variabile di nome **comando** (name="comando") e impostata al valore "**cancella**" (value="cancella").

Questa variabile la controlliamo nel primo pezzo di codice attraverso la prima **IF** : **if** (**$_POST**["comando"] == "cancella")

Nell'ultima **else** abbiamo un'altra form HTML (writecookie) che ci servirà per impostare il **cookie**.

La form **writecookie** invia 3 valori :

1. **dato** - Il valore da memorizzare nel **cookie**
2. **expire** - La durata del **cookie** espressa in secondi
3. **comando** - Assegnato a "**imposta**" per indicare allo script che deve impostare il **cookie**

La prima volta che si esegue la pagina **test.php** e tutte le volte in cui il **cookie** non è impostato o scaduto, si ottiene questo output sul browser.

Una volta impostato il cookie, riaprendo la pagina (non aggiornate con **F5** o reinvierete i dati per **POST**) otterrete invece questo output.

Se poi state usando Firefox, che consiglio vivamente soprattutto se create siti in quanto ha una miriade di strumenti utili per noi sviluppatori, allora potete andare su "**Strumenti**" -> "**Opzioni...**" -> "**Privacy**" -> "**Mostra i cookie...**" e cercare il **cookie** appena impostato cercando per nome cookie o per dominio.

Una schermata del gestore **cookie** di **Firefox 2** con la lettura del **cookie** impostato nell'esempio soprastante.

Inoltre ricordatevi sempre di non memorizzare dati sensibili nei **cookie**, e se mettete delle password create un algoritmo di criptazione o appoggiatevi ad un algoritmo di terze parti che faccia un buon lavoro.

Aggiungo infine che se un browser rispetta gli standard dei **cookie**, non sarà possibile creare dei **cookie** più grandi di 4KB per un massimo di 20 **cookie** per dominio, e il browser dell'utente non accetterà inoltre dei **cookie** se ne ha già un totale di 200.

Per ovviare a questi problemi ci serviremo delle "**Sessioni**" che vedremo nei prossimi capitoli.

Nota : nella sezione Download potete trovare altre estensioni utili di Firefox 2 per sviluppare applicazioni web. Le consiglio tutte ma in particolare provate Firebug e Web Developer ... è tutto aggratis eh!

Tags : Cookie Php 5 Guida
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 24/08/2007 - 5:27

Una classe per i Cookie

In questo capitolo extra, ripeterò l'esempio del capitolo precedente utilizzando la programmazione orientata agli oggetti (**OOP**).

Ho scritto una classe **Cookie** molto semplice ma che ci consente di trarre dei vantaggi che non avremmo con la sola funzione nativa **setcookie**.

Classe Cookie

Attributi :

- **name** - Un nome per il **cookie**
- **value** - Il valore da registrare nel **cookie**
- **expire** - La durata del **cookie** espressa in secondi
- **path** - **Path** del sito da cui è accessibile il **cookie**
- **domain** - **Dominio** web da cui è accessibile il **cookie**
- **secure** - Per impostare il cookie solo su connessioni sicure HTTPS
- **httponly** - Per rendere accessibile il cookie solo attraverso il protocollo HTTP

Come per **setcookie**(), solo **name** è obbligatorio mentre tutti gli altri attributi sono facoltativi.

Metodi :

- **setcookie**() - Metodo per impostare il **cookie**
- **getPath**() - Metodo statico che restituisce la **path** dello script attualmente in esecuzione
- **unserialize**() - Metodo statico per recuperare da un **cookie** l'istanza originale della classe **Cookie**
- **cancella**() - Metodo statico per cancellare un **cookie**
- **__toString**() - Metodo che restituisce direttamente il valore del **cookie** se questo viene usato come stringa

Listato della classe Cookie

Introduco ora il codice della classe **Cookie** che risiede in un file esterno nominato Cookie.php :

```
<?php

     class Cookie
     {
          public $name;
          public $value;
          public $expire;
          public $path;
          public $domain;
          public $secure;
          public $httponly;

          public function Cookie($n, $v = "", $e = 0, $p = NULL, $d = NULL,
$s = false, $h = false)
          {
               $this->name = $n;
               $this->value = $v;
               $this->expire = ($e == 0) ? $e : (int) ($e * 60 * 60) +
time();
               $this->path = ($p == NULL) ? Cookie::getPath() : $p;
               $this->domain = $d;
               $this->secure = $s;
               $this->httponly = $h;
          }

          public function setcookie($salva_tutto = false)
          {
               $cookie = ($salva_tutto) ? serialize($this) : $this->value;

               $_COOKIE[$this->name] = $cookie;
               setcookie($this->name, $cookie, $this->expire, $this->path,
               $this->domain, $this->secure, $this->httponly);
          }

          static public function getPath()
          {
               $cartelle = explode("/", $_SERVER["PHP_SELF"]);

               $path = "";
               $dimensione = count($cartelle) - 1;

               for ($i = 0; $i < $dimensione; $i  )
                    $path .= $cartelle[$i] . "/";

               return $path;
          }

          static public function unserialize($nomeCookie)
          {
               $cookie = unserialize(stripslashes($_COOKIE[$nomeCookie]));

               if ($cookie instanceof Cookie)
                    return $cookie;
               else
                    throw new Exception("Cookie corrotto");
          }
```

```
        static public function cancella($nomeCookie)
        {
             unset($_COOKIE[$nomeCookie]);
             setcookie($nomeCookie, "", time() - 86400); // Imposta la
scadenza al giorno prima
        }

        public function __toString()
        {
             return $this->value;
        }
    }

?>
```

Vantaggi della classe Cookie

Uno dei vantaggi di questa classe è quello di poter usare solo i parametri che ci interessano realmente, senza obbligatoriamente specificare anche tutti gli altri come avviene con **setcookie()**.

Se ad esempio volessimo specificare solo il nome del **cookie**, un valore e impostare a **true** l'opzione **httponly** sarebbe sufficiente utilizzare il seguente codice :

```
<?php

     require_once("Cookie.php");

     $cookie = new Cookie("nome cookie", "valore del cookie");
     $cookie->httponly = true;
     $cookie->setcookie();

?>
```

Con **setcookie()** invece avremmo dovuto specificare per forza anche **expire**, **path**, **domain** e **secure** per poter arrivare ad **httponly** rendendo il codice molto meno intuitivo e leggibile. La variante che segue è equivalente al codice soprastante :

```
<?php

     setcookie("nome cookie", "valore del cookie", 0, "/", NULL, false,
true);

?>
```

Altra particolarità di questa **classe** è che ci consente di specificare come scadenza direttamente il numero di ore a partire dal momento in cui viene impostato il **cookie**. Potete facilmente modificare la classe, ad esempio per considerare **expire** come il numero di giorni di vita del **cookie** e così via :

```
<?php

     $cookie = new Cookie("nome cookie", "valore del cookie", 24); // Dura
24 ore
     $cookie->httponly = true;
     $cookie->setcookie();

     // oppure

     setcookie("nome cookie", "valore del cookie", time()   (60 * 60 * 24),
"/", NULL, false, true);

?>
```

Se un'istanza di **Cookie** viene stampata o concatenata come stringa, interviene il metodo magico **__toString()** restituendo direttamente il valore memorizzato nel cookie.

Punto di forza di questa classe è il metodo pubblico **setcookie()**, che come avrete notato prende un parametro **$salva_tutto** impostato di **default** al valore **false**.

Se richiamando la funzione lo impostate invece a **true**, il metodo **setcookie()** serializza l'oggetto e lo salva integro nel **cookie**.

Sarà poi possibile recuperare l'oggetto salvato con il metodo statico **unserialize**, permettendoci di leggere anche in un secondo momento tutte le proprietà del **cookie**, cosa impossibile se avessimo usato solo la funzione nativa **setcookie()**.

Esempio classe Cookie

Di seguito riporto l'esempio del capitolo precedente riscritto per la classe **Cookie** con la serializzazione dell'oggetto ...

classi_cookie.php

```
<?php

     require_once("Cookie.php");

     if ($_POST["comando"] == "cancella")
     {
          Cookie::cancella("dato");
     }
     else if ($_POST["comando"] == "imposta")
     {
          $cookie = new Cookie("dato", $_POST["dato"], $_POST["expire"]);
          $cookie->setcookie(true);
     }

?>
<html>
     <head>
          <title>Impostiamo e cancelliamo i Cookie</title>
     </head>
     <body>
```

```
<?php

     if (isset($_COOKIE["dato"]))
     {
          try
          {
               $cookie = Cookie::unserialize("dato");

               echo "Il Cookie \"$cookie->name\" ha il valore =
\"$cookie->value\"";
               echo "<br />\nScade in data " . date("d/m/Y -
H:i:s", $cookie->expire);
               echo "<br />\nE' accessibile dal dominio \"" .
(($cookie->domain) ? $cookie->domain : "default") . "\"";
               echo " per la path \"$cookie->path\"";
               echo "<br />\nsecure = \"" . (($cookie->secure) ?
"true" : "false") . "\"";
               echo "<br />\nhttponly = \"" . (($cookie-
>httponly) ? "true" : "false") . "\"";
          }
          catch (Exception $e)
          {
               echo "<h3>" . $e->getMessage() . "</h3>";
          }

          ?>

          <br /><br />

          <form name="deletecookie" action="classi_cookie.php"
method="post">
               <input type="hidden" name="comando"
value="cancella" />
               <input type="submit" value="Cancella cookie" />
          </form>

          <?php
     }
     else
     {
          ?>

          <form name="writecookie" action="classi_cookie.php"
method="post">
               <table width="400" border="0" cellspacing="5"
cellpadding="5">
                    <tr>
                         <td>Dato da memorizzare</td>
                         <td><input type="text" name="dato"
value="" /></td>
                    </tr>
                    <tr>
                         <td>Scadenza del cookie</td>
                         <td><input type="text" name="expire"
value="" /></td>
                    </tr>
                    <tr>
                         <td><input type="submit" value="Imposta
cookie" /></td>
                         <td> </td>
                    </tr>
               </table>
               <input type="hidden" name="comando"
value="imposta" />
          </form>

          <?php
     }

?>

</body>
</html>
```

Una volta impostato il **cookie** si ottiene questo output.

Tags : Classi Cookie Php 5 Guida
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 25/08/2007 - 7:51

Le Sessioni (Parte 1 di 2)

Il protocollo HTTP è **stateless**, ossia "privo di stato", il che significa che navigando un sito il navigatore effettua delle richieste al server che vengono di volta in volta terminate, rendendo tutte le connessioni "uguali" agli occhi del server senza poter stabilire quali provengono effettivamente dal medesimo utente.

Le **sessioni** sono molto simili ai **cookie** e ci consentono anche essi di tenere in memoria dei dati su un determinato utente.

Per questi motivi i **cookie** e le **sessioni** sono così largamente utilizzati dai Web Developer, rendendo ormai piuttosto raro trovare un navigatore che non accetta i **cookie**.

Tenere traccia dello **stato** di un utente è molto importante per costruire delle applicazioni web efficaci, come ad esempio un sito **e-commerce** dove è necessario "ricordarsi" il contenuto del carrello dell'utente mentre questo naviga il catalogo dei prodotti.

PHP ci fornisce delle funzioni molto utili per assegnare un ID di sessione univoco ad ogni utente, e ci consente di tenerne traccia sostanzialmente in due modi : passando l'ID della sessione negli URL (come accade nell'invio dei dati con **GET**) oppure memorizzandolo in un **cookie** dedicato.

Per motivi di sicurezza vi sconsiglio decisamente il primo metodo, usando le sessioni sempre tramite cookie.

Nelle impostazioni di **default**, **PHP** è settato per memorizzare le **sessioni** in un **cookie**, non dovreste quindi dover apportare alcuna modifica al vostro server.

> A questo punto potremmo anche descrivere una **sessione** come un **cookie** che scade alla fine di una "sessione browser", considerando la fine di una sessione browser come la chiusura di quest ultimo da parte dell'utente.
>
> Un'altra differenza molto importante fra **cookie** e **sessioni** è dove effettivamente vengono memorizzati i dati.
>
> I cookie, come spiegato nei capitoli precedenti, memorizzano i dati nel browser dell'utente (lato client) mentre le **sessioni** le memorizzano direttamente sul server, tenendo nel lato client solamente l'ID della sessione, ID che sarà utilizzato poi per risalire ai dati memorizzati sul server.

Questo rende le **sessioni** molto più indicate quando è necessario gestire dei dati sensibili, dal momento che i **cookie** e il loro contenuto sono sempre accessibili dall'utente e quindi anche dai linguaggi di scripting client side, come ad esempio Javascript, consentendo ai malintenzionati di apportare modifiche al cookie e ai suoi dati mettendo a serio rischio la sicurezza della vostra applicazione.

Usando le **sessioni** inoltre non andrete incontro ai problemi di spazio che avreste con i **cookie**, che come ho scritto due capitoli addietro, qualora un browser rispetti gli standard, non è possibile creare un **cookie** più grande di 4KB per un massimo di 20 **cookie** per dominio e un totale di 200 **cookie** in un browser.

Esempio Sessioni PHP

Vedremo ora un esempio molto semplice dove illustrerò uno scambio di dati fra due pagine mediante l'uso delle **sessioni**.

index.php

```
<?php

     session_start();

     $_SESSION["utente"] = "Mario Rossi";

?>

<a href="test.php">Test</a>
```

test.php

```
<?php

     session_start();

     if (isset($_SESSION["utente"]))
          echo "Utente = " . $_SESSION["utente"];
     else
     {
          echo "Utente non trovato.<br>\n";
          echo "Andare prima <a href=\"index.php\">qui</a>.";
     }

?>
```

Come possiamo osservare nell'esempio soprastante, per utilizzare le **sessioni** è sufficiente chiamare la funzione **session_start()** in tutte le pagine dove si intende accedere ai dati **sessione**, e l'array globale **$_SESSION[]** viene reso automaticamente disponibile.

La prima volta che viene chiamata **session_start()** viene creato un **cookie** di nome **PHPSESSID**, il cui contenuto sarà l'ID di sessione univoco assegnatoci dal server, e che sarà poi utilizzato da quest ultimo per accedere ai dati di **sessione**.

Se state utilizzando Firefox 2, una volta eseguita una delle due pagine di esempio, dalla console '**Mostra i cookie**" avrete modo di leggere il cookie di **sessione**.

Se eseguirete per prima la pagina **index.php**, verrà inizializzata la sessione memorizzandovi il dato "utente" e verrà infine stampato il link per accedere alla seconda pagina ossia **test.php**.

Se invece eseguirete per prima **test.php**, la pagina stamperà una notifica (Utente non trovato) e a seguire un link per **index.php** dove sarà memorizzato il dato "utente".

Riavviando il browser la sessione verrà automaticamente terminata, oppure è possibile eliminarla volontariamente in questo modo :

```
<?php

     session_start();

     unset($_SESSION["utente"]); // Da usare per eliminare un solo dato di
sessione, in questo caso "utente"

     // Da usare per distruggere completamente la sessione
     $_SESSION = array(); // oppure chiamate session_unset();
     session_destroy();

?>
```


Tags : Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 26/08/2007 - 10:14

Le Sessioni (Parte 2 di 2)

In questo capitolo osserveremo più da vicino le **sessioni**, studiando alcuni settaggi e funzioni che ritengo di maggiore rilievo e che ci consentiranno un maggiore controllo sulla nostra applicazione.

Iniziamo dal file di configurazione php.ini, attraverso cui potremo personalizzare il comportamento del server.

- **session.save_path** - La cartella del server dove andranno memorizzati i file di sessione.
- **session.use_cookies** - Di default è impostato a 1, il che significa che il server tenterà sempre di memorizzare l'ID di sessione in un **cookie** nel browser dell'utente.
- **session.use_only_cookies** - Di default è impostato a 0, ma vi consiglio di assegnarlo a 1 in modo da impedire attacchi dove vengono passati gli ID di **sessione** tramite URL. In questo caso però abbiate cura di eseguire sempre dei controlli sull'abilitazione dei **cookie**, e in caso di esito negativo notificate l'utente sulla necessità di avere i **cookie** abilitati per poter usufruire appieno del servizio da voi offerto.
- **session.name** - Il nome di default della sessione, solitamente **PHPSESSID**.
- **session.cookie_lifetime** - E' impostato a 0 di default eliminando il **cookie** di **sessione** alla chiusura del browser. Se avete necessità di rendere disponibile la sessione anche dopo la chiusura del browser allora impostate il parametro con i secondi di durata desiderati.
- **session.cache_expire** - La durata in minuti della **cache** sul server, solitamente impostata a 3 ore.

Se non avete la possibilità di modificare il php.ini del vostro server, oppure avete più applicazioni sullo stesso server con esigenze diverse, allora potete impostare un comportamento diverso per ogni applicazione richiamando direttamente nello script le funzioni native che seguono :

- string **session_save_path** ([string path]) - Se non viene specificato alcun parametro la funzione restituisce la path dove vengono salvati i file di sessione. Se invece specificate un parametro viene cambiato il percorso dove vengono salvati i file con quello nuovo.
- string **session_name** ([string name]) - Restituisce il nome della **sessione** corrente. Se specificato un parametro invece imposta il nome di **sessione** con il nuovo.
- void **session_set_cookie_params** (int lifetime [, string path [, string domain]]) - Il primo parametro è obbligatorio e definisce la durata in secondi del **cookie** di **sessione**. Gli altri due parametri facoltativi cambiano momentaneamente i rispettivi settaggi nel **php.ini**.
- int **session_cache_expire** ([int cache_expire]) - Restituisce la durata in minuti della cache di **sessione**, oppure se impostate un parametro il valore corrente della scadenza cache verrà sostituito col nuovo.
- string **session_id** ([string id]) - Ritorna l'ID di **sessione** corrente oppure se impostate un parametro sostituisce l'id corrente col nuovo.
- void **session_write_close** (void) - Termina la **sessione** corrente e ne archivia i dati.

Per una descrizione dettagliata delle funzioni sopracitate e per l'elenco completo vi rimando alla Documentazione ufficiale di **PHP**.

Attraverso queste funzioni non è però possibile modificare due dei parametri più importanti :

"**session.use_cookies**" e "**session.use_only_cookies**".

Per modificare queste due impostanzioni ricorreremo alla funzione **ini_set()** come mostrato di seguito :

```
<?php

     ini_set('session.use_cookies', 1);
     ini_set('session.use_only_cookies', 1);

?>
```

Prima di chiudere il capitolo vediamo come può tornarci utile la funzione **session_write_close()**, aumentando le prestazioni del server con applicazioni che scrivono grandi quantità di dati per **sessione**.

E' necessario sapere che i dati di **sessione** vengono bloccati per evitare che più script scrivano contemporaneamente in essi, fino a che viene terminato uno script e viene concesso il permesso al successivo.

Di default **PHP** archivia i dati di sessione al termine dello **script**, ma è utile anticiparne l'archiviazione qualora utilizzaste ad esempio i frameset, che verranno caricati uno alla volta proprio a causa di questo blocco.

Non appena avete assegnato tutti i valori ai dati di sessione quindi, sarà bene terminare quest'ultima anticipando l'archiviazione dei dati mediante una chiamata a **session_write_close()**, per consentire l'accesso al prossimo script.

Nel prossimo capitolo vedremo come definire delle funzioni personalizzate per gestire in modo avanzato le **sessioni.**

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 27/08/2007 - 12:37

Gestione avanzata delle Sessioni

Con questo capitolo concludiamo il discorso sulle **Sessioni**, costruendo una libreria esterna che ci consentirà un maggiore controllo sulle operazioni che ne coinvolgono la gestione.

Di default **PHP** salva i dati di **sessione** nella directory /tmp/ che è solitamente accessibile anche dall'esterno, questo potrebbe comportare seri problemi di sicurezza a seconda dei dati che intendete tenere in memoria nelle **sessioni**.

Definirò di seguito dei gestori che saranno automaticamente chiamati da **PHP** al verificarsi di un determinato evento di sessione.

Per definire quale gestore deve essere chiamato per quel determinato evento abbiamo a disposizione la funzione nativa **session_set_save_handler()**.

Prototipo della funzione session_set_save_handler()

```
void session_set_save_handler ( string apertura, string chiusura, string
lettura, string scrittura, string distruzione, string gc )
```

I parametri di questa funzione sono delle stringhe il cui nome rappresenta una funzione definita dall'utente da richiamare al verificarsi di quel determinato evento di **sessione** :

- **apertura** - Chiamata ad inizio sessione da **session_start()**
  Prende due parametri : "**path**" e "**nome**", rispettivamente il percorso per i file di sessione e il nome della sessione corrente appena inizializzata.
  Deve restituire **true** se l'apertura della sessione è avvenuta con successo.
- **chiusura** - Chiamata a fine **sessione**
  Non prende paramentri e deve restituire **true** se la chiusura della sessione è avvenuta con successo.
- **lettura** - Chiamata dopo "apertura" per leggere i dati di **sessione** e renderli disponibili allo script tramite l'array globale **$_SESSION[]**
  Prende solo il parametro "**id**", ossia l'id della sessione corrente che sarà utilizzato per accedere ai dati di sessione.
  Dovrà restituire una stringa che rappresenta la serializzazione dell'array **$_SESSION[]** oppure una stringa vuota.
- **scrittura** - Chiamata quando viene scritto un dato di **sessione** (es. $_SESSION["dato"] = 3)
  Prende due parametri : "**id**" e "**dati**", rispettivamente l'id della sessione e tutti i dati da memorizzare (l'intero contenuto di **$_SESSION[]**).
  Deve restituire **true** se la scrittura è avvenuta con successo.
- **distruzione** - Chiamata da **session_destroy()**

Prende solo il parametro "**id**" ossia l'id della sessione corrente e deve restituire se la distruzione è avvenuta con successo.

- **gc** - **Garbage Collector**, viene chiamata casualmente per controllare che la sessione non sia scaduta
  Prende solo il parametro "**max_lifetime**" che rappresenta la durata della sessione in secondi (default 1440 secondi, quindi 24 minuti).
  Deve restituire **true** per proseguire.

Prima di iniziare a scrivere le funzioni di gestione è necessario confgurare alcuni settaggi per far funzionare correttamente lo script.

Innanzitutto dobbiamo fare in modo che la chiamata a**session_set_save_handler()** vada a buon fine, impostando il parametro "**session.save_handler**" al valore "**user**". Trovate il parametro nel file di configurazione php.ini.

Se non potete accedere al file php.ini o volete settare questa impostazione solo per lo script corrente allora ci affidiamo come sempre alla funzione **ini_set()**.

Vediamo ora il file sorgente della piccola libreria di esempio (sessioni.php) che ho costruito, ovviamente incompleta essendo il suo scopo puramente didattico.

sessioni.php

```
<?php

     // Attiva i cookie di sessione
     ini_set("session.use_cookies", 1);

     // Accetta ID solo da cookie, rifiuta se vengono propagati da URL
     ini_set("session.use_only_cookies", 1);

     // Informa PHP che dovrà usare dei gestori definiti da noi
     ini_set("session.save_handler", "user");

     // Impostiamo la cartella del server che dovrà contenere i file di
sessione
     ini_set("session.save_path", "C:/Appserv/www/test/sessioni");

     define("FILE_PREFISSO", "sess_"); // Inizio nome dei file di sessione
     define("FILE_ESTENSIONE", ".txt"); // Estensione dei file di sessione

     $session_file = NULL;

     function ss_apertura($path, $nome)
     {
          session_cache_expire(60 * 24); // I file di sessione sono validi
per 24 ore
          session_set_cookie_params(60 * 60 * 24); // Il cookie di sessione
dura 24 ore
          return true;
     }

     function ss_chiusura()
     {
          global $session_file;
          return @fclose($session_file); // Chiude il puntatore al file di
sessione aperto
     }

     function ss_lettura($id)
     {
          $id = session_id();
          $filename = session_save_path() . "/" . FILE_PREFISSO . $id .
FILE_ESTENSIONE;

          // Tenta di leggere il contenuto del file. Se il file non esiste
lo crea
          $session_data = @file_get_contents($filename);

          if (strlen($session_data))
          { return $session_data; }
          else
          { return ""; }
     }

     function ss_scrittura($id, $dati)
     {
          global $session_file;

          $id = session_id();
          $filename = session_save_path() . "/" . FILE_PREFISSO . $id .
FILE_ESTENSIONE;

          try
          {
               $session_file = @fopen($filename, "w+");
               @fwrite($session_file, $dati); // Scriviamo nel file i nuovi
dati di sessione

               return true;
          }
          catch (Exception $e)
          { return false; }
     }

     function ss_distruzione($id)
     {
          $filename = session_save_path() . "/" . FILE_PREFISSO . $id .
FILE_ESTENSIONE;
          return @unlink($filename); // Cancella il file
     }

     function ss_spazzino($max_lifetime) { return true; }
```

```
     session_set_save_handler("ss_apertura", "ss_chiusura", "ss_lettura",
"ss_scrittura", "ss_distruzione", "ss_spazzino");
     session_start();

?>
```

Il codice è piuttosto semplice e salva tutti i file di sessione all'interno di file di testo in una cartella "sessioni".

La libreria essendo solo dimostrativa non gestisce alcuna eccezione, ho infatti messo la chiocciola@ prima di quelle funzioni che potrebbero sollevare errori o avvertenze, che farebbero fallire l'invio degli header non avendo bufferizzato l'output con **ob_start()**.

Come potete vedere dalle funzioni **ss_lettura()** e **ss_scrittura**, la serializzazione dei dati viene gestita automaticamente da **PHP**, non vi è quindi alcuna necessità di chiamare altre funzioni, lo scopo di questi due gestori è solo di scrivere e leggere i dati di sessione già pronti.

Di seguito due pagine di esempio : "**index.php**" e "**test.php**", la prima scrive i dati di sessione e la seconda li legge.

index.php

```
<?php

     require_once("sessioni.php");

     $_SESSION["nome"] = "Mario";
     $_SESSION["cognome"] = "Rossi";

?>

<a href="test.php">Test</a>
```

test.php

```
<?php

     require_once("sessioni.php");

     echo "Nome = " . $_SESSION["nome"] . "<br />";
     echo "Cognome = " . $_SESSION["cognome"] . "<br />";

?>
```

Eseguendo **index.php** viene inizializzata la sessione a partire dal **cookie** di sessione che come potete vedere dalla schermata non scadrà alla chiusura del browser ma 24 ore dopo. Il cookie come sempre conterrà l'id di **sessione** che sarà utilizzato per dare un nome univoco al file di sessione (schermata) che conterrà i dati di sessione già serializzati (schermata).

Osservazioni

A mio avviso questo sistema di gestione delle **sessioni** può tornare molto utile e versatile qualora si possedesse un database MySQL, in modo da memorizzare le sessioni nella base di dati, non solo rendendo l'applicazione più sicura, ma consentendovi di condividere i dati delle sessioni anche con applicazioni esterne al server, utilizzando quindi gli stessi dati da più siti contemporaneamente.

Ho creato un esempio semplice con i file invece di usare MySQL, per darvi modo di prendere dimestichezza con queste funzioni.

Vi consiglio pertanto di tornare a dare una spolverata a questo capitolo non appena avrete imparato ad utilizzare un database MySQL, e allora avrete modo di costruirvi una libreria vostra molto più potente e utile di questa.

Il Garbage Collector

Quando definirete una funzione per il **Garbage Collector**, vi sarà utile sapere per la fase di debug che questa funzione verrà chiamata casualmente in base a dei parametri contenuti nel file di configurazione php.ini :

- **session.gc_maxlifetime** - Specifica i secondi dopo i quali i dati non saranno più validi
  Di default è impostato a 1440 secondi. Vi consiglio di tenere in memoria da qualche parte (es. variabile globale) il timestamp di quando è nata la sessione, in modo da verificare rapidamente con un semplice confronto se la sessione è scaduta, sommando al timestamp di nascita il valore **maxlifetime** passato come parametro al **Garbage Collector**. Se sarà minore del timestamp corrente allora la sessione è scaduta.
- **session.gc_divisor** - Usato per il calcolo della probabilità che venga chiamato il **Garbage Collector**
  Legato a "**session.gc_probability**", vedi sotto.
- **session.gc_probability** - Usato per il calcolo della probabilità che venga chiamato il **Garbage Collector**
  I parametri "**gc_divisor**" e "**gc_probability**" sono di default settati rispettivamente a 100 e 1.
  Questo significa che c'è una probabilità dell'1% che venga chiamato il garbage collector, o meglio ogni 100 esecuzioni dello script.

In fase di debug vi consiglio quindi di impostare i parametri come specificato di seguito :

- **session.gc_lifetime** - 60 secondi invece di 1440, così non dovrete aspettare 24 minuti ogni volta che controllate se "pulisce"
- **session.gc_divisor** - Lasciate il valore inalterato a 100
- **session.gc_probability** - Impostate il valore a 100 come **gc_divisor**, in modo che il Garbage Collector venga chiamato ad ogni esecuzione dello script e vi dia il modo di verificarne immediatamente il funzionamento

Anche queste impostazioni possono essere modificate tramite **ini_set()**.

Nota : ricordatevi che potete usare **$_SESSION[]** come un normale array, memorizzandovi quindi anche dati complessi :

```
<?php

     $_SESSION["utente"] = new Utente("Mario", "Rossi");

?>
```

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 28/08/2007 - 15:01

Upload di file

L'invio del file avviene in modo analogo all'invio di dati classico che abbiamo già visto con i metodi**POST** e **GET**.

Bisogna sapere che l'invio di un qualsiasi dato corrisponde ad una determinata richiesta HTTP, ognuna per gestire un tipo di dato diverso, il che significa che se abbiamo una normale pagina HTML con all'interno 3 immagini, avremo 4 richieste HTTP da parte del browser come segue :

```
Richieste HTTP :

Content-Type: text/html
Content-Type: image/jpeg
Content-Type: image/jpeg
Content-Type: image/jpeg
```

Per questo motivo quando dobbiamo inviare un file sono necessari alcuni accorgimenti in più rispetto ad un normale dato, a partire da due principali modifiche nella form HTML di invio :

- **enctype** - E' obbligatorio specificare questo attributo nella form HTML assegnandolo al valore "**multipart/form-data**"
- **file** - E' necessario inserire un campo **input** del tipo **file**

Form HTML per l'invio di file

Vediamo la pagina **HTML** che si occuperà dell'invio del file :

```
<!-- invio_file.html -->
<html>
     <head>
          <title>Upload di file</title>
     </head>
     <body>
          <form enctype="multipart/form-data" action="upload.php"
method="post">
               <input type="file" name="immagine" />
               <br />
               <input type="submit" value="Invia file" />
          </form>
     </body>
</html>
```

Come possiamo vedere nella pagina d'esempio soprastante, il codice è molto simile a quello delle form viste nei capitoli precedenti, con la presenza però di **enctype** e utilizzando **file** come tipo di input, che sarà trasformato dal browser in un campo vuoto con un bottone la cui etichetta è solitamente "Sfoglia", ma che viene automaticamente sostituita a seconda della lingua del browser senza bisogno di un nostro intervento.

Il file inviato sarà poi accessibile alla pagina inviata tramite l'array globale **$_FILES[]** con una lista di elementi predefinita :

- **$_FILES["name"]** - Il nome del file inviato (es. "foto1.jpg")
- **$_FILES["type"]** - Il tipo del file inviato in formato MIME (es. "image/jpeg")
- **$_FILES["tmp_name"]** - Il nome del file temporaneo che risiede sul server (es. "/tmp/php8K45x") Una volta inviato il file, questo sarà memorizzato in una cartella temporanea del server, e sarà cancellato automaticamente a fine richiesta. Se quindi non gestite il file prima della fine dello script, perderete il file inviato.
- **$_FILES["error"]** - Contiene un codice di errore corrispondente a un evento verificatosi durante l'invio del file
- **$_FILES["size"]** - La dimensione in **byte** del file inviato

PHP ci mette a disposizione molte funzioni per gestire il file inviato, le due che ci torneranno più utili sono :

- is_uploaded_file() - Controlla se il file è stato effettivamente inviato
- move_uploaded_file() - Sposta il file inviato nella cartella specificata

E' essenziale sapere inoltre i tipi di errore che possiamo gestire per poter creare uno script funzionale.

Come abbiamo visto prima, il codice di errore viene reso accessibile attraverso l'array globale **$_FILES[]** e il suo attributo "**error**", che conterrà un numero intero corrispondente all'evento verificato (Lista completa dei messaggi di errore).

Torniamo ora alla form HTML dell'esempio, che invierà il file (name="immagine") alla pagina **upload.php** utilizzando il metodo **POST**.

upload.php

Prima di costruire uno script più robusto e flessibile, vediamo uno script leggero che riceverà il file, lo sposterà in una cartella chiamata "uploads" e mostrerà infine l'immagine appena inviata :

```
<?php

     $immagine = $_FILES["immagine"];
     move_uploaded_file($immagine["tmp_name"], "C:/Appserv/www/test/
uploads/" . $immagine["name"]);

?>

<img src="<?= "uploads/" . $immagine["name"] ?>" />
```

Ho assegnato l'elemento di **$_FILES[]** ad un'altra variabile solo per comodità, sono infatti equivalenti le due **echo** che seguono :

```
<?php

     $immagine = $_FILES["immagine"];
     echo "Il nome del file è inviato è " . $immagine["name"];
     echo "Il nome del file è inviato è " . $_FILES["immagine"]["name"];

?>
```

Lo script mostrato è fin troppo semplice e soggetto a molte complicazioni, non effettua infatti nessun controllo sulla dimensione nè sul tipo del file inviato, avremmo quindi potuto inviare un file di testo che lo script avrebbe comunque tentato di mostrarlo attraverso il tag html '**img**".

Nel prossimo capitolo vedremo invece un esempio più funzionale e robusto con la programmazione orientata agli oggetti.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 29/08/2007 - 17:24

Classi per l'upload di file

In questo capitolo vedremo un esempio più complesso del precedente, ma che ci fornirà un'interfaccia più intuitiva e sicura per gestire l'upload dei file sui nostri server.

Per questo scopo ho creato due classi a cui vi consiglio più di applicare i concetti di ereditarietà e polimorfismo per creare poi le vostre classi per una libreria personalizzata più adeguata alle esigenze della vostra applicazione.

Prima di introdurre il codice della libreria è necessario sapere che esiste uno specifico tipo **MIME** per ogni tipo di file inviato.

Come abbiamo visto nel capitolo precedente, una volta inviato il file questo è accessibile attraverso l'array globale **$_FILES[]**, in particolare con la chiave "**type**" è possibile accedere alla stringa che descrive che tipo di file è stato inviato.

Se "**type**" è uguale a "image/jpeg" abbiamo ricevuto una jpeg, se è uguale a "audio/x-mpeg-3" abbiamo ricevuto un file **mp3** e così via ...
Una lista dettagliata dei tipi di file con i rispettivi **MIME**.

Iniziamo dalla classe "**ErroreFile**" per una gestione mirata delle eccezioni, che si appoggerà alle costanti globali degli errori di upload che abbiamo già visto nel capitolo precedente.

Nella libreria che segue ho definito inoltre altre costanti per gestire delle eccezioni non previste dal set di costanti globali.

Lo scopo principale della **classe** che segue è quello di fornire un set di messaggi d'errore tradotti in italiano, attraverso un array statico. Derivando da questa classe potrete quindi fornire dei messaggi in altre lingue o messaggi di errore aggiuntivi per gestire eccezioni particolari, personalizzate per le vostre applicazioni.

ErroreFile.php

```
<?php

      define("UPLOAD_ERR_MOVE", 7);
      define("UPLOAD_ERR_NO_FORM", 8);
      define("UPLOAD_ERR_NO_TYPE", 9);
      define("UPLOAD_ERR_UNKNOWN", 10);

      class ErroreFile extends Exception
      {
            private $errore;
            static $tipo = array(
                  UPLOAD_ERR_OK => "File inviato con successo",
                  UPLOAD_ERR_INI_SIZE => "Dimensione del file troppo grande",
                  UPLOAD_ERR_FORM_SIZE => "Dimensione del file troppo grande",
                  UPLOAD_ERR_PARTIAL => "File ricevuto parzialmente",
                  UPLOAD_ERR_NO_FILE => "Nessun file ricevuto",
                  UPLOAD_ERR_NO_TMP_DIR => "Cartella temporanea non trovata",
                  UPLOAD_ERR_MOVE => "Impossibile archiviare il file",
                  UPLOAD_ERR_NO_FORM => "Nessun file specificato",
                  UPLOAD_ERR_NO_TYPE => "Formato file non consentito",
                  UPLOAD_ERR_UNKNOWN => "Errore sconosciuto"
            );

            public function ErroreFile($err_no)
            { $this->errore = (int) $err_no; }

            public function messaggio()
            {
                  foreach (ErroreFile::$tipo as $chiave => $valore)
                        if ($this->errore == $chiave)
                              return $valore;

                  return ErroreFile::$tipo[UPLOAD_ERR_UNKNOWN];
            }
      }

?>
```

Ho aggiunto 4 costanti per descrivere altri tipi di eccezioni che potrebbero verificarsi, facendo attenzione a non utilizzare i codici già presi dalle altre costanti predefinite, ossia 0, 1, 2, 3, 4 e 6 (non chiedetemi che fine ha fatto il 5).

Vediamo ora la classe **File** che si occuperà di gestire il file inviato e di controllare che sia compatibile con le nostre esigenze.

Classe File

A inizio sorgente dichiarerò anche qui un set di costanti globali, per non dover ricordare il tipo **MIME** dei file che consideriamo validi per la nostra applicazione. Per il mio esempio ho scelto questi :

- **GIF** - "image/gif"
- **JPEG** - "image/jpeg"
- **PNG** - "image/png"
- **PSD** - "image/psd"
- **BMP** - "image/bmp"
- **TIFF** - "image/tiff"
- **FLASH** - "application/x-shockwave-flash"

Un'occhiata veloce agli attributi e ai metodi della **classe File** :

- **private $file** - L'array che conterrà solo il file specificato
- **private $cartella** - La cartella remota dove andrà memorizzato il file inviato

- **public function File()** - Il costruttore della **classe** che prenderà in input il nome del file da gestire e la cartella di destinazione
- **public function tipo()** - Metodo che prende un numero di parametri variabile, ossia tutti i **tipi** di file che consideriamo validi
- **public function dimensione()** - La dimensione massima accettata per il file inviato. Prende due parametri, il primo la dimensione massima e il secondo l'unità di misura per valutare la dimensione, ossia "byte", "kb" o "mb" (default "**byte**")
- **public function sposta()** - Sposta il file nella cartella di destinazione
- **public function immagine()** - Restituisce la stringa già formattata in HTML per visualizzare l'immagine inviata
  Prima di usare verificare che il file sia un'immagine con il metodo **tipo()**
- **public function __get()** - Restituisce il valore dell'attributo specificato
- **public function __toString()** - Restituisce il nome originale del file

File.php

```
<?php

     require_once("ErroreFile.php");

     // MIME Types
     define("GIF", "image/gif");
     define("JPEG", "image/jpeg");
     define("PNG", "image/png");
     define("PSD", "image/psd");
     define("BMP", "image/bmp");
     define("TIFF", "image/tiff");
     define("FLASH", "application/x-shockwave-flash"); // Filmato Adobe
Flash

     class File
     {
          private $file;
          private $cartella;

          public function File($nome, $cartella = "/uploads/")
          {
               if (isset($_FILES[$nome]))
               {
                    $this->file = $_FILES[$nome];
                    $this->cartella = $cartella;
               }
               else
                    throw new ErroreFile(UPLOAD_ERR_NO_FORM);
          }

          public function tipo()
          {
               if (!func_num_args()) throw new
ErroreFile(UPLOAD_ERR_NO_TYPE);

               $tipi = func_get_args();

               foreach ($tipi as $chiave => $valore)
                    if ($this->file["type"] == $valore)
                         return true;

               throw new ErroreFile(UPLOAD_ERR_NO_TYPE);
          }

          public function dimensione($dim, $unita = "byte")
          {
               if (strcasecmp($unita, "byte") == 0) $dim = (int) $dim;
               else if (strcasecmp($unita, "kb") == 0) $dim = (int) $dim *
1024;
               else if (strcasecmp($unita, "mb") == 0) $dim = (int) $dim *
1024 * 1024;
               else $dim = 0;

               if ($this->file["size"] > $dim) throw new
ErroreFile(UPLOAD_ERR_FORM_SIZE);
          }

          public function sposta()
          {
               $ok = @move_uploaded_file($this->file["tmp_name"], $this-
>cartella . $this->file["name"]);

               if (!$ok) throw new ErroreFile(UPLOAD_ERR_MOVE);
          }

          public function immagine($dir, $alt = NULL)
          {
               $dimensioni = getimagesize($dir . $this->file["name"]);
               $dimensioni = $dimensioni[3]; // height="yyy" width="xxx"

               $imgtag = "<img src=\"$dir" . $this->file["name"] . "\" ";
               $imgtag .= "$dimensioni ";
               $imgtag .= (strlen($alt) ? "alt=\"$alt\"" : "") . " />"; //
Imposta l'attributo ALT per l'immagine

               return $imgtag;
          }

          public function __get($attributo)
          {
               if (array_key_exists($attributo, $this->file))
                    return $this->file[$attributo];

               return false;
          }

          public function __toString()
          { return $this->file["name"]; }
```

```
	}

?>
```

Per l'esempio ho utilizzato alcune funzioni native che non avevo ancora spiegato, di seguito i nomi e la relativa documentazione :

- **strcasecmp()** - Documentazione
- **getimagesize()** - Documentazione
- **array_key_exists()** - Documentazione

A seguire la pagina **upload.php** dell'esempio precedente, riscritta per l'uso delle nuove classi.

upload.php

```
<?php

	require_once("File.php");

	try
	{
		$file = new File("immagine", "C:/Appserv/www/test/uploads/");
		$file->tipo(JPEG, PNG); // Accettate solo immagini in formato
JPEG o PNG
		$file->dimensione(100, "kb"); // Solo immagini più piccole di
100Kb
		$file->sposta(); // Sposta il file inviato nella cartella di
destinazione

		echo "Nome file : $file<br /><br />";
		echo $file->immagine("uploads/", "Immagine inviata"); //
Visualizza l'immagine appena inviata
	}
	catch (ErroreFile $e)
	{ echo $e->messaggio(); }
	catch (Exception $e)
	{ echo $e->getMessage(); }

?>
```

Il risultato ottenuto inviando un'immagine che soddisfa i parametri dell'esempio.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 30/08/2007 - 19:47

Inviare email

Inviare una e-mail con **PHP** è molto semplice, ma costruendo applicazioni di un certo livello vi occorrerà senzaltro dover inviare e-mail in formati particolari e con uno o più allegati.

Per fare questo useremo la funzione nativa **mail()**, che si può tranquillamente gestire dando un'occhiata alla documentazione ufficiale.

Lo scopo principale di questo capitolo quindi, è quello di fornirvi le conoscenze per poter gestire anche e-mail più complesse, ad esempio in formato HTML con più file allegati.

Vediamo ora il prototipo della funzione **mail()** :

```
bool mail ( string destinatario, string oggetto, string messaggio [, string header_addizionali [, string parametri_addizionali]] )
```

Per inviare una e-mail standard è quindi sufficiente il codice che segue :

```
<?php

     $inviata = mail("john.doe@domain.com", "Oggetto del messaggio", "Testo del messaggio");

     if ($inviata) echo "Email inviata con successo";
     else echo "Errore durante l'invio dell'email";

?>
```

Per inviare e-mail più complesse invece, sarà necessario specificare un'intestazione (**header**) passandola per parametro alla funzione **mail()**.

Nell'header dell'e-mail è possibile specificare dei parametri per stabilire svariate impostazioni, vediamo le principali :

- **MIME-Version** - Indica quale standard viene usato
- **Content-Type** - Indica di che tipo è il contenuto del messaggio
- **Content-Transfer-Encoding** - Stabilisce il metodo da utilizzare per rappresentare i dati (es. 7bit)
- **From** - Indica il mittente del messaggio
- **Reply-To** - Specifica l'e-mail predefinita a cui rispondere
- **Cc** - Indirizzi a cui inviare l'e-mail in copia carbone
- **Bcc** - Indirizzi a cui inviare l'e-mail in copia carbone
- **Date** - Data di invio dell'e-mail
- **X-Mailer** - Software utilizzato per l'invio dell'e-mail

Il codice dell'esempio seguente si occuperà di inviare una e-mail in formato HTML, impostando come mittente l'indirizzo mittente@dominio.it :

```
<?php

     define("EOL", "\r\n");

     $header = "MIME-Version: 1.0" . EOL;
     $header .= "Content-Type: text/html" . EOL;
     $header .= "From: mittente@dominio.it";

     $object = "Oggetto del messaggio";

     $message = "Linea 1<br />Linea 2<br />Linea 3";

     mail("posta@realizzazione-sito.info", $object, $message, $header);

?>
```

Per garantire un corretto funzionamento dello script è necessario separare i parametri dell'header con un **EOL** (End Of Line), che deve corrispondere ai due caratteri di escape **\r\n**, anche se alcuni sistemi Unix potrebbero funzionare con il solo **\n**.

Eseguendo lo script soprastante si ottiene questo risultato.

E' possibile inoltre specificare più destinatari per la stessa e-mail senza agire sull'header, ma semplicemente separando gli indirizzi con una virgola.

```
<?php

     $destinatari = "Mario Rossi <mario.rossi@dominio.it>, Luca Verdi <luca.verdi@dominio.it>";
     $destinatari .= ", John Doe <john.doe@domain.com>";

     mail($destinatari, "Oggetto", "Messaggio");

?>
```

Non è obbligatorio racchiudere gli indirizzi fra i tag **< >**, ma è necessario se volete assegnare un nome al proprietario dell'indirizzo.

Nel prossimo capitolo vedremo come è possibile inviare e-mail con file allegati.

Tags : Inviare Email Guida Php 5
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 31/08/2007 - 22:11

Inviare email con allegati

Inviare email con allegati è relativamente semplice dal punto di vista comprensivo, ma al momento in cui scrivo non sono riuscito a trovare una documentazione chiara e semplice in italiano, allora ho deciso di scrivere questo capitolo che comunque non fornirà una documentazione completa e dettagliata sull'argomento, ma vi consentirà di capire il meccanismo per realizzare degli script **PHP** funzionanti, anche se non escludo dei possibili problemi di compatibilità con alcuni client di posta.

Per allegare dei file ad una email, il nostro script differirà principalmente per due fattori.

Inizialmente dovremo impostare il formato dell'email come **multipart/mixed** invece dei classici **text/plain** o **text/html**, questo per indicare che l'email sarà divisa in sottoparti di contenuto diverso (es. testo del messaggio in HTML + file allegato).

Per fare questo è sufficiente impostare correttamente il **Content-Type** nell'header utilizzando una codifica a 7 o a 8 bit (Content Transfer Encoding).

Fatto questo procediamo indicando un **boundary**, ossia una stringa che utilizzeremo per indicare la fine e l'inizio di ogni sottoparte all'interno del corpo del messaggio.

Non è necessario impostare altri parametri nell'header, procediamo quindi inserendo i dati nel corpo del messaggio, separando correttamente i contenuti dai file allegati attraverso il **boundary** sopracitato preceduto da due trattini **--**.

Per ogni sottoblocco nel corpo del messaggio dovremo indicare il **Content-Type** adatto, e nel caso in cui il sottoblocco sia un file allegato sarà necessario utilizzare la codifica **Base64** (Content-Transfer-Encoding: base64).

```
<?php

     define("EOL", "\r\n");

     $boundary = "a08gunwtggjga0a89hg0a72hg93"; // Stringa generata
premendo tasti a caso

     $header = "MIME-Version: 1.0" . EOL;
     $header .= "Content-Type: multipart/mixed; boundary=\"$boundary\"" .
EOL;
     $header .= "Content-Transfer-Encoding: 8bit" . EOL;
```

```
     $object = "Oggetto del messaggio";

     /* Leggo il contenuto del file da allegare e lo memorizzo in $allegato
*/
     $file = @fopen("immagine.jpg", "r");
     $allegato = fread($file, filesize("immagine.jpg"));
     $allegato = base64_encode($allegato);

     /* Inserisco il testo del messaggio in formato HTML */
     $message = "--" . $boundary . EOL;
     $message .= "Content-Type: text/html; charset=iso-8859-1" . EOL;
     $message .= "Content-Transfer-Encoding: 8bit" . EOL;
     $message .= "TESTO DEL MESSAGGIO IN FORMATO HTML" . EOL;
     $message .= "--" . $boundary . EOL;

     /* Ora inserisco un nuovo sottblocco che conterrà il file allegato */
     $message .= "--" . $boundary . EOL;
     $message .= "Content-Type: image/jpeg; name=\"immagine.jpg\"" . EOL;
     $message .= "Content-Transfer-Encoding: base64" . EOL;
     $message .= "Content-Description: Immagine" . EOL;
     $message .= "Content-Disposition: attachment; filename=\"immagine.jpg
\"" . EOL;
     $message .= $allegato . EOL;
     $message .= "--" . $boundary . EOL;

     mail("posta@realizzazione-sito.info", $object, $message, $header);

?>
```

Eseguendo lo script soprastante viene prodotto questo risultato.

Lo script è stato testato correttamente con GMail, ma potrebbe non funzionare correttamente con altri client di posta.
In questi casi provate a cambiare l'**EOL** o ad esempio la sintassi del **boundary** di chiusura blocco.

Tags : Email Allegati Php 5 Guida
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 02/09/2007 - 0:34

Classi per inviare email (Parte 1 di 2)

In questo capitolo costruiremo una libreria che ci consentirà di inviare email complesse, ad esempio in HTML con uno o più allegati, in modo molto trasparente e veloce.

Questa libreria non è completa anche se funziona correttamente. Non ho infatti implementato alcuna gestione delle eccezioni che vi consiglio pertanto di aggiungere qualora decidiate di usarla nello sviluppo di applicazioni reali.
Inoltre sarebbe stato utile interfacciare la libreria con la classe '**File**" vista nei capitoli precedenti, ma ho preferito comunque non inserire altre classi esterne essendo lo scopo di questa libreria puramente didattico.

Vediamo ora le **Classi** che compongono questa nuova libreria e il loro scopo :

- **MIME** - Classe usata come contenitore di **costanti** per definire i tipi MIME supportati dall'applicazione
- **MailBlock** - Rappresenta un sottoblocco del messaggio email, quest ultimo sarà quindi composto da una o più istanze di **MailBlock**
- **Allegato** - Derivata da **MailBlock**, rappresenta un file allegato ossia un sottoblocco più complesso con codifica **Base64**
- **Email** - Classe principale creata per fornire un'interfaccia fra tutte le altre classi, costruendo il messaggio email completo e inviandolo

Classe : "MIME"
Nome file : "MIME.php"

```
<?php

     class MIME
     {
          const HTML = "text/html";
          const TEXT = "text/plain";
          const MULTI = "multipart/mixed";

          const GIF = "image/gif";
          const JPEG = "image/jpeg";
          const PNG = "image/png";
          const PSD = "image/psd";
          const BMP = "image/bmp";
          const TIFF = "image/tiff";
          const FLASH = "application/x-shockwave-flash";
     }

?>
```

Classe : "MailBlock"
Nome file : "MailBlock.php"

```
<?php

     define("EOL", "\r\n");

     class MailBlock
     {
          protected $content_type;
          protected $charset;
          protected $content_transfer_encoding;

          public $content;
          protected $boundary;

          public function MailBlock($content_type, $boundary, $content,
$charset = "iso-8859-1", $c_t_encoding = "8bit")
          {
               $this->content_type = $content_type;
               $this->charset = $charset;
               $this->content_transfer_encoding = $c_t_encoding;

               $this->content = $content;
               $this->boundary = $boundary;
          }

          public function __toString()
          {
               $content = "--" . $this->boundary . EOL;
               $content .= "Content-Type: " . $this->content_type . ";
charset=" . $this->charset . EOL;
               $content .= "Content-Transfer-Encoding: " . $this-
>content_transfer_encoding . EOL;
               $content .= $this->content . EOL;
               $content .= "--" . $this->boundary . EOL;

               return $content;
          }
     }

?>
```

La classe proposta è piuttosto semplice, non fa altro che collezionare i dati necessari a rappresentare un sottoblocco e ad organizzarli attraverso il metodo **__toString()**.
Il codice che segue inizializza correttamente un sottoblocco in formato HTML :

```
<?php
```

```
	require_once("MIME.php");
	require_once("MailBlock.php");

	$boundary = "fa0s7u98hg87ngf0hgk05695j";
	$contenuto = "<ul><li>Linea uno</li><li>Linea due</li></ul>";

	$blocco = new MailBlock(MIME::HTML, $boundary, $contenuto);

	echo $blocco; // Richiama "__toString()" stampando il sottoblocco con
la sintassi corretta

?>
```

Classe : "Allegato"
Nome file : "Allegato.php"

```
<?php

	require_once("MailBlock.php");

	class Allegato extends MailBlock
	{
		public $url;
		public $name;
		public $description;

		public function Allegato($name, $url, $content_type, $boundary,
$description = NULL)
		{
			$this->url = $url;
			$this->name = $name;
			$this->boundary = $boundary;
			$this->description = $description;

			$this->content = $this->leggi();
			parent::MailBlock($content_type, $boundary, $this->content,
"utf-8", "base64");
		}

		private function leggi()
		{
			$file = @fopen($this->url, "r");
			$allegato = fread($file, filesize($this->url));
			return base64_encode($allegato);
		}

		public function __toString()
		{
			$content = "--" . $this->boundary . EOL;
			$content .= "Content-Type: " . $this->content_type . ";
name=\"" . $this->name . "\"" . EOL;
			$content .= "Content-Transfer-Encoding: " . $this-
>content_transfer_encoding . EOL;
			$content .= "Content-Description: " . $this->description .
EOL;
			$content .= "Content-Disposition: attachment; filename=\"" .
$this->name . "\"" . EOL;
			$content .= $this->content . EOL;
			$content .= "--" . $this->boundary . EOL;

			return $content;
		}
	}

?>
```

La **classe** soprastante deriva dalla precendente **MailBlock**, aggiungendo però tre importanti parametri necessari per gestire il file allegato, ossia l'**url**, il **nome** del file che sarà visualizzato dal destinatario e infine una **descrizione** facoltativa.

Il metodo privato **leggi()** si occupa invece di leggere il file specificato, per poi inserirlo nel sottoblocco in questione usando la codifica **Base64**.

Il metodo pubblico **__toString()** si comporta come il suo genitore, organizzando però il sottoblocco con i tre parametri aggiuntivi e inserendo alla fine il contenuto del file specificato.

Nel prossimo capitolo vedremo l'oggetto **Email**, l'ultima **classe** della nostra libreria con del codice d'esempio.

Tags : Guida Php 5 Classi Inviare Email Allegati
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 03/09/2007 - 2:57

Classi per inviare email (Parte 2 di 2)

In questa lezione vedremo la **classe Email**, analizzandone brevemente metodi e attributi ed infine un esempio per l'utilizzo della nuova libreria con l'invio di due allegati.

Attributi

- **private $to_mail** - Array contenente la lista dei destinatari.
- **private $object** - Oggetto dell'email.
- **private $message** - Array contenente tutti i sottoblocchi che formeranno il messaggio, ossia istanze di "**MailBlock**" e "**Allegato**".
- **public $mime** - Versione **MIME** utilizzata per lo standard dell'email.
- **public $content_type** - **Content-Type** per identificare il contenuto del messaggio (es. "**text/html**" oppure "**multipart/mixed**" ...).
- **private $boundary** - Delimitatore **boundary** generato criptando il timestamp Unix corrente.
- **public $cc** - Stringa contenente gli indirizzi per l'invio dell'email in copia carbone. Consiglio di rendere **private** l'attributo e di trasformarlo in un array, costruendo un metodo per un inserimento controllato.
- **public $bcc** - Stringa contenente gli indirizzi per l'invio dell'email in copia carbone. Anche qui consiglio di agire come con l'attributo **$cc**.
- **public $date** - Data di invio email (tipo stringa).
- **public $from** - Indirizzo email del mittente, consiglio anche qui di effettuare la stessa modifica descritta per l'attributo **$cc**.
- **public $reply_to** - Specifica l'email predefinita a cui il destinatario dell'email potrà rispondere. Anche qui stesso consiglio.
- **public $xmailer** - Stringa che identifica il software utilizzato per l'invio dell'email.

Metodi

- **public** Email() - Costruttore della classe. Inizializza **$object**, **$content_type** e altri attributi.
- **public** blocco() - Aggiunge un sottoblocco standard al corpo del messaggio, ossia un'istanza **MailBlock**.
- **public** allegato() - Aggiunge un sottoblocco più complesso al corpo del messaggio che rappresenterà un file allegato, ossia un'istanza di **Allegato**.
- **public** destinatario() - Aggiunge un indirizzo alla lista dei destinatari.
- **private** header() - Metodo privato che costruisce l'header dell'email basandosi sui parametri memorizzati negli attributi della classe.
- **public** invia() - Metodo per l'invio dell'email. Restituisce **true** o **false** per identificare se l'invio è avvenuto con successo.

Classe : "Email"

Nome file : "Email.php"

```
<?php

     require_once("MIME.php");
     require_once("Allegato.php");

     class Email
     {
          private $to_mail;
          private $object;
          private $message;

          public $mime = "1.0";
          public $content_type;
          private $boundary = NULL;

          public $cc = NULL;
          public $bcc = NULL;
          public $date = NULL;
          public $from = NULL;
          public $replyto = NULL;
          public $xmailer = NULL;

          public function Email($object, $content_type = MIME::TEXT)
          {
               $this->to_mail = array();
               $this->object = $object;
               $this->message = array();

               $this->boundary = md5(time());
               $this->content_type = $content_type;
          }

          public function blocco($content_type, $content, $charset =
"iso-8859-1", $c_t_encoding = "8bit")
          {
               $succ = count($this->message);
               $this->message[$succ] = new MailBlock($content_type, $this-
>boundary, $content, $charset, $c_t_encoding);
          }

          public function allegato($name, $url, $mime_type, $description =
NULL)
          {
               $succ = count($this->message);
               $this->message[$succ] = new Allegato($name, $url,
$mime_type, $this->boundary, $description);

               /* Se questo metodo viene richiamato significa che è stato
inserito almeno un
```

```
                allegato quindi per sicurezza modifico il Content-Type a
"multipart/mixed" */
                $this->content_type = MIME::MULTI;
          }

          public function destinatario($to_mail)
          {
                array_push($this->to_mail, $to_mail);
          }

          private function header()
          {
                $header = "MIME-Version: " . $this->mime . EOL;
                $header .= "Content-Type: " . $this->content_type . ";
boundary=\"" . $this->boundary . "\"" . EOL;
                $header .= "Content-Transfer-Encoding: 8bit" . EOL;

                if ($this->from != NULL) { $header .= "From: " . $this-
>from . EOL; }
                if ($this->replyto != NULL) { $header .= "Reply-To: " .
$this->replyto . EOL; }
                if ($this->cc != NULL) { $header .= "Cc: " . $this->cc .
EOL; }
                if ($this->bcc != NULL) { $header .= "Bcc: " . $this->bcc .
EOL; }
                if ($this->date != NULL) { $header .= "Date: " . $this-
>date . EOL; }
                if ($this->xmailer != NULL) { $header .= "X-Mailer: " .
$this->xmailer; }

                return $header;
          }

          public function invia()
          {
                $message = "";
                $blocchi = count($this->message);

                for ($i = 0; $i < $blocchi; $i++)
                      $message .= $this->message[$i]; // Richiama il metodo
__toString() di "Allegato" o "MailBlock"

                $to = implode(", ", $this->to_mail);
                return mail($to, $this->object, $message, $this->header());
          }
     }

?>
```

Come potete vedere anche la **classe Email** è piuttosto semplice dovendo principalmente fungere da interfaccia unendo i dati.

Con questa **classe** la nostra libreria è al completo. Per ultimo vedremo un piccolo foglio **PHP** come esempio di utilizzo della libreria, inviando un'email con un blocco di testo, un blocco in HTML e 2 file allegati.

test.php

```
<?php

     require_once("Email.php");

     $mail = new Email("Mail con allegati", MIME::MULTI);

     /* Codice per aggiungere i destinatari */
     $mail->destinatario("RS Staff <posta@realizzazione-sito.info>");
     //$mail->destinatario("Utente1 <utente1@dominio.it>");
     //$mail->destinatario("Utente2 <utente2@dominio.it>");

     $mail->from = "Mittente <email_di_invio@gmail.com>";
     $mail->replyto = "Risposta <email_per_risposta@gmail.com>";

     $mail->blocco(MIME::TEXT, "Iniziamo con un pò di testo nel primo
blocco!!");

     /* Memorizzo l'html per il secondo blocco in una
     variabile per favorire la leggibilità del codice */
     $html = "<ul><li>Linea uno</li><li>Linea due</li></ul>";
     $html .= "<table border=\"1\"><tr><td>topolino<td><td>minnie</td></
tr></table>";

     $mail->blocco(MIME::HTML, $html);
     $mail->allegato("litfiba.jpg", "allegati/file.jpg", MIME::JPEG);
     $mail->allegato("220 volts.jpg", "allegati/220v.jpg", MIME::JPEG);

     $inviata = $mail->invia();

     if ($inviata) { echo "Mail inviata con successo!!"; }
     else { echo "Mail non inviata!!"; }

?>
```

Il risultato ottenuto.

Anche in questo caso ho testato il codice solo con **GMail**, non escludo quindi problemi di compatibilità con altri siti o client di posta (Outlook, Eudora ...), e dovendo riscontrarne vi consiglio di provare a modificare l'**EOL** o il **boundary** di chiusura.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 04/09/2007 - 5:21

I Database

In questo capitolo vedremo le funzioni che **PHP 5** ci mette a disposizione per gestire il proprio database **MySQL**.

Non è scopo di questa lezione spiegarvi cosa è un **Database** e come funziona, pertanto vi consiglio di apprendere queste informazioni prima di procedere con la lettura di questo capitolo.

Per iniziare vi consiglio due ottimi documenti in italiano su **Wikipedia** :

- **Database Management System**
- **Modello relazionale**

Prima di iniziare ad analizzare le funzioni base di **PHP** vi consiglio di procurarvi **phpMyAdmin** (chi ha installato **AppServ** lo ha già incluso nel pacchetto) e di installarlo nel vostro server, poichè vi permetterà di gestire il vostro **database** in modo molto intuitivo e veloce.

Al riguardo una serie di articoli utili e interessanti :

- phpMyAdmin - Sito ufficiale con documentazione e download ultima versione
- Wikipedia - Articolo in italiano su **phpMyAdmin** con link a risorse utili
- Guida HTML.it - Guida all'utilizzo di questo **tool** (in lingua italiana)

Funzioni

- **mysql_connect()** - Apre una connessione con il server MySQL specificato.
- **mysql_close()** - Chiude la connessione con il server MySQL.
- **mysql_pconnect()** - Apre una connessione persistente con il server MySQL specificato, non necessita di un'antagonista "**mysql_pclose()**" per la chiusura della connessione.
- **mysql_select_db()** - Seleziona il database MySQL specificato.
- **mysql_query()** - Esegue una **query MySQL**.
- **mysql_num_rows()** - Restituisce il numero di righe prodotto da una **query**.
- **mysql_fetch_array()** - Trasforma una riga del risultato di una query in un array.

Per un elenco completo e dettagliato vi rimando alla documentazione ufficiale.

Vediamo ora un esempio di utilizzo con le funzione sopracitate, prendendo in esame una tabella "**impiegati**" con 4 campi di cui riporto sotto il codice **MySQL** per la creazione (potete incollarlo direttamente nel campo **SQL** di **phpMyAdmin**) :

```
CREATE TABLE `impiegati` (
```

```
     `id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
     `nome` VARCHAR( 200 ) CHARACTER SET latin1 COLLATE latin1_general_ci
NOT NULL ,
     `cognome` VARCHAR( 200 ) CHARACTER SET latin1 COLLATE
latin1_general_ci NOT NULL ,
     `stipendio` BIGINT UNSIGNED NOT NULL DEFAULT '0'
) ENGINE = MYISAM CHARACTER SET latin1 COLLATE latin1_general_ci;
```

Inserendo quattro record nella tabella ottengo questo risultato.

Vediamo ora il codice **PHP** per leggere la tabella dal database **MySQL** e stamparla nella pagina sotto forma di tabella HTML :

```
<?php

     $connessione = mysql_connect("localhost", "admin", "password");
     mysql_select_db("nome_database", $connessione);

     // Leggiamo l'intera lista di impiegati in ordine crescente per il
campo "id"
     $query = "SELECT * FROM impiegati ORDER BY id ASC";
     $risultato = mysql_query($query);

     $righe = mysql_num_rows($risultato);

     echo "La query ha prodotto $righe righe.\n<br /><br />";
     echo "<table border=\"1\" cellspacing=\"5\">";
     echo "<tr><td>ID</td><td>NOME</td><td>COGNOME</td><td>STIPENDIO</td></
tr>";

     while ($riga = mysql_fetch_array($risultato, MYSQL_ASSOC))
     {
          echo "<tr><td>" . $riga["id"] . "</td>";
          echo "<td>" . $riga["nome"] . "</td>";
          echo "<td>" . $riga["cognome"] . "</td>";
          echo "<td>" . $riga["stipendio"] . "</td></tr>";
     }

     echo "</table>";

     mysql_close();

?>
```

Una volta connessi al server **MySQL** e selezionato il database corretto, è sufficiente eseguire una query **SQL** tramite la funzione **mysql_query()**. Quest'ultima restituirà una risorsa che leggeremo attraverso la funzione **mysql_fetch_array().**

mysql_fetch_array() estrae una riga dal risultato e lo trasforma in un array associativo (MYSQL_ASSOC) incrementando poi il contatore per puntare alla riga successiva alla prossima chiamata. E' infatti sufficiente richiamare nuovamente la funzione per ottenere l'array associativo della riga successiva.

Per accedere poi ai valori basta usare il nome del campo come chiave di accesso dell'array.

Il risultato ottenuto.

Vi segnalo al riguardo il mio tutorial per la costruzione di una libreria in **PHP 5** che vi fornirà un'interfaccia più immediata e potente per gestire il vostro **database MySQL**, attraverso la programmazione orientata agli oggetti.

Tags : Guida Php 5 Mysql
Copia link a questo articolo

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 05/09/2007 - 7:44

Sicurezza

Un pò di **sicurezza** non guasta mai, soprattutto quando il traffico generato dal vostro sito diventa consistente. Purtroppo (parlo per esperienza personale) è pieno di individui inetti senza una vita sociale, che passeranno le ore a cercare bachi e buchi nei vostri script e server con il solo scopo di rompervi i coglioni.

Non riuscirete mai a creare un sistema veramente infallibile, ma cercate almeno di rendere la vita un pò più difficile a questi bastardi, anche se credo che in questo caso ne godranno ancora di più.

Cerchiamo innanzitutto di dare meno informazioni possibili sul nostro server.

Provocando ad esempio un **404** (Pagina non trovata) possiamo subito vedere la risposta standard del nostro web server, nel mio caso :
Apache/2.2.4 (Win32) PHP/5.2.3 Server at localhost Port 80

Diamo fin troppe informazioni ai malintenzionati ... informazioni di nessuna utilità per i navigatori del nostro sito, perciò procediamo aprendo il file di configurazione di **Apache** (Apache2.2/conf/httpd.conf) e modificando i seguenti parametri :

- **ServerTokens** - Di default questo parametro è impostato a **Full**, cambiatelo con **Prod**
- **ServerSignature** - Di default questo parametro è impostato a **On**, cambiatelo con **Off**

In questo modo l'ultima riga viene eliminata restituendo un semplice **Not Found**.

Altra cosa molto importante è l'estensione dei file di librearia. Molti programmatori hanno l'abitudine di dare un'estensione differente ai file che contengono classi e funzioni (solitamente .inc) dimenticando però di configurare correttamente il web server per proteggere il codice.

A scanso di equivoci io consiglio di utilizzare sempre e solo l'estensione **.php** anche per i file da includere, in tal modo se viene richiesta una pagina verrà interpretata come codice **PHP** dal server, e non produrrà quindi alcun output sul browser.

Un altro consiglio che posso darvi è quello di criptare sempre i dati sensibili dei vostri utenti come ad esempio le password.

Personalmente preferisco algoritmi di criptazione a "senso unico", ossia che data una stringa ne viene generata un altra univoca criptata, ma data la stringa criptata non è possibile risalire alla stringa originale (vedi "md5()").

In questo modo se un utente dimentica la password, neppure per noi sarà possibile recuperarla, e il metodo migliore sarà quindi resettare la password con una temporanea generata casualmente consentendo poi all'utente di modificarla a sua volta.

Input & Output

Uno script robusto dovrà processare sempre tutti gli input degli utenti, specie se questi dati in input andranno poi memorizzati in supporti comuni come un database MySQL.

Un attacco comune basato su questi errori è chiamato **SQL Injection**, di seguito un articolo valido sull'argomento.

Vi consiglio di controllare sempre in modo restrittivo gli input degli utenti attraverso ad esempio le Espressioni regolari, eliminando caratteri speciali quali l'apostrofo, i trattini (-- commento in SQL) e altri.

Se permettete agli utenti di lasciare ad esempio commenti sul sito che saranno automaticamente pubblicati senza approvazione, è buona abitudine eliminare i caratteri speciali HTML dall'output, per rendere vani i tentativi di inserire codice maligno ad esempio in JavaScript.

Questo è possibile attraverso la funzione **htmlentities()** che sostituisce i caratteri speciali dell'HTML nelle rispettive entità.

Cercate inoltre di evitare di passare dati sensibili e ID di sessione col metodo **GET**. In questi casi affidatevi a **POST**.

File

Il valore di default in **PHP 5** per la dimensione massima di un file che può essere ricevuto dal server è 8MB.

Molto spesso capita di costruire applicazioni che dovranno gestire solo file di circa 1MB o meno, assicuratevi quindi di non sovraccaricare inutilmente il server con file che non saranno poi gestiti ma eliminati immediatamente.

Questo è possibile modificando la proprietà **post_max_size** del php.ini attraverso la funzione **ini_set()** o modificando direttamente il file di configurazione.

Conclusioni

Direi che è tutto, anche se sull'argomento ci si potrebbe scrivere un libro intero.

Vi consiglio quindi di non fermarvi a queste premesse ma di approfondire l'argomento, perchè ci sarà sempre qualcuno più in gamba, e quel qualcuno probabilmente sarà anche più bastardo e disoccupato di voi.

Guida a PHP 5

autore : Casula Francesco
Pubblicato il 06/09/2007 - 10:07

Conclusioni

Eccoci finalmente alla fine di questo piccolo viaggio.

La mia guida non vuole essere una risorsa didattica definitiva per **PHP 5**, sarebbe un imperdonabile peccato di presunzione.

PHP è un linguaggio in continua evoluzione, e la sua potenza ed economicità lo ha portato ad essere ben presto uno dei linguaggi server-side più usati, soprattutto quando abbinato ad un potente linguaggio database open source come **MySQL**.

A questo punto se avete ancora fame di conoscenza e volete diventare dei veri esperti affidatevi al buon **Google**.

Da parte nostra sono tuttora in fase di redazione dei **tutorial** per la realizzazione di librerie base che potranno esservi d'aiuto nella quotidiana scrittura di codice.

Spero di aver contribuito nel mio piccolo, spero che questa guida vi sia stata utile e che siate quindi riusciti a risparmiare qualche euro.

Buon lavoro.

Tags : Guida Php 5
Copia link a questo articolo

Realizzazione-Sito.info - © 2007 - Alcuni diritti riservati - Posta
forged by Francesco Casula
Questo/a opera è pubblicato sotto una Licenza Creative Commons.
Pubblicità su Realizzazione-Sito.info