Corso di Web Programming

11. PHP - Complementi

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa
http://www.di.unipi.it/∼milazzo
milazzo@di.unipi.it

Corso di Laurea in Informatica Applicata
A.A. 2010/2011

Sommario

Programmazione orientata agli oggetti in PHP (1)

- La possibilità di lavorare con classi e oggetti in PHP è stata introdotta in tempi abbastanza recenti
  - supporto di base agli oggetti da PHP 4.0
  - supporto completo agli oggetti e migliori performance da PHP 5.0
- In PHP le variabili e i metodi degli oggetti possono essere specificati a priori attraverso la definizione di classi
- E' anche possibile (ma sconsigliato) aggiungere nuove variabili d'istanza a oggetti già creati (in stile JavaScript)

Programmazione orientata agli oggetti in PHP (2)

- Una classe si definisce attraverso la keyword `class` e specificando le variabili d'istanza e i metodi (in questo caso funzioni)

```
class User {
  public $username;
  public $nome;
  public $cognome;

  function __construct($n, $c, $un) {
    $this->username = $un;
    $this->nome = $n;
    $this->cognome = $c;
  }

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

  function toString() {
    return $this->getName() . " (" . $this.username . ")";
  }
}
```

Programmazione orientata agli oggetti in PHP (3)

- Si può specificare un costruttore come funzione che si chiama come la classe oppure (da PHP 5.0) tramite una funzione `__construct`
- Si può specificare un metodo distruttore (richiamato ogni volta che un oggetto non ha più riferimenti) chiamato `__destruct`
- Si può usare `$this` per accedere alle variabili e ai metodi della classe stessa (stessa istanza)
- Un oggetto di una classe viene creato usando la parola chiave `new`
- Si usa la notazione `->` per accedere alle variabili e ai metodi di un oggetto (Esempi: `$o->nome`, `$o->toString()`)
  - Attenzione: quando si accede a una variabile di un oggetto non si deve usare `$` a destra di `->`

Programmazione orientata agli oggetti in PHP (4)

Esempio di creazione e uso di un oggetto

```
/* creazione di un oggetto */
$paolo = new User("Paolo","Milazzo","pmilazzo");

/* accesso a una variabile d'istanza */
if ($paolo->username=="pmilazzo") {

  /* invocazione di un metodo */
  print $paolo->toString();

}
```

Programmazione orientata agli oggetti in PHP (5)

- Si possono usare (da PHP 5.0 in poi) i modificatori di variabili e metodi `public`, `protected` e `private` come in Java (se omesso si intende `public`)
- Si possono definire variabili e metodi di classe (statici) tramite il modificatore `static`, usando `::` al posto di `->` e `self` al posto di `$this`

```
class ContaIstanze {
  static $contatore = 0;
  function __construct() {
    self::contatore++;
  }
}

$o1 = new ContaIstanze();
$o2 = new ContaIstanze();
echo ContaIstanze::$contatore; //scrive 2
```

Programmazione orientata agli oggetti in PHP (6)

- Si possono definire gerarchie di classi tramite l'operatore `extends` (single inheritance) con le possibilità di fare overloading dei metodi, ecc...

```
class Subscriber extends User {
  public $email;

  function __construct($n, $c, $un, $em) {
    parent::__construct($n, $c, $un);
    $email = $em;
  }

  function display() {
    echo "Nome: " . $this.nome . "<br>";
    echo "Cognome: " . $this.cognome . "<br>";
    echo "Email: " . $this.email . "<br>";
  }
```

Programmazione orientata agli oggetti in PHP (7)

Può far comodo...

- ... per stampare il contenuto di un oggetto (nomi delle variabili e valori assegnati) in formato "leggibile" usare la funzione `print_r()`. Il seguente codice

```
$o = new User("Mario","Rossi","mr1");
print_r($o);
```

stampa

```
User Object ( [username] => mr1 [nome] => Mario [cognome] => Rossi )
```

- ... per clonare un oggetto (creare un nuovo oggetto uguale ad un altro) usare l'operazione `clone` (attenzione, non è una funzione)

```
$o1 = new User();
$o1->nome = "Mario";
$o2 = clone $o1;
```

Sommario

L'estensione MySQLi (1)

- La libreria di funzioni per accedere a MySQL che viste nelle lezioni precedenti fanno parte dell'estensione PHP detta `MySQL`
- Recentemente è stata introdotta una nuova estensione detta `MySQLi` (MySQL improved) con un'interfaccia object oriented e alcune nuove funzionalità
- Sebbene la vecchia estensione `MySQL` sia ritenuta al momento più stabile di `MySQLi`, quest'ultima diventerà nel prossimo futuro l'estensione "ufficiale"
- L'uso di `MySQLi` non è in realtà molto diverso dall'uso di `MySQL`....

L'estensione MySQLi (2)

Connessione a un database:

```
/* parametri per la connessione */
$host = "localhost";     /* server MySQL */
$user = "paolo";         /* utente */
$pwd = "xxxxxxx";        /* password */
$dbname = "archvio";     /* nome database */

/* connessione al database */
$conn = new mysqli($host,$user,$pwd,$dbname);

/* gestione errori di connessione */
if (mysqli_connect_errno()) {
  print "<p>Errore di connessione al database</p>";
  exit();
}
```

Da notare:

- Connessione e scelta database in un unico passo
- Funzione separata per la gestione degli errori

L'estensione MySQLi (3)

Esecuzione della query:

- Primo metodo (sconsigliato se ci sono variabili)

```
/* preparazione della query usando variabili */
$conn->query("SELECT nome,cognome FROM people
              WHERE citta=$city AND eta=$age");
```

- Secondo metodo, utilizzo di un "prepared statement"

```
/* preparazione della query con parametri "?" */
$query = $conn->prepare("SELECT nome,cognome FROM people
                         WHERE citta=? AND eta=?");
/* binding dei parametri */
$query->bind_param('si',$city,$age);
/* esecuzione della query */
$query->execute();
```

- L'uso del prepared statemente e di `bind_param` è consigliato in quanto questa funzione svolge controlli di sicurezza sulle variabili (vedremo)
- La stringa passata come primo argomento a `bind_param` contiene caratteri che rappresentano i tipi attesi dei parametri nell'ordine
  - `s` per string, `i` per integer, `d` per double e `b` per blob

L'estensione MySQLi (4)

Gestione del risultato della query:

```
/* specifica delle variabili da legare ai vari campi del risultato */
$query->bind_result($name,$lastname);

print "<table>";
print "<tr><th>Nome</th><th>Cognome</th></tr>";

/* ciclo che scandisce il risultato della query riga per riga */
while ($query->fetch()) {
  print "<tr><td>$name</td><td>$lastname</td></tr>";
}

print "</table>";
```

Da notare:

- L'invocazione di `bind_result` serve per dire a `fetch` dove andare a mettere i risultati della query
- Bisogna fare attenzione a passare a `bind_result` tante variabili quante sono le colonne del risultato della query

L'estensione MySQLi (5)

Chiusura della connessione a MySQL:

```
$conn->close();
```

L'estensione SQLite (1)

- MySQL è sempre stato il DBMS più comunemente usato con PHP
- Un'istallazione (o configurazione) minimale di PHP potrebbe però non comprendere MySQL
- Inoltre per piccole basi di dati si potrebbe preferire qualcosa di più semplice di MySQL
- Nelle ultime versiondi di PHP (da PHP 5.0) è stata inclusa la nuova estension SQLite
- SQLite consiste di una libreria (sia procedurale che object-oriented) per memorizzare dati in piccoli database memorizzati come singoli file binari
- Trattandosi di una libreria e non di un DBMS, SQLite esiste solo all'interno dei linguaggi di programmazione che la implementano (tra cui PHP). Non si può accedere ai dati tramite console, o simili...

L'estensione SQLite (2)

Esempio di uso di SQLite:

```
// apre (o crea) un database
$db=new SQLiteDatabase("miodatabase.sqlite",0666,$err);
if (!$db) die("Errore SQLite: $err");

// esegue una query che crea una tabella e la popola
$db->query(
     "CREATE TABLE nomi (id INTEGER PRIMARY KEY, nome VARCHAR(255));
      INSERT INTO nomi (nome) VALUES ('uno');
      INSERT INTO nomi (nome) VALUES ('due');
      INSERT INTO nomi (nome) VALUES ('tre');"
     );

// interroga il database
$risultato=$db->query("SELECT * FROM nomi");

// scandisce il risultato dell'interrogazione
while($risultato->valid()) { // finche' non siamo alla fine dei dati
    $dati=$risultato->current(); // restituisce il risultato
    print_r($dati);
    print("<br>");
    $risultato->next(); // va alla prossima riga di dati
}
```

L'estensione SQLite (3)

Altro esempio in cui si usa il metodo bufferizzato `arrayQuery`:

```
// apre (o crea) un database
$db=new SQLiteDatabase("miodatabase.sqlite",0666,$err);
if (!$db) die("Errore SQLite: $err");

// interroga il database caricando tutti i dati in memoria
$risultato=$db->arrayQuery("SELECT * FROM nomi", SQLITE_ASSOC);

// scandisce il risultato dell'interrogazione
foreach ($risultato as $row) {
  print $row['nome'] . "<br>";
}
```

- la costante `SQLITE_ASSOC` permette alla funzione buffered `sqlite_array_query` di ottenere un array associativo basato sui nomi dei campi della tabella
- Ci sono anche altre costanti, tra cui:
  - `SQLITE_NUM`: restituisce un array con indice numerico a partire da zero
  - `SQLITE_BOTH`: restituisce un array con indice sia numerico sia basato sui nomi dei campi della tabella

L'estensione SQLite (4)

SQLite consente di usare funzioni PHP all'interno del codice SQL

```
// definisce una semplice funzione
function conta($str) { return strlen($str); }

// apre (o crea) un database
$db=new SQLiteDatabase("miodatabase.sqlite",0666,$err);
if (!$db) die("Errore SQLite: $err");

// lega la funzione PHP "conta" al nome di funzione SQL "contaSQL"
$db->createFunction("contaSQL", "conta", 1);

// esegue una query usando la nuova funzione e scandisce il risultato
$ris=$db->arrayQuery("SELECT nome, contaSQL(nome) AS len FROM nomi");
foreach ($ris as $row) {
  print $row['nome'] . "(" . $row['len'] . ") <br>";
}
```

`createFunction()` comunica a SQLite l'esistenza della nuova funzione:

- primo parametro: nome della funzione da usare nella sintassi SQL
- secondo parametro: nome originale della funzione PHP
- terzo parametro: numero di parametri che utilizza la funzione

Sommario

Alcuni aspetti di sicurezza in PHP

- L'utilizzo di PHP e MySQL espone ad alcuni possili attacchi alla sicurezza del sistema
- Anche nei piccoli siti web è bene prestare attenzione agli aspetti di sicurezza!
- I tipi di attacchi più semplici da realizzare consistono nell'effettuare richieste all'applicazione PHP nascondendo frammenti di codice nei parametri:
  - MySQL Injection Attacks: frammenti di codice SQL da far eseguire all'applicazione PHP
  - Cross-site Scripting (XSS) Attacks: frammenti di codice HTML o JavaScript da memorizzare nella base di dati e successivamente eseguiti da altri client

MySQL Injection Attacks (1)

Esempio:

- supponiamo che l'applicazione PHP contenga la seguente query a un database

```
$result = mysql_query("SELECT * FROM people
                       WHERE nome=\"$name\"", $conn);
```

  dove il contenuto di `$nome` è specificato dagli utenti tramite un form HTML
- supponiamo inoltre che un utente malevolo inserisca nel form il seguente dato (virgolette incluse)

```
"; DROP people; SELECT * FROM foo WHERE nome="
```

- andando a sostituire `$nome` nella query otteniamo

```
SELECT * FROM people WHERE nome=""; DROP people; SELECT * FROM foo
                                    WHERE nome=""
```

  che cancella la tabella `people` dalla base di dati!!!

MySQL Injection Attacks (2)

Soluzione:

- dare in pasto i dati ottenuti dagli utenti ad una funzione di "escaping" tipo `mysql_real_escape_string()` (oppure `stripslashes()`) che prefissa le virgolette e gli altri caratteri sensibili in SQL con \
- nell'esempio:

```
$name = mysql_real_escape_string($name);
$result = mysql_query("SELECT * FROM people
                       WHERE nome=\"$name\"", $conn);
```

- usando il metodo `bind_param` in MySQLi l'escaping dei dati viene effettuato in automatico!
- E' buona norma inoltre controllare sempre che il tipo, il formato e la lunghezza dei dati ricevuti dai form sia quello atteso (usando ad esempio le espressioni regolari)

Cross-site Scripting (XSS) Attacks (1)

- In molti siti web i dati immessi da un utente sono memorizzati in una base di dati, e successivamente visualizzati da un altro utente (es. Forum, blog, wiki, ecc....)
- Che succede se un utente malevolo inserisce codice HTML o JavaScript nella base di dati (ad esempio postando un messaggio in un forum)?

Cross-site Scripting (XSS) Attacks (2)

Esempio:

- supponiamo che l'applicazione PHP visualizzi i dati di un database come una tabella usando il codice seguente:

```
print "<table>";
print "<tr><th>Messaggio</th></tr>";
while ($row = mysql_fetch_row($result))
    print "<tr><td>$row[0]</td></tr>";
print "</table>";
```

- supponiamo inoltre che un utente malevolo inserisca nel database la seguente stringa:

```
</td></tr></table><script>alert("colpito!");</script>
```

- il frammento di JavaScript cosi' inserito viene eseguito nei browser di tutti gli utenti che si collegano successivamente all'attaccante
- questo è un buon modo per
  - redirigere tutti gli utenti a un sito diverso
  - carpire password (chiedendole direttamente agli utenti)
  - accedere ai cookies degli utenti (magari per ottenerne l'id di sessione – Session hihacking)
  - ....

Cross-site Scripting (XSS) Attacks (3)

Soluzione:

- dare in pasto i dati ottenuti dagli utenti alla funzione di `htmlspecialchars()` che modifica i dati in modo che i contenuti rimangano gli stessi, ma non possano più essere interpretati come HTML
- nell'esempio:

```
print "<table>";
print "<tr><th>Messaggio</th></tr>";
while ($row = mysql_fetch_row($result))
    print "<tr><td>htmlspecialchars($row[0])</td></tr>";
print "</table>";
```

- come funziona:
  `htmlspecialchars('<script>alert("colpito!");</script>')`
  restituisce:
  `<script$gt;alert(&quot;colpito!&quot;);</script>`