LatoServer.it

Sviluppo e Programmazione Server-Side

www.latoserver.it

Punto Informatico

L'evoluzione della Rete

www.punto-informatico.it

presentano

Corso di PHP

di

Luca Balzerani

Corso di PHP

Realizzato da Luca Balzerani

L'home page del corso è http://www.latoserver.it/php/corso/

L'autore può essere contattato all'indirizzo di posta elettronica **lou@latoserver.it**

Informazioni sulla versione

Versione 1.0 (2001/05/23 16.56)

Versioni aggiornate di questo documento: http://www.latoserver.it/php/corso/

L'autore

Luca Balzerani è Presidente di LatoServer.it, sito dedicato allo sviluppo di applicazioni web considerato “dal lato del server”.

1. Introduzione a PHP

PHP è un linguaggio di programmazione che consente di realizzare in modo semplice e rapido pagine web dinamiche, cioè pagine il cui contenuto viene generato (dinamicamente) nel momento in cui queste vengono richieste al web server.

Uno script PHP è semplicemente una pagina HTML all'interno della quale viene inserito il codice, cioè le istruzioni che costituiscono il programma. Per questo motivo PHP viene descritto come un linguaggio "HTML-embedded". Il codice PHP viene eseguito sul server prima che la pagina venga inviata al browser (il "client"); il client, quindi, vedrà solo il risultato (cioè l'output) del programma PHP.

Nello scenario della programmazione per il web, PHP si candida ad eccellente alternativa rispetto a linguaggi come Perl e Python ed a tecnologie quali Microsoft ASP.

In questo corso proporremo un approccio a PHP di tipo pratico ed incrementale: tutte le nozioni verranno presentate tramite esempi di difficoltà via via crescente, accompagnati da spiegazioni e commenti.

Il primo esempio che vedremo sarà, inevitabilmente, il classico messaggio di saluto "Ciao mondo" e ci servirà per mostrare la sintassi da utilizzare per includere codice PHP in una pagina web. Ecco il nostro sorgente:

```
<html>
<head><title>Esempio 1</title></head>
<body>

<?php
echo "<h1>Ciao mondo!</h1>";
?>

</body>
</html>
```

Come si vede, si tratta di una normale pagina HTML in cui compaiono speciali marcatori che denotano l'inizio e la fine di un blocco di istruzioni PHP. Nell'esempio mostrato viene utilizzata la sintassi "classica": l'inizio del codice viene contrassegnato con <?php mentre con ?> se ne indica la fine.

Il risultato che otterremo, e cioè la pagina che verrà inviata al browser, sarà il seguente.

```
<html>
<head><title>Esempio 1</title></head>
<body>

<h1>Ciao mondo!</h1>

</body>
</html>
```

Si nota immediatamente che non vi è nessuna traccia del codice originario! In altri termini, il client non ha alcun modo per risalire alle istruzioni PHP che hanno generato la pagina richiesta.

Tornando alla sintassi per l'immersione di codice nell'HTML, ne esistono altre varianti; lo stesso blocco di istruzioni del primo esempio può essere scritto, in modo del tutto equivalente, così (sintassi abbreviata):

```
<?
echo "<h1>Ciao mondo!</h1>";
?>
```

così (sintassi in stile Microsoft ASP):

```
<%
echo "<h1>Ciao mondo!</h1>";
%>
```

o, ancora, cosi':

```
<script language="php">
echo "<h1>Ciao mondo!</h1>";
</script>
```

Quest'ultima forma può essere particolarmente conveniente se si utilizzano degli editor HTML visuali (come ad esempio Front Page) che potrebbero non tollerare gli altri tipi di sintassi.

Per questa prima puntata è tutto; nella prossima passeremo in rassegna le funzioni fondamentali del linguaggio PHP.

Bookmarks

PHP: Hypertext Preprocessor
http://www.php.net

Introduzione a PHP
http://www.latoserver.it/php/intro/

Includere codice PHP in una pagina web
http://www.latoserver.it/php/includere/

Codice che scompare
http://www.latoserver.it/php/scompare/

2. Le funzioni fondamentali

In questa lezione conosceremo da vicino alcune funzioni fondamentali del linguaggio PHP.

Nelle prossime lezioni di questo corso si avrà modo di apprendere i rudimenti del linguaggio PHP; prima di tutto, però, è opportuno prendere familiarità con alcune funzioni di uso estremamente comune.

La prima funzione di cui ci occupiamo è phpinfo(). Quello che fa è generare dinamicamente una (lunga) pagina contenente moltissime informazioni sulla versione di PHP installata e sull'ambiente di esecuzione. Per richiamare phpinfo() e' sufficiente uno script come il seguente:

```
<html>
<head><title>phpinfo()</title></head>
<body>
<?php
phpinfo();
?>
</body>
</html>
```

La pagina web generata da phpinfo() consente di visualizzare la configurazione della nostra installazione di PHP, di conoscere quali estensioni sono disponibili e, cosa particolarmente importante per un neofita, di imparare i nomi delle variabili predefinite che PHP mette a disposizione del programmatore.

La seconda funzione di cui facciamo la conoscenza è certamente la più utilizzata in ogni script PHP: echo(). La funzione echo() serve per scrivere (o stampare) l'output che viene inviato al browser del visitatore che accede al nostro script. Si consideri il seguente esempio:

```
<html>
<head><title>echo</title></head>
<body>

<?php
echo "<h1>Benvenuto!</h1>";
?>

</body>
</html>
```

La pagina che verrà inviata al client, dopo l'elaborazione da parte dell'interprete PHP, sarà la seguente:

```
<html>
```

```
<head><title>echo</title></head>
<body>

<h1>Benvenuto!</h1>

</body>
</html>
```

Grazie ad echo() possiamo visualizzare il contenuto di variabili; nell'esempio seguente viene mostrato, nel messaggio di benvenuto, anche il nome di dominio del sito su cui lo script viene eseguito (nome contenuto nella variabile $HTTP_HOST).

```
<html>
<head><title>echo</title></head>
<body>

<?php
echo "<h1>Benvenuto su $HTTP_HOST!</h1>";
?>

</body>
</html>
```

Se, ad esempio, lo script viene eseguito sul sito www.latoserver.it, la pagina risultante sarà

```
<html>
<head><title>echo</title></head>
<body>

<h1>Benvenuto su www.latoserver.it!</h1>

</body>
</html>
```

A rigore occorre rilevare che echo non è propriamente una funzione bensì un costrutto del linguaggio; per questo motivo non è necessario, come si è visto negli esempi, utilizzare le parentesi tonde per racchiudere gli argomenti.Le ultime due funzioni che esaminiamo sono exit() e die(); entrambe producono il risultato di arrestare l'esecuzione dello script, con la differenza che die() consente anche di stampare un messaggio. Ad esempio il seguente script

```
<html>
<head><title>exit</title></head>
<body>
<? exit(); ?>
<p>Questa frase non si vedrà</p>
</body>
</html>
```

produce questo output:

```
<html>
<head><title>exit</title></head>
<body>
```

Le funzioni exit() e die() possono essere utilizzate, quindi, per gestire eventuali situazioni di errore che non consentono di proseguire l'esecuzione del nostro script PHP (i cosiddetti "errori fatali"). In queste circostanze può essere conveniente usare die() per mostrare un messaggio di errore appropriato.

Vediamo un semplice esempio. Nello script seguente controlliamo il valore della variabile globale $n e mostriamo un messaggio di errore, bloccando l'esecuzione del programma, se questo è maggiore di 1.

```
<html>
<head><title>die</title></head>
```

```
<body>

<?
$n = 5;
if ($n > 1) die("<h1>\$n è maggiore di uno!!!</h1>");
?>

<h1>$n è minore o uguale ad uno!</h1>

</body>
</html>
```

Il risultato sarà il seguente:

```
<html>
<head><title>die</title></head>
<body>

<h1>$n è maggiore di uno!!!</h1>
```

Se, invece, sostituiamo l'istruzione $n=5 con $n=1 il risultato diventa:

```
<html>
<head><title>die</title></head>
<body>

<h1>$n è minore o uguale ad uno!</h1>

</body>
</html>
```

Nella prossima puntata parleremo di variabili e tipi di dato nel linguaggio PHP.

Bookmarks

Dal manuale PHP: phpinfo()
http://www.php.net/manual/en/function.phpinfo.php

Dal manuale PHP: echo
http://www.php.net/manual/en/function.echo.php

Dal manuale PHP: exit()
http://www.php.net/manual/en/function.exit.php

Dal manuale PHP: die()
http://www.php.net/manual/en/function.die.php

3. Variabili e tipi di dato

In questa lezione impareremo l'uso delle variabili in PHP e conosceremo i tipi di dato che il linguaggio ci mette a disposizione.

In uno script PHP i nomi di variabili sono prefissati dal simbolo $ (dollaro); ad esempio, la seguente istruzione:

```
$a = $b + $c
```

assegna ad una variabile di nome `a' la somma dei valori di altre due variabili, rispettivamente `b' e `c'. Il simbolo `=' è l'operatore di assegnamento.

Per creare una variabile è sufficiente assegnarle un valore; in altre parole non vi è alcuna necessità di dichiararla esplicitamente, come avviene, invece, in altri linguaggi. Si parla, dunque, di dichiarazione implicita.

Se vogliamo visualizzare nella nostra pagina PHP il valore di una variabile, in modo che venga mostrata dal browser del visitatore, possiamo usare la funzione 'echo'. Ad esempio

```
// Questa istruzione visualizza il valore della
// variabile $a
echo $a;
```

A disposizione del programmatore c'è anche un certo numero di variabili predefinite, cioè di variabili il cui valore è già impostato. Solitamente queste variabili contengono informazioni circa l'ambiente di esecuzione dello script PHP. Esaminiamone alcune di uso frequente.

La variabile $PHP_SELF contiene il percorso dello script in esecuzione; ad esempio, all'interno dello script PHP raggiungibile all'indirizzo

```
http://www.latoserver.it/index.php3
```

il valore della variabile $PHP_SELF sarà `/index.php3'.

La variabile $HTTP_HOST contiene il nome del server su cui lo script viene eseguito; nel caso dell'esempio precedente il valore di $HTTP_HOST sarebbe `www.latoserver.it'.

Le variabili $HTTP_REMOTE_HOST e $HTTP_REMOTE_ADDR, invece, forniscono rispettivamente il nome di dominio e l'indirizzo IP del visitatore.

L'elenco completo delle variabili predefinite può essere visualizzato con usando la funzione phpinfo(); si veda, in proposito, la lezione precedente.

Vediamo adesso cosa può contenere una variabile. Abbiamo parlato di valori, ma a che tipo di valori facciamo riferimento? Introduciamo dunque i tipi di dato che PHP mette a disposizione del programmatore.

I tipi di dato supportati da PHP si distinguono in tipi scalari e tipi compositi; sono tipi scalari i numeri (sia interi che in virgola mobile) e le stringhe; sono tipi compositi gli array e gli oggetti.

A partire dalla versione 4 di PHP, inoltre, è stato introdotto un nuovo tipo scalare: quello booleano. Una variabile booleana può assumere solo due valori, vero (costante TRUE) o falso (costante FALSE).

```
// $b è una variabile di tipo bool
$b = TRUE;

// $num è una variabile di tipo intero
$num = 25;

// $pi_greco è un numero in virgola mobile;
// si noti il punto (virgola decimale)
$pi_greco = 3.14;

// $messaggio è una stringa
$messaggio = "Ciao a tutti!";
```

Un array in PHP può corrispondere sia ad un vettore, cioè ad una struttura dati in cui ogni elemento è individuato da un indice numerico, sia ad una tabella di hash, cioè (in modo molto semplice) una collezione di coppie nome/valore. In PHP, infatti, tali strutture sono sostanzialmente la stessa cosa.

Un array può essere creato esplicitamente utilizzando il costrutto array() oppure implicitamente. Vediamo alcuni esempi.

```
// Questo è un array di numeri interi
// Lo creo esplicitamente usando array()
$primi = array( 2, 3, 5, 7, 11 );

// Questo array lo creo implicitamente
$pari[0] = 2;
$pari[1] = 4;
$pari[2] = 6;

// Questa è una tabella di hash
$bookmark["puntoinf"] = "punto-informatico.it"
$bookmark["latoserver"] = "www.latoserver.it"
```

Per accedere ad un elemento di un array si può utilizzare sia il corrispondente indice numerico, sia la chiave (se si tratta di una tabella di hash). Ad esempio

```
// Questa istruzione stampa "7"
// cioe' l'elemento di indice 3 dell'array $primi
echo $primi[3];

// Questa istruzione stampa "www.latoserver.it"
echo $bookmark["latoserver"];
```

A differenza di quanto avviene in altri linguaggi di programmazione, un array in PHP può contenere elementi di tipo diverso. Ad esempio, è perfettamente legittimo costruire un array che contenga sia numeri interi che stringhe.

```
// Questo è un array valido!
// Contiene: un numero intero, una stringa,
// un numero in virgola mobile ed un altro array!
$mix = array( 1, "ciao", 3.14, array( 1, 2, 3 ) );
```

Il tipo di dato `object' verrà trattato in una lezione successiva dedicata alla programmazione orientata agli oggetti con PHP.

Nella prossima puntata parleremo delle form (moduli) HTML e di come utilizzarle per inviare informazioni ad uno script PHP.

Bookmarks

Dal manuale ufficiale: Variabili
http://www.php.net/manual/en/language.variables.php

Dal manuale ufficiale: Tipi di dato
http://www.php.net/manual/en/language.types.php

Nomi variabili di variabili (variabili dinamiche)
http://www.latoserver.it/php/nomivar/

4. Form HTML

In questa lezione vedremo come utilizzare le form HTML per inviare informazioni (e passare parametri) ad uno script PHP.

Una form (modulo) HTML è costituita da un certo numero di elementi quali caselle di testo, menù a discesa, caselle di spunta e così via. Tali elementi consentono al visitatore della pagina di inserire delle informazioni esattamente come nel caso della compilazione di un questionario cartaceo.

Un buon manuale HTML può aiutare molto per la comprensione di questa lezione e degli esempi che verranno proposti; è necessario, infatti, un minimo di dimestichezza con i tag utilizzati per la creazione di form.

Come funziona una form? Una volta inseriti i dati richiesti questi vengono inviati, tipicamente con la pressione di un apposito tasto (Submit), ad uno script o ad un'applicazione CGI che li elabora. La URL dell'applicazione viene specificata con l'attributo `action' del tag `form'.

Un esempio di form HTML:

```
<!-- file form.html -->
...
<form action="/scripts/elabora.php" method="get">
<input type="text" name="campione">
<input type="submit" name="bInvia" value="Invia i dati">
</form>
```

L'attributo `method', invece, specifica il metodo da utilizzare per l'invio delle informazioni e può essere GET o POST. Con il metodo GET le informazioni vengono incluse nell'indirizzo URL; sono pertanto visibili nella barra degli indirizzi del browser ma, soprattutto, sono vincolate dalla lunghezza massima di una URL (256 caratteri).

Con il metodo POST, invece, i dati della form vengono scritti sullo "standard input" dell'applicazione destinataria e non sono visibili nella barra degli indirizzi del browser; inoltre, non ci sono limiti sulla quantità di dati inviata.

Dal punto di vista del programmatore PHP i due metodi sono di fatto equivalenti: l'interprete PHP, infatti, analizza sia le informazioni passate tramite la URL (metodo GET) che quelle ricevute sullo standard input (metodo POST) e le rende immediatamente disponibili nello script di destinazione sotto forma di variabili globali.

In alternativa, si può accedere alle informazioni inviate con i metodi GET e POST utilizzando gli array associativi $HTTP_GET_VARS e $HTTP_POST_VARS, rispettivamente. Ad esempio, se tramite il metodo GET inviamo ad uno script PHP un parametro di nome `pagina' e di valore pari a 1, all'interno dello script stesso potremo accedere a tale informazione sia usando la variabile globale $p, sia accedendo a $HTTP_GET_VARS["p"], cioè all'elemento dell'array associativo $HTTP_GET_VARS individuato dalla chiave "p".

Riprendiamo la form dell'esempio precedente e supponiamo di scrivere nella casella di testo la parola "Schumacher" e di premere il pulsante "Invia i dati". Il browser si sposterà all'indirizzo

```
http://www.miosito.tld/scripts/elabora.php?campione=Schumacher
```

Cosa è successo? Poiché la form dell'esempio usa il metodo GET per trasferire i propri dati, questi vengono codificati nell'indirizzo dello script a cui devono essere inviati. Nello script 'elabora.php' adesso troveremo definita una variabile globale di nome $campione il cui valore è la stringa "Schumacher".

```
// Nel file `elabora.php' ...
// Questo stampa "Schumacher"
echo $campione;
```

A questo punto è utile accennare al modo in cui le informazioni provenienti da una form HTML vengono codificate per essere trasmesse con il metodo GET. Partiamo dall'indirizzo URL dello script a cui vogliamo inviare tali dati; ad esso aggiungiamo (a destra) un punto interrogativo che separerà la URL vera e propria (a sinistra) dalla `query string' che andiamo a costruire.

La struttura della query string consiste di una serie di coppie nome/valore separate da una "e" commerciale (&); i caratteri non ammissibili in un indirizzo URL (ad esempio gli spazi) vengono sostituiti dal simbolo di percentuale seguito dal corrispondente codice ASCII (in esadecimale).

Adesso che sappiamo come funziona il metodo GET possiamo sfruttarlo per passare parametri ad uno scrip PHP. Supponiamo di avere uno script di nome `news.php' che estrae notizie ed articoli da un database; ad esso vogliamo passare un parametro, chiamiamolo $argomento, che determinerà il tipo di notizie che ci verranno mostrate. Ad esempio, per ottenere notizie sportive, invocheremo:

```
http://www.miosito.tld/news.php?argomento=Sport
```

In questo caso la codifica manuale della URL era semplice, ma cosa fare se ci interessano le notizie di `Attualità e Cultura'? Niente paura, esiste una funzione PHP, urlencode(), che ci permette di codificare una stringa in una forma ammissibile per una URL. Il frammento di script per creare il link appropriato sarà:

```
<?
echo '<a href="http://www.miosito.tld/news.php?argomento=';
echo urlencode("Attualità e Cultura");
echo '">Clicca qui</a>';
?>
```

E il risultato... scopritelo da voi.

Bookmarks

Dal manuale PHP: HTML Forms (GET and POST)
http://www.php.net/manual/en/language.variables.external.php

Dal manuale PHP: URL functions
http://www.php.net/manual/en/ref.url.php

Dal manuale PHP: la direttiva track-vars
http://www.php.net/manual/en/configuration.php#ini.track-vars

Dalle specifiche di HTML 4: Forms in HTML documents
http://www.w3.org/TR/html4/interact/forms.html

5. Dichiarazione di funzioni

In questa lezione impareremo come costruire le nostre funzioni PHP ed introdurremo le nozioni di variabili locali e globali.

La dichiarazione di proprie funzioni consente al programmatore di estendere il linguaggio utilizzato. Una funzione può essere interpretata in due modi: come funzione in senso `matematico' o come un meccanismo per aggiungere nuovi comandi al linguaggio.

Tramite una funzione, infatti, è possibile raggruppare assieme un blocco di istruzioni PHP che potranno essere richiamate ed eseguite come un tutt'uno. Inoltre una funzione può essere utilizzata in `senso matematico' come relazione tra un input (i parametri passati) ed un output (il risultato della funzione).

Se queste considerazioni possono sembrare, troppo `teoriche', almeno per i neofiti, possiamo senz'altro proporne altre di impatto ben più concreto. In primo luogo, l'uso di funzioni è il primo passo verso una maggiore riutilizzabilità del codice: ogni programmatore può costruire, e magari condividere, una propria libreria di funzioni da riutilizzare al bisogno, senza dover ogni volta `reinventare la ruota'.

Inoltre, è evidente che se in uno script PHP devo ripetere più volte una stessa sequenza di istruzioni è assai più conveniente racchiuderle in una funzione (ed invocare quella) piuttosto che copiare l'intera sequenza in tutti i punti in cui serve. Vediamo, allora, come si dichiara una funzione in PHP.

La dichiarazione di una funzione avviene tramite la parola chiave `function'; la sintassi generale è la seguente:

```
function <nome-funzione> ( <argomenti> ) {
 <corpo-della-funzione>
}
```

dove <nome-funzione> deve essere sostituito dal nome da assegnare alla funzione che si va a dichiarare; <argomenti> è un elenco, eventualmente vuoto, di argomenti (o parametri) da fornire alla funzione; <corpo-della-funzione> è un blocco di istruzioni PHP che implementano la funzione stessa. Vediamo, a titolo di esempio, la dichiarazione di una funzione che somma due numeri.

```
// Una semplice funzione

// Dichiarazione
function somma($a, $b) {
 return $a + $b;
}
```

Per utilizzare la funzione appena dichiarata è sufficiente invocarla usando il nome ad essa attribuito e fornendo gli argomenti previsti, come nell'esempio seguente:

```
// Uso della funzione `somma'

```

```
// Il valore di $risultato è 2
$risultato = somma(1,1);
```

L'istruzione `return' viene utilizzata per restituire il risultato della funzione (cioè, se preferiamo, il suo output) ed uscire da essa, riprendendo l'esecuzione dal punto dello script in cui era stata invocata.

E' interessante osservare che in un linguaggio `orientato alle espressioni' come PHP, il risultato di una funzione rappresenta il valore della funzione intesa come caso particolare di una espressione.

All'interno di una funzione è possibile, naturalmente, utilizzare delle variabili, anche se in questo caso occorre prestare attenzione alla distinzione tra variabili globali e locali. Le variabili globali sono quelle definite all'esterno della funzione (nella parte principale dello script); viceversa, le variabili locali sono quelle definite al suo interno.

Le variabili locali ad una funzione non sono accessibili al di fuori di essa. Ad esempio, supponiamo di aver definito una funzione, `prova', all'interno della quale utilizziamo una variabile di nome $numero; ebbene, la variabile $numero non sarà utilizzabile all'esterno del corpo della funzione,

```
<?
function prova() {
 $numero = 3;
 ...
}

// Qui $numero non è definita!
?>
```

Le variabili globali, invece, sono accessibili anche dall'interno di una funzione ma solo se vengono esplicitamente dichiarate come globali, usando la parola chiave `global'. Ad esempio

```
<?
// $numero e' una variabile globale
$numero = 3;

function prova() {
 // Quando dico $numero
 // intendo la variabile globale
 global $numero;

 echo $numero;
}
?>
```

Nel precedente frammento di codice, la parola chiave `global' è stata utilizzata per indicare che $numero è una variabile globale; in questo modo $numero è accessibile anche dall'interno della funzione. Cosa sarebbe successo se avessimo dimenticato di usare `global'? In questo caso, illustrato nel'esempio seguente, l'interprete PHP avrebbe considerato $numero una variabile locale e l'avrebbe trovata `non definita'.

```
<?
// $numero e' una variabile globale
$numero = 3;

function prova() {
 // Ho dimenticato global!
```

```
 // Questa istruzione non stampa nulla
 // perche' $numero viene considerata locale
 echo $numero;
}
?>
```

In alternativa all'uso di `global' è possibile, dall'interno di una funzione, accedere ad una variabile globale utilizzando l'array associativo $GLOBALS. Ad esempio, dall'interno della funzione `prova' si poteva accedere alla variabile globale $numero, senza usare `global', nel modo seguente: $GLOBALS["numero"].

Alla prossima puntata!

Bookmarks

Dal manuale PHP: Functions
http://www.php.net/manual/en/functions.php

Dal manuale PHP: $GLOBALS
http://www.php.net/manual/en/language.variables.php

6. Costrutti di controllo

In questa puntata impariamo quali sono e come si usano i costrutti che controllano il flusso di esecuzione di uno script.

In questa puntata mostreremo, tramite semplici esempi, quali sono e come si utilizzano, nel linguaggio PHP, i costrutti che controllano il flusso di esecuzione del programma, cioè del nostro script. Possiamo distinguerne due tipologie: i costrutti condizionali e quelli iterativi.

I costrutti condizionali sono quelli che utilizziamo quando vogliamo che l'esecuzione di un certo blocco di istruzioni avvenga o meno in funzione del valore di una espressione (la condizione); questo è il caso del costrutto `if'. Questa situazione comprende anche il caso in cui abbiamo diversi blocchi di istruzioni e vogliamo che, in base al valore dell'espressione data, venga eseguito soltanto uno di questi (costrutto `switch').

I costrutti iterativi, invece, sono quelli che ci permettono di ripetere (iterare, per l'appunto) l'esecuzione di un blocco di istruzioni; il numero di iterazioni potrà sia dipendere dal valore di una condizione (costrutto `while') sia essere prefissato (costrutto `for').

Il costrutto `if' permette di eseguire un certo insieme di istruzioni solo quando risulti verificata una certa condizione. Nell'esempio seguente, la divisione tra due numeri $a e $b avviene solo se il divisore, $b, è diverso da zero.

```
// $a e $b sono due numeri
if ($b != 0) {
  // Il divisore e' diverso da zero
  $c = $a / $b;
}
```

In generale, la sintassi da utilizzare è la seguente:

```
if (condizione)
  blocco-istruzioni
```

dove `blocco-istruzioni' può essere sia una singola istruzione PHP sia un insieme di più istruzioni racchiuso tra parentesi graffe. Se `condizione' è vera allora si esegue `blocco-istruzioni'.

E' possibile anche fare in modo che un secondo blocco di istruzioni venga eseguito nel caso la condizione risulti falsa. In modo informale, il significato è "se è vera la condizione esegui il blocco 1, altrimenti esegui il blocco 2". La sintassi è la seguente:

```
if (condizione)
  blocco-istruzioni1
else
```

```
  blocco-istruzioni2
```

Supponiamo adesso di avere più blocchi di istruzioni e di voler selezionare quale di questi debba essere eseguito in base al valore di una espressione; il modo più conveniente ci viene fornito dal costrutto `switch'. Vediamo un esempio.

```
// La schedina!
switch($segno) {
 case 1:
    echo "E' uscito il segno 1!";
    break;
 case 2:
    echo "E' uscito il segno 2!";
    break;
 default:
    echo "E' uscito il segno X.";
}
```

La sintassi generale dello `switch' è la seguente:

```
switch(espressione) {
  case valore1:
    blocco-istruzioni1
  case valore2:
    blocco-istruzioni2
  ...
}
```

Ogni blocco di istruzioni si chiude con il costrutto `break', che determina l'uscita da `switch'. Come si evince dall'esempio, tramite la parola chiave `deafult' si può prevedere un caso residuale, il cui blocco di istruzioni viene eseguito solo se tutti i confronti precedenti hanno dato esito negativo.

Veniamo adesso ai costrutti iterativi, iniziando dal `while'. La sintassi è la seguente:

```
while (condizione)
  blocco-istruzioni
```

L'esecuzione del `while' inizia con la valutazione della condizione; se quest'ultima risulta verificata allora viene eseguito, una volta, il blocco di istruzioni e si ricomincia da capo con una nuova valutazione della condizione. La prima volta che la condizione risulta non verificata si esce dal `while'. Nell'esempio seguente viene utilizzato un `while' per contare fino a dieci.

```
$contatore = 1;
$max = 10;
while ( $contatore <= $max ) {
  echo "Ho contato fino a $contatore <br>";
  $contatore++;
}
```

L'output prodotto dall'esempio precedente sarà

```
Ho contato fino a 1
Ho contato fino a 2
...
Ho contato fino a 10
```

Il costrutto `while' è particolarmente conveniente quando non è possibile stabilire a priori quante saranno le iterazioni da effettuare.

In molte situazioni pratiche si ha a che fare con iterazioni che lavorano su indici o contatori; in questi casi utilizzeremo il costrutto `for', la cui sintassi è

```
for ( espr1; espr2; espr3 )
  blocco-istruzioni
```

L'esecuzione del costrutto `for' inizia con la valutazione dell'espressione espr1, tipicamente una

inizializzazione, che avviene un'unica volta. Successivamente si valuta espr2 (una condizione) e se questa risulta verificata si eseguono il blocco di istruzioni ed espr3, per poi ritornare alla valutazione di espr2 e così via.

Per fornire un esempio d'uso del `for', riscriviamo con tale costrutto l'esempio precedente del contatore da 1 a 10.

```
$max = 10;
for ($contatore=1; $contatore<=$max; $contatore++) {
  echo "Ho contato fino a $contatore <br>";
}
```

Per ognuno dei costrutti di controllo PHP supporta anche una sintassi alternativa che consente di eliminare le parentesi graffe. In proposito si veda il manuale di riferimento del linguaggio.

Si conclude così la sesta puntata del nostro corso di PHP; con questa lezione si chiude anche la fase propedeutica del corso: introdotte le nozioni fondamentali sul linguaggio, dalla prossima lezione inizieremo ad applicare PHP a casi concreti, mostrando come utilizzarlo per risolvere esigenze reali. Non mancate!

Bookmarks

Dal manuale PHP: Control Structures
http://www.php.net/manual/en/control-structures.php

7. Inclusione di file esterni

In questa puntata vedremo come sfruttare l'inclusione di file o script esterni nella realizzazione di un sito dinamico.

In questa puntata introduciamo due costrutti del linguaggio PHP che ci consentono di includere (ed eseguire) nei nostri script dei file esterni. Tali file esterni potranno essere sia dei semplici frammenti di codice HTML sia degli script PHP: nel primo caso avremo una situazione simile all'uso delle popolarissime SSI (Server-Side Includes); nel secondo, invece, saremo in grado di condividere codice PHP tra più script.

Vediamo subito, allora, quali sono e come si usano i costrutti di cui stiamo parlando, e cioè `require' e `include'. La loro sintassi è mostrata nell'esempio seguente.

```
// Questa istruzione include ed esegue il file
// 'libreria.php3' contenuto nella directory corrente
require "libreria.php3";

// Usando 'include' avrei scritto
include "libreria.php3";
```

Quello che fanno `require' ed `include' è semplicemente leggere il file specificato dall'argomento (si noti che non è obbligatorio utilizzare le parentesi tonde), considerandolo parte integrante dello script PHP. Di conseguenza se il file incluso (`libreria.php3' nell'esempio) contiene istruzioni PHP, queste verranno regolarmente eseguite.

E' importante osservare, comunque, che nel momento in cui PHP inizia ad analizzare il file incluso l'interprete si pone in `modalità HTML', per cui eventuali frammenti di codice PHP, per essere correttamente riconosciuti ed eseguiti, dovranno essere racchiusi dai consueti demarcatori (si veda la prima lezione del corso). Terminata l'analisi del file esterno, si torna in modalità PHP e si continua l'elaborazione dello script principale.

Nella realizzazione di script PHP i costrutti `require' ed `include' vengono spesso utilizzati indifferentemente; d'altra parte, nella maggior parte dei casi essi sono perfettamente equivalenti ed intercambiabili. Esistono, tuttavia, delle differenze di cui occorre essere consapevoli per evitare brutte sorprese. Per l'approfondimento di questo argomento si rimanda ai link in fondo all'articolo.

Vediamo adesso un esempio molto semplice.

Supponiamo di voler realizzare un sito in cui tutte le pagine devono contenere la `classica' barra di navigazione con i link alle varie sezioni del sito stesso. Naturalmente non è ipotizzabile pensare di inserire manualmente un appropriato frammento di HTML (magari con il `copia e incolla') in tutte le pagine: la manutenzione diventerebbe presto ingestibile.

Adottiamo allora un sistema diverso, per il quale ricorreremo a PHP. Per prima cosa isoliamo il frammento di HTML corrispondente alla barra di navigazione e lo salviamo in un file che chiameremo `intestazione.html'. Ecco un esempio di come potrebbe essere un file del genere:

```
<!-- file intestazione.html === INIZIO -->
```

```
<body color="black" bgcolor="white">

<a href="index.php3">Home page</a> |
<a href="pagina2.php3">Pagina 2</a> |

<hr size="1">

<!-- file intestazione.html === FINE -->
```

Si noti che abbiamo inserito nel frammento anche il tag BODY con l'impostazione dei colori della pagina; in questo modo, tutte le pagine utilizzeranno lo stesso insieme di colori per il testo, lo sfondo, etc.

Realizziamo adesso le varie pagine del sito. Ogni pagina sarà uno script PHP piuttosto che una pagina HTML statica, per cui dovremo utilizzare una estensione appropriata per il nome del file, di modo che il web server possa riconoscerla e farla eseguire dal processore PHP. Nel nostro esempio utilizzeremo l'estensione più diffusa e cioè `.php3'.

Per comporre le pagine possiamo utilizzare i nostri strumenti di authoring preferiti; una volta completata una pagina andremo ad inserire il codice PHP necessario per includere il file con la barra di navigazione, come mostrato dall'esempio seguente.

```
<!-- file index.php3 -->
<html>
<head>
<title>Pagina che include un file esterno</title>
</head>

<? require "intestazione.html" ?>

<h1>Pagina che include un file esterno</h1>

(...Contenuto della pagina...)

</body>
</html>
```

Si noti che nella pagina `index.php3' è stato rimosso il tag BODY in quanto presente nella barra di navigazione inclusa. In tutte le pagine PHP costruite in questo modo, l'istruzione

```
<? require "intestazione.html" ?>
```

verrà rimpiazzata dinamicamente dal contenuto del file `intestazione.html'. Qualunque modifica apportata a tale file verrà immediatamente riflessa in tutte le pagine del sito che la includono, rendendone così molto semplice la manutenzione.

Nella prossima puntata vedremo in dettaglio la realizzazione di una barra di navigazione `intelligente' per il nostro sito.

Bookmarks

Inclusione di file esterni
http://www.latoserver.it/php/esterni/

SSI (Server-Side Includes)

http://www.latoserver.it/SSI/

Dal manuale PHP: il costrutto include
http://www.php.net/manual/en/function.include.php

Dal manuale PHP: il costrutto require
http://www.php.net/manual/en/function.require.php

8. Una barra di navigazione “intelligente”

In questa puntata mostreremo come costruire in modo semplice una barra di navigazione ``intelligente'' per il nostro sito in PHP.

Una barra di navigazione è un elemento molto familiare per chi abitualmente naviga sulla Rete; si tratta di quella regione di una pagina web in cui sono raccolti i link alle principali sezioni del sito cui essa appartiene. In questa lezione vedremo come utilizzare PHP per costruirne una ``intelligente'', in grado cioè di evidenziare automaticamente la voce corrispondente alla pagina corrente.

La nostra barra di navigazione sarà generata da uno script PHP separato, che chiameremo `barra.php3', e che verrà richiamato da tutte le pagine in cui vorremo visualizzare la barra.

Cominciamo con il costruire una struttura dati appropriata; nel nostro caso sarà sufficiente un array associativo (vedi lezione 3), che chiameremo $links, in cui avremo tanti elementi quanti sono i link che vogliamo inserire nella barra di navigazione. Ogni elemento consisterà di una chiave e di un valore: la chiave sarà l'indirizzo della pagina web corrispondente, il valore una sua breve descrizione. Ecco un esempio di un siffatto array:

```
// Struttura dati: array con link e descrizioni
$links = array( "index.php3" => "Home page",
           "pagina2.php3" => "Pagina 2",
           "pagina3.php3" => "Pagina 3"
           );
```

In questo modo abbiamo rappresentato, internamente al nostro script, la struttura della barra di navigazione. Veniamo adesso alla parte ``intelligente'': vogliamo, infatti, che quando la barra di navigazione viene visualizzata, venga evidenziata la voce corrispondente alla pagina in cui ci troviamo. Come fa a saperlo? Ci sono sostanzialmente due possibilità: possiamo essere noi a stabilire di volta in volta quale voce evidenziare oppure, come nel nostro caso, fare in modo che ciò avvenga in modo automatico.

Nel nostro esempio, quello che facciamo è confrontare il nome del file della pagina corrente con quelli contenuti nell'array $links: la pagina corrente è quella per cui il confronto ha esito positivo. Il percorso della script PHP corrente è contenuto nella variabile globale $PHP_SELF; poiché nel nostro caso ci interessa soltanto il nome del file piuttosto che l'intero percorso, utilizziamo la funzione `basename' per estrarlo, nel modo seguente:

```
// Nome del file dello script corrente
$pagina_corrente = basename($PHP_SELF);
```

A questo punto non ci resta altro da fare che scrivere il codice PHP necessario alla visualizzazione della barra di navigazione; si tratta, in realtà, di un compito molto semplice: dobbiamo scorrere l'array creato precedentemente, confrontando ogni indirizzo con quello della pagina corrente e, in base all'esito del confronto, rappresentare la voce in maniera opportuna.

Nel nostro caso, ad esempio, decidiamo di racchiudere ogni link della barra di navigazione tra parentesi quadre, con la sola eccezione di quello corrispondente alla pagina corrente, visualizzato in grassetto e senza collegamento ipertestuale. Vediamo il sorgente.

```
// Visualizzazione barra di navigazione

// Riga orizzontale prima dei link
echo "<hr>\n";

// Inizio a scorrere l'array
while (list($url,$desc)=each($links)) {
  if ($url==$pagina_corrente) {
    // Pagina corrente
    echo "<b>$desc</b> ";
  } else {
    // Altre pagine
    echo "[<a href=\"$url\">$desc</a>] ";
  }
}

// Riga orizzontale dopo i link
echo "<hr>\n";
```

Nel frammento di codice proposto si è fatto ricorso alle funzioni `list' e `each' per scorrere gli elementi dell'array associativo; se si utilizza PHP 4 la stessa operazione può essere effettuata utilizzando il costrutto `foreach' (non disponibile nella versione 3).

Per concludere, vediamo come inserire la barra di navigazione nelle pagine del sito (anch'esse in PHP). Ad esempio, la pagina iniziale, sarebbe del tipo:

```
<!-- Questo e' il file index.php3 -->
<html>
<head>
<title>Barra di navigazione intelligente</title>
</head>
<body>

<? require "barra.php3" ?>

<h1>Barra di navigazione intelligente</h1>

<h2>Prima pagina</h2>

<p>Questa &egrave; la home page; in alto dovrebbe
essere visibile la barra di navigazione...

</body>
</html>
```

Come si vede, è sufficiente includere in cima alla pagina, dopo il tag BODY, una istruzione `require' che include ed esegue il codice contenuto in `barra.php3'. (si veda, in proposito, la lezione precedente).

L'esempio proposto, pur nella sua estrema semplicità, probabilmente è già applicabile a molte esigenze concrete; in ogni caso la sua personalizzazione dovrebbe risultare piuttosto agevole.

Anche per questa lezione è tutto, per cui vi saluto ma non prima, naturalmente, di avervi porto gli auguri: buone feste, dunque, e alla prossima puntata!

Bookmarks

Gli esempi da scaricare
http://www.latoserver.it/php/corso/

Dal manuale PHP: la funzione basename
http://www.php.net/manual/en/function.basename.php

Dal manuale PHP: il costrutto list
http://www.php.net/manual/en/function.list.php

Dal manuale PHP: la funzione each
http://www.php.net/manual/en/function.each.php

Dal manuale PHP: il costrutto foreach
http://www.php.net/manual/en/control-structures.foreach.php

9. I cookies

In questa lezione impareremo cosa sono i cookies e come funzionano in PHP.

Ogni utente di Internet ha sentito parlare, una volta o l'altra, dei cookies; l'argomento è tradizionalmente piuttosto controverso, risultando difficile stabilire in modo definitivo se i ``biscotti'' (questo il significato del termine) siano siano da considerare buoni o cattivi. Cerchiamo di capire, allora, di cosa si tratta.

La definizione originale dei cookies li descrive come un meccanismo mediante il quale delle applicazioni lato server posso memorizzare e recuperare informazioni sul lato client (rappresentato dal browser); in questo modo è possibile associare ad ogni visitatore un rudimentale `stato'. Vediamo adesso come si manipolano i cookies con il linguaggio PHP.

Tutte le operazioni di scrittura, modifica o cancellazione di cookies in PHP avvengono mediante una stessa funzione, `setcookie()'. Tale funzione deve obbligatoriamente essere invocata prima che qualsiasi contenuto venga inviato al browser; i cookies, infatti, vengono trasmessi tra client e server sotto forma di intestazioni (headers) HTTP.

La funzione prevede solo due argomenti obbligatori: il nome da assegnare al cookie ed il suo valore. Ad esempio, se vogliamo memorizzare nel browser di un visitatore un cookie, che chiameremo $nomeutente, contenente la stringa "latoserver", l'istruzione da utilizzare sarà la seguente

```
// Imposto un cookie: $nomeutente = "latoserver.it";
setcookie( "nomeutente", "latoserver.it" );
```

E' possibile specificare anche altri argomenti (facoltativi); nell'ordine abbiamo: scadenza del cookie, percorso, dominio e `secure'. Per una discussione dettagliata si rinvia alla sitografia in fondo alla pagina.

Quando un cookie è stato impostato, il suo valore può essere modificato richiamando nuovamente la funzione setcookie() ed associando allo stesso nome il nuovo valore. La cancellazione di un cookie, infine, può avvenire in due modi: assegnandogli un valore nullo o impostando la scadenza ad una data passata.

Resta da vedere in quale modo si possano leggere da uno script PHP le informazioni memorizzate nei cookies; in questo compito siamo molto facilitati, in quanto l'interprete PHP analizza automaticamente i cookies inviati dal browser e li rende disponibili in altrettante variabili globali e nell'array associativo $HTTP_COOKIE_VARS.

Passiamo ad analizzare un esempio pratico di utilizzo dei cookies, realizzando una semplice pagina PHP che ricorda la data e l'ora dell'ultimo accesso del visitatore. A tale scopo impostiamo sul browser un cookie $ultimavisita, la cui scadenza determina quanto a lungo viene conservata tale ``memoria''. Nel nostro esempio non è stata specificata alcuna scadenza, in modo da far scomparire il cookie alla chiusura del browser.

Il valore assegnato di volta in volta al cookie $ultimavisita è il risultato della funzione `time()' e consiste nel numero di secondi trascorsi dalla cosiddetta `Unix epoch' (primo gennaio 1970).

```
<?php
// file `saluto.php'
// Il saluto predefinito
$saluto = "Benvenuto!";

// Controllo se esiste il cookie...
if (isset($HTTP_COOKIE_VARS["ultimavisita"])) {
  // Cambio il saluto con uno piu' appropriato
  $saluto = "Bentornato!";
}

// Imposto il cookie relativo a questa visita
setcookie( "ultimavisita", time() );
?>
<html>
<head>
<title><? echo $saluto ?></title>
</head>
<body>
<h1><? echo $saluto ?></h1>

<?php
if (isset($HTTP_COOKIE_VARS["ultimavisita"])) {
  // Stampo la data dell'ultima visita
  echo "L'ultima volta sei stato qui il " . date( "d/m/Y");
  echo " alle ore " . date( "H:i:s.", $ultimavisita );
  // Link per cancellare il cookie
  echo "<p><a href=\"cancella.php\">Cancella il cookie</a>";
} else {
  echo "Non sei mai stato qui prima?";
}
?>
</body>
</html>
```

Lo script `cancella.php', richiamabile cliccando sul link `Cancella il cookie', non fa altro che cancellare il cookie $ultimavisita, assegnandogli un valore nullo, e dirottare il browser di nuovo alla pagina precedente.

```
<?
// file `cancella.php'
setcookie( "ultimavisita", "" );
header( "Location: saluto.php" );
?>
```

L'esempio proposto, sebbene di improbabile utilità, dovrebbe aver chiarito le modalità d'impiego dei cookies; pur nella sua semplicità, inoltre, può essere il punto di partenza per lo sviluppo di funzionalità personalizzate, lasciate alla vostra fantasia.

Ancora una volta, per oggi è tutto. Alla prossima puntata!

Bookmarks

Gli esempi del corso
http://www.latoserver.it/php/corso/

Cookies: facciamo i biscotti con PHP
http://www.latoserver.it/php/cookies/

Dal manuale PHP: Cookies
http://www.php.net/manual/en/features.cookies.php

Dal manuale PHP: la funzione setcookie()
http://www.php.net/manual/en/function.setcookie.php

Le specifiche originali (Netscape)
http://www.netscape.com/newsref/std/cookie_spec.html

10. Le date in PHP

Lavorare con date ed orari è un'operazione non banale, considerata la particolare struttura dati con cui si ha a che fare. In molte applicazioni, tuttavia, si ha necessità di manipolare date, confrontarle, validarle e così via. In questa lezione vedremo in che modo possiamo operare su date ed orari nei nostri script PHP, ricorrendo ad esempi concreti.

Nel linguaggio PHP le date vengono rappresentate sotto forma di `timestamp'; un timestamp è semplicemente un numero intero che corrisponde al numero di secondi trascorsi dalla cosiddetta `Unix epoch' (le ore 0:00:00 del primo gennaio 1970). Ad esempio, le ore 0:00:00 del primo gennaio 2001 corrispondono al timestamp 978303600.

Per conoscere il timestamp corrispondente ad una certa data basta invocare la funzione mktime() passandole i parametri previsti e cioè, nell'ordine, ore, minuti, secondi, mese, giorno, anno. Un ulteriore argomento, opzionale, consente di tener conto dell'ora legale. Vediamo quale istruzione abbiamo utilizzato per determinare il timestamp mostrato in precedenza.

```
// Timestamp delle ore 0:00:00 del primo gennaio 2001
echo mktime(0, 0, 0, 1, 1, 2001);
```

I timestamp, naturalmente, sono poco comprensibili per un essere umano, per cui vediamo subito in che modo si possano tradurre in una forma ``human readable''. PHP ci mette a disposizione, a questo scopo, la funzione date(). Gli argomenti di date() sono due: il primo, obbligatorio, è una stringa che rappresenta il formato in cui codificare la data; il secondo, facoltativo, è il timestamp da formattare. Se quest'ultimo viene omesso, verrà considerato il timestamp corrente. Vediamo un esempio.

```
// Questa istruzione stampa la data corrente
// nel formato gg/mm/aaaa
echo "Data di oggi " . date("d/m/Y");
```

Nell'esempio mostrato, la stringa che esprime il formato in cui visualizzare la data è "d/m/Y". Ognuna delle lettere che vi compaiono corrisponde ad un elemento della data; nel nostro caso al giorno del mese (01-31), al mese (01-12) ed all'anno, espresso con quattro cifre. Si consulti la pagina del manuale PHP relativa alla funzione date() per un elenco completo delle lettere utilizzabili. Nel frattempo, divertiamoci con altri esempi.

```
// Come sopra ma senza lo zero prima di giorni e/o mesi
```

```
// di una sola cifra
echo "Data di oggi " . date("j/n/Y");
```

L'esempio seguente stampa il numero di giorni trascorsi dall'inizio dell'anno.

```
echo "Sono trascorsi ".date("z")." giorni dall'inizio dell'anno.";
```

Riprendiamo l'esempio della data corrente; come possiamo visualizzarla in modo testuale? Ecco una possibile soluzione.

```
// Nomi dei giorni della settimana
$giorni = array( "Dom", "Lun", "Mar", "Mer", "Gio", "Ven", "Sab" );

echo "Oggi e': " . $giorni[date("w")];
```

In modo del tutto analogo si può ottenere il nome dei mesi. Per evitare di chiamare ripetutamente la funzione date() se ne può utilizzare un'altra, getdate(), che restituisce le informazioni sulla data ed ora correnti in un array associativo.

Fino ad ora abbiamo considerato operazioni estremamente semplici sulle date; nel seguito mostriamo, invece, come eseguirne altre più evolute. In primo luogo, vediamo come sia possibile validare una data, cioè accertarsi che si tratti di una data valida. Per citare qualche esempio sono date non valide il 31 aprile o il 29 febbraio di un anno non bisestile. La funzione preposta alla verifica di una data è checkdate().

```
// Verifichiamo una data (31 aprile 2001!?)
$giorno = 31;
$mese = 4;
$anno = 2001;

echo "La data $giorno/$mese/$anno ";
if (checkdate($mese,$giorno,$anno)) echo "è corretta.";
else echo "non è valida!";
```

Supponiamo adesso di voler confrontare due date per verificare se la prima è antecedente alla seconda. In PHP un modo rapido per risolvere il problema è quello di convertire entrambe le date nei corrispondenti timestamp e confrontare quelli.

```
// Data n° 1: ancora il primo gennaio 2001
$data1 = mktime(0, 0, 0, 1, 1, 2001, 0);

// Data n° 2: il 29 luglio 2001
$data2 = mktime(0, 0, 0, 7, 29, 2001, 0);

echo "La prima data è ";

if ($data1 < $data2) echo "precedente";
else echo "successiva";

echo " alla seconda.";
```

Altrettanto semplicemente si può calcolare il numero di giorni che separa due date arbitrarie, come mostrato nell'esempio seguente:

```
// Data n° 1: ancora il primo gennaio 2001
$data1 = mktime(0, 0, 0, 1, 1, 2001, 0);

// Data n° 2: il 29 luglio 2001
$data2 = mktime(0, 0, 0, 7, 29, 2001, 0);

echo "Tra le due date ci sono ";
echo ($data2 - $data1)/(60*60*24);
echo " giorni.";
```

La differenza tra i due timestamp, espressa in secondi, può essere converita in giorni dividendo

per 24 ore al giorno, 60 minuti l'ora, 60 secondi al minuto. L'unica particolarità riguarda l'ultimo parametro fornito a mktime(), impostato a zero in modo da ignorare l'ora legale.

A questo punto non mi resta che augurarvi buon divertimento con le date e darvi appuntamento alla prossima lezione.

Bookmarks

Gli esempi del corso
http://www.LatoServer.it/php/corso/

Dal manuale PHP: Date and Time functions
http://www.php.net/manual/en/ref.datetime.php

11. PHP e il database MySQL

In questa lezione vedremo come operare su un database MySQL tramite il linguaggio PHP.

La coppia PHP e MySQL è sicuramente una delle più diffuse nella realizzazione di applicazioni web basate su software OpenSource. MySQL è un DBMS, Data Base Management System, cioè un software per la gestione di basi di dati; la sua popolarità è indiscussa, grazie alle prestazioni di tutto rispetto e nonostante la mancanza di diverse caratteristiche avanzate (transazioni, stored procedures, etc.).

Nel seguito vedremo in che modo è possibile, da uno script PHP, collegarsi ad un database MySQL ed eseguire operazioni su di esso tramite il linguaggio SQL. Per una completa comprensione della lezione è necessaria una conoscenza di base di SQL; assumeremo, inoltre, che MySQL sia correttamente installato.

L'accesso ad un database MySQL avviene mediante autenticazione; questo vuol dire che, prima di poter effettuare qualsiasi operazione su di esso, dobbiamo disporre dei privilegi necessari. In particolare, le informazioni di cui abbiamo bisogno sono: il nome dell'host su cui è in esecuzione il server MySQL, il nome del nostro database, il nome utente che ci è stato assegnato e la relativa password (per averle occorre rivolgersi all'amministratore di sistema). Supponiamo che i parametri nel nostro caso siano i seguenti:

```
// Il nome dell'host (hostname) su cui si trova MySQL
$dbhost = "localhost";

// Il nome del nostro database
$dbname = "dbprova";

// Il nostro nome utente (username)
$dbuser = "luca";

// La nostra password
$dbpass = "secret";
```

E' opportuno salvare tali parametri in apposite variabili piuttosto che utilizzarli direttamente nelle chiamate di funzione; in tal modo, infatti, potremo inserirli in un file separato che includeremo in tutti gli script che accedono al database.

La prima funzione che utilizziamo è mysql_connect(), che ci servirà per instaurare la connessione con il server MySQL. I parametri da fornire a tale funzione sono il nome dell'host, il nome utente e la password. Vediamo un esempio:

```
// Funzione mysql_connect()
$conn = mysql_connect($dbhost,$dbuser,$dbpass)
 or die("Impossibile collegarsi al server MySQL.");
```

Si osservi la sintassi utilizzata nell'esempio precedente; il significato è il seguente: se la funzione restituisce un valore nullo, situazione che denota l'impossibilità a collegarsi al server MySQL, viene invocata die() per arrestare l'esecuzione e visualizzare un messaggio di errore. Inoltre, in una variabile che abbiamo chiamato $conn salviamo il valore restituito da mysql_connect(), che servirà

da identificativo della connessione stabilita.

Il passo successivo è la selezione del database su cui operare; la funzione da utilizzare è mysql_select_db(), alla quale occorre fornire il nome del database e, opzionalmente, l'identificativo della connessione (se non viene indicata verrà utilizzata l'ultima aperta).

```
// Funzione mysql_select_db()
mysql_select_db($dbname,$conn)
 or die("Impossibile selezionare il database $dbname");
```

Anche questa volta in caso di insuccesso viene arrestata l'esecuzione del programma PHP e viene visualizzato un messaggio di errore appropriato.

Arriviamo così alla parte più importante, quella dell'interazione con la base di dati, interazione che avverrà mediante il linguaggio SQL. Le funzioni PHP che utilizzeremo in questa fase sono mysql_query(), per l'esecuzione di comandi SQL, e mysql_fetch_row(), per prelevare il risultato restituito da MySQL quando il comando eseguito è un'istruzione SELECT (quindi una interrogazione).

Supponiamo di voler creare una tabella del database con cui gestire una rudimentale rubrica telefonica. Il comando SQL da utilizzare sarà del tipo

```
CREATE TABLE rubrica(
 Progressivo int PRIMARY KEY AUTO INCREMENT,
 Nome varchar(40),
 Cognome varchar(40),
 Telefono varchar(20))
```

Per eseguire tale comando dal nostro script PHP, lo inseriamo dapprima in una stringa, ad esempio nel modo seguente

```
$sql = "CREATE TABLE rubrica( "
 . "Progressivo int PRIMARY KEY AUTO INCREMENT, "
 . " Nome varchar(40), Cognome varchar(40), Telefono varchar(20))";
```

Passiamo poi all'esecuzione vera e propria, invocando la funzione mysql_query().

```
// Esegue il comando SQL o stampa un messaggio di errore
$res = mysql_query($sql,$conn)
 or die( "Errore: " . mysql_error() );
```

Voilà, il gioco è fatto! In caso di errori, comunque, l'istruzione mysql_error() ci fornisce la descrizione del problema che si è verificato.

Esaminiamo adesso il caso di una interrogazione del database. L'esecuzione del relativo comando SQL (sarà, naturalmente, una "SELECT") è del tutto analoga al caso già visto. L'unica differenza risiede nel fatto che, in questo caso, abbiamo un risultato da prelevare. Una delle funzioni che possiamo utilizzare a tale scopo è mysql_fetch_row().

```
// Interroghiamo la nostra rubrica

// Comando SQL da eseguire
$sql = "SELECT Telefono FROM rubrica "
 . "WHERE Nome='Luca' AND Cognome='Balzerani'";

// Esecuzione comando SQL o messaggio di errore
$res = mysql_query($sql,$conn)
 or die( "Errore: " . mysql_error() );
```

```
// Estrazione del risultato
$info = mysql_fetch_row($res);
echo "Il mio numero di telefono è " . $info[0];
```

Al termine della sessione di lavoro si può invocare la funzione mysql_close() per chiudere la connessione con il server MySQL. Si tratta, in ogni caso, di un passo opzionale in quanto tutte le connessioni ``dimenticate'' aperte verranno chiuse automaticamente alla fine dello script.

```
// Funzione mysql_close()
mysql_close($conn);
```

Si conclude qui questa lezione dedicata a PHP e MySQL; nella prossima lezione, che sarà anche l'ultima del corso, ci occuperemo delle sessioni. Non mancate!

Bookmarks

MySQL: il sito ufficiale
http://www.mysql.com

Installare MySQL su Windows 95/98
http://www.latoserver.it/mysql/install-win32/

Dal manuale PHP: MySQL functions
http://www.php.net/manual/en/ref.mysql.php

Gli esempi del corso
http://www.latoserver.it/php/corso/

12. Le sessioni

Cosa sono le sessioni? Come si usano in PHP? A questi ed altri interrogativi risponderemo in questa ultima lezione del nostro corso.

Nella lezione dedicata all'uso dei cookies abbiamo visto che essi, consentendoci di rendere persistenti delle variabili nell'arco di più accessi HTTP, possono essere sfruttati per il mantenimento di un rudimentale stato. Il meccanismo dei cookies, tuttavia, presenta delle limitazioni intrinseche che lo rendono inadatto ad applicazioni complesse. Un esempio per tutti: vogliamo essere in grado di mantenere uno stato anche nell'eventualità in cui il browser del visitatore non supporti i cookies (o li abbia disabilitati).

Arriviamo così a parlare di sessioni. Precisiamo subito che il supporto nativo per le sessioni è stato introdotto nella versione 4 di PHP; se si ha necessità di gestire le sessioni con la versione precedente sarà necessario ricorrere ad apposite librerie, come ad esempio le ottime PHPLIB. Cominciamo con l'esaminare alcune nozioni di base sulle sessioni.

Una sessione, in questo contesto, consiste di una serie di accessi alle nostre pagine PHP, effettuati in un determinato arco di tempo durante il quale viene mantenuto uno stato. Ogni sessione è individuata da un identificatore, univoco, utilizzato per l'associazione tra client e relativa sessione.

Per utilizzare le sessioni in PHP 4 abbiamo bisogno essenzialmente di tre funzioni, e precisamente: session_start(), session_register() e session_destroy().

La prima funzione, session_start(), viene invocata per creare una nuova sessione o per ripristinarla, nel caso sia stata creata in precedenza. Questa funzione tenta anche di impostare, nel browser, un cookie contenente l'identificativo di sessione, per cui è necessario che venga invocata all'inizio degli script, esattamente come nel caso della funzione setcookie().

La funzione session_register(), invece, viene utilizzata per registrare delle variabili come variabili di sessione. Ad esempio, se abbiamo una variabile $nomeutente e vogliamo renderla persistente per tutta la durata della sessione, invocheremo session_register() nel modo seguente

```
// $nomeutente diventa variabile di sessione
session_register("nomeutente");
```

Infine, la funzione session_destroy() viene invocata per distruggere i dati relativi alla sessione, tipicamente al momento del ``log out".

Come si vede, quindi, lavorare con le sessioni è piuttosto semplice. L'unico aspetto critico, al quale il programmatore deve prestare la massima attenzione, è quello della propagazione del SID, il già citato identificatore di sessione. Sebbene, infatti, nella maggior parte delle situazioni reali, sarà

sufficiente memorizzare tale valore in un cookie nel browser del visitatore (PHP lo fa automaticamente), è opportuno gestire anche il caso in cui i cookies non possano essere utilizzati (perché il browser non li supporta o è stato configurato per rifiutarli).

In questo ``caso pessimo'', è il programmatore PHP che deve preoccuparsi della propagazione del SID, modificando i link tra i vari script che condividono la sessione e passandolo come parametro. A dispetto di quanto possa sembrare, si tratta di una operazione piuttosto semplice: basta includere nei link alle altre pagine PHP il valore della costante SID, come nell'esempio seguente:

```
<!--
 Un esempio di link che propaga l'identificativo
 di sessione senza richiedere cookies
 -->
<a href="altroscript.php?<?= SID ?>">Altro script</a>
```

Un esempio completo

Mettiamo in pratica quanto abbiamo imparato sulle sessioni realizzando un semplice esempio: la ``sequenza di colori''. Si tratta di uno script PHP che propone al visitatore di cliccare su uno dei tre colori disponibili (bianco, rosso e verde), mostrando al contempo la sequenza delle selezioni effettuate fino a quel momento. Il funzionamento dello script è molto semplice: la sequenza dei click sui vari colori viene memorizzata in un array, registrato come variabile di sessione.

Vediamo in dettaglio il codice. La prima cosa da fare è l'avvio della sessione; è importante ricordare che affinché PHP possa impostare il cookie con l'identificativo di sessione, questa chiamata di funzione deve avvenire prima che qualsiasi output sia inviato al browser del visitatore.

```
// Attivo (o ripristino) la sessione
session_start();
```

Il secondo passo è la registrazione dell'array $clicks come variabile di sessione.

```
// 'clicks' e' una variabile di sessione: devo registrarla
session_register("clicks");
```

Passiamo adesso all'analisi dei parametri passati dall'utente. Per prima cosa verifichiamo se è stato cliccato il link ``Ricomincia da capo'', che comporta l'azzeramento della sequenza (cioè dell'array).

```
// Devo azzerare?
if ($azzera) {
  $clicks = array();
}
```

Se l'utente ha cliccato su uno dei colori lo aggiungiamo alla sequenza, inserendo un nuovo elemento in coda all'array.

```
if ($click) {
  $clicks[] = $click;
}
```

Infine, arriviamo alla visualizzazione della sequenza dei colori. Se l'array che rappresenta la sequenza contiene almeno un elemento (quindi, se è stato scelto almeno un colore) vengono visualizzati tutti i suoi elementi; diversamente viene mostrata la scritta ``(sequenza vuota)''.

```
if (count($clicks)) {
  foreach ($clicks as $colore) { echo "$colore "; }
} else {
  echo "(sequenza vuota)";
}
```

Con l'esempio appena presentato si conclude questa lezione che è anche l'ultima del Corso di PHP. Prima dei saluti, però, voglio ringraziare tutti i lettori che lo hanno seguito, ricordando che per

inviare commenti, critiche o suggerimenti sul corso è a vostra disposizione il forum su PHP di LatoServer.it: http://www.latoserver.it/forum/

Grazie e... buon divertimento con PHP! :-)

Luca Balzerani

Bookmarks

Commenti, critiche, suggerimenti sul corso?
http://www.latoserver.it/forum/

Gli esempi del corso
http://www.latoserver.it/php/corso/

Dal manuale PHP: session handling functions
http://www.php.net/manual/en/ref.session.php