PHP per programmatori C++

Documento in costruzione

Autore: Matteo Lucarelli

Ultima versione su: matteolucarelli.net

Introduzione

Il PHP (l'acronimo sta per processore di ipertesti) è un linguaggio di scripting ampiamente utilizzato per la programmazione lato server. Con questo termine si intende la realizzazione di siti la cui " parte dinamica" viene svolta dal server, nei quali al client viene servito del semplice HTML, ottenuto come output dell'elaborazione PHP.

Il primo problema nella scrittura di codice di questo tipo è nella disponibilità di una piattaforma server per lo sviluppo. Pur essendo infatti possibile utilizzare il PHP come qualsiasi altro interprete nel caso il risultato finale sia volto alle relizzazione di un sito web è indispensabile installare localmente di un web server che ci permetta di testare il risultato come lo vedremo una volta pubblicato.

L'uso come interprete sarà invece prezioso in fase di debugging. Per testare ad esempio la realizzazione di una funzione particolarmente complessa sarà infatti possibile isolare la stessa in un file con estensione .php e passare il comando:

```
Php4 file.php
```

per eseguire il codice come si farebbe con qualsiasi altro script, evitando quindi l'interazione con web server e browser.

Nella maggior parte degli altri casi invece sarà necessario porre le nostre pagine php nella directory del web server (generalmente /var/www/ per i sistemi unix) e visitarle tramite il browser (http://localhost/...).

Il cuore della comprensione del PHP sta proprio nell'interazione client-server, essendo la sintassi in buona parte analoga a quella del linguaggio C. In particolare va tenuto presente che:

- Il programma, a differenza di ciò che avviene con un esegubile locale, NON interagisce in esclusiva con un solo client.
- In ogni successiva richiesta il rapporto client-server " riparte da zero" , non essendo in generale possibile fare affidamento su valori precedentemente memorizzati. Questo a meno di non fare uso di cookie o delle sessioni, il cui campo d'azione è comunque limitato rispetto ad un rapporto dedicato applicativo-utente.
- E' necessario non dare mai per scontata la provenienza dell'utente da una specifica pagina, o la sequenza di visita delle differenti pagine (i cui URL possono essere richiamati in molti modi, oltre a quelli da noi previsti).
- Internet Explorer NON è l'unico browser utilizzato, MS-Windows NON è l'unico sistema operativo esistente.

Inserimento nell'HTML

Il sistema più diretto per inserire il codice PHP è di porlo direttamente all'interno dell'HTM come qualsiasi altro tag:

```
<body>
<?php echo "Hello World!"; ?>
</body>
```

I tag utilizzabili sono

```
<?php ... ?>
<script language=”php”>...</script>
<? ... ?>
<% ... %>
```

Le ultime due forme devono essere abilitate nella configurazione (file " php.ini"). E' possibile inoltre alternare HTML e PHP nel seguente modo:

```
<body>
<?php if (strstr($_SERVER["HTTP_USER_AGENT"], "MSIE")) { ?>
<h3>Stai usando Internet Expolorer</h3>
<?php } else { ?>
<h3>Non stai usando Internet Expolorer
</h3>
```

```
<?php } ?>
</body>
```

NOTA: è consigliabile, per compatibilità, utilizzare i tag in apertura e chiusura anche quando il codice PHP sia posto in file separati (“.php”).

Tipi primitivi

La definizione di tipo è implicita. PHP decide runtime il tipo in funzione del contesto. E’ possibile forzare il tipo tramite la funzione **settype**. Il type casting ha una sintassi analoga al C.

- **boolean**: true/false. Sono considerati falsi: 0,”” (stringa vuota), array vuoti, NULL.
- **integer**, **float** e **double**: numeri interi e in virgola mobile. Float e double sono equivalenti
- **string**: serie di caratteri. Gestita in modo dinamico (praticamente senza limiti di dimensione). Per delimitare le stringhe si possono usare sia gli a pici che le virgolette. Ammette caratteri di escape (“\n”,”\\”,”\t”,”\$”,ecc).
- **resource**: tipo speciale. Contiene il riferimento ad una generica risorsa (file, db, stream, ecc), inizializzata e utilizzata da funzioni dedicate.
- **NULL**: tipo speciale che rappresenta una variabile priva di valore.

La funzione **gettype($var)** restituisce il tipo assegnato alla variabile specificata in argomento. Per ogni tipo esiste una funzione booleana di verifica: **is_int($var)**, **is_float($var)**, ecc.

Variabili

Come già visto le variabili non vanno definite, ma possono essere utilizzate direttamente al momento dell’assegnazione. Il tipo di una variabile può inoltre cambiare durante l’esecuzione per programma.

L’identificativo di una variabile inizia sempre con il segno $. Il nome può contenere qualsiasi carattere alfanumerico e l’underscore. Il primo carattere non può essere un numero. Il nome è case-sensitive.

L’assegnamento (da PHP4) può avvenire **by-reference**. In questo caso la nuova variabile è un alias alla precedente:

```
// da qui in poi modificando $foo si modifica anche $bar
$bar = &$foo;
```

Per eliminare una variabile si usa la funzione **unset($ref)**, per verificare se è settata si usa la funzione **isset($ref)**.

Se non diversamente specificato tutte le variabili hanno scope locale, limitatamente al contesto di definizione. La parola chiave “**global**” importa variabili globali nel contesto corrente:

```
$a = 1;
$b = 2;
function Sum()
{
        global $a, $b;
        $b = $a + $b;
}
```

L’array associativo $GLOBALS contiene tutte le variabili globali utilizzandone il nome come chiave.

La parola chiave “**static**” permette di conservare il valore di una variabile a scope locale tra più chiamate successive di una funzione.

Il PHP ha la particolarità di permettere l’uso di nomi di variabili a loro volta variabili:

```
$a=’hello’
$$a=’ciao’
echo $hello
```

Costanti

Devono essere definite con la funzione “**define**” e si utilizzano senza il “$”:

```
define("COSTANTE", "Ciao mondo.");
echo COSTANTE; // stampa "Ciao mondo."
```

Il nome è case-insensitive. Non possone essere ridefinite o annullate. Lo scope è sempre globale. Possono essere solo valori scalari.

Stringhe

Per delimitare le stringhe si possono usare sia gli apici singoli ' che le virgolette " . La differenza principale è che nelle stringhe tra virgolette le variabili vengono valutate.

Per concatenare le stringhe si utilizza il punto, " ." :

```
$str .= "stringa aggiunta";
```

Le funzioni **htmlentities** e **htmlspecialchars** convertono, in una stringa, tutti i caratteri speciali in entità HTML. La differenza è che la prima converte tutti i caratteri (accentati, simboli, ecc.) mentre le seconda converte solo i caratteri speciali (&,<,>, ecc.);

Esistono numerosissime funzioni per trattare stringhe: **strlen** restituisce la lunghezza, **substr** estrae una sottostringa, **trim** elimina i blank iniziali e finali, **strstr** trova una sottostringa, **strtolower** e **strtoupper** convertono in caratteri minuscoli o maiuscoli, **strcmp** confronta due stringhe, ecc.

Array

Gli array sono associativi (quindi possono contenere coppie chiave-valore oltre che indice-valore) e misti (possono contenere tipi diversi, array compresi), questo significa che un array può avere dimensioni differenti su indici differenti (!). Gli indici float vengono troncati ad intero, gli indici boolean vengono castati ad intero.

```
$arr = array("foo" => "bar", 12 => true);
echo $arr["foo"]; // bar
echo $arr[12];    // 1
```

Gli indici numerici non specificati vengono assegnati automaticamente incrementando l'ultimo indice:

```
// Questi due array sono uguali
array(5 => 43, 32, 56, "b" => 12);
array(5 => 43, 6 => 32, 7 => 56, "b" => 12);

$arr = array(5 => 1, 12 => 2);
$arr[] = 56; // è come scrivere $arr[13] = 56;
```

Per eliminare si usa la funzione **unset** :

```
unset($arr[5]); // Elimina l'elemento '5'
unset($arr);    // Cancella l'intero array
```

la struttura **foreach** (aggiunta in PHP4) realizza una iterazione sugli elementi di un array

```
$arr = array(1, 2, 3, 4);
foreach ($arr as &$value) {
        $value = $value * 2;
}
        // $arr vale ora array(2, 4, 6, 8)- notare il passaggio by-ref

// seconda forma (assegna valore e chiave)
foreach ($arr as $key => $value) {
        echo "Chiave: $key; Valore: $value<br>\n";
}
```

Tramite l'operatore " +" è possibile unire differenti array. Le chiavi duplicate non sono sovrascritte.

Gli array accettano gli operatori di confronto (" ==" ," !=" ,ecc).

Per visualizzare il contenuto e la struttura di un array in maniera chiara esiste la funzione **print_r**.

La funzione **count** ritorna il numero di elementi di un array.

Gli array possono inoltre essere maneggiati come stream, tramite le funzioni **next**,**prev**,**reset**, **current**,ecc. oppure come stack, tramite le funzioni **array_push**,**array_pull**,ecc.

Le funzioni **implode** ed **explode** possono essere utilizzate per convertire facilmente liste-stringa in arrays e viceversa:

```
$txt = "Moto Guzzi, BMW, Honda";
$motociclette = explode(", ", $txt); // crea l'array(Moto Guzzi, BMW, Honda)
```

Variabili autoglobali e predefinite

Sono tutte array associativi. Contengono valori globali che sono definiti automaticamente del PHP.

- $_SERVER["HTTP_USER_AGENT"]: contiene la descrizione del browser utilizzato
- $_SERVER["REMOTE_ADDR"]: indirizzo del client
- $_GET[...]: variabili paasate allo script attraverso il metodo GET
- $_POST[...]: variabili paasate allo script attraverso il metodo POST
- $_COOKIE[...]:
- $_REQUEST[...]: è l'insieme degli array $_GET, $_POST e $_COOKIE

Nelle versioni precedenti la 4.1.0 si utilizzavano array simili con nomi tipo $HTTP_SERVER_VARS. Dalla versione 4.1.0 di php l'uso delle variabili **$HTTP_*_VARS** è deprecato.

L'uso di post e get

I valori " postati" si ottengono tramite il vettore autoglobale **$_POST**.

Ad esempio i dati inseriti in un form in una pagina HTML:

```
<form action="action.php" method="POST">
Nome: <input type="text" name="name" value="" />
Età: <input type="text" name="age" value ="" />
<input type="submit">
</form>
```

saranno disponibili nella pagina action.php (richiamata dalla pressione del pulsante " submit") come $_POST["name"] e $_POST["age"].

Analogamente i valori passati tramite l'URL (http://pagina.php?key=val&key=val&...) si ottengono tramite il vettore autoglobale **$_GET**.

Altre differenze rispetto al linguaggio C

La sintassi dei vari costrutti iterativi e condizionali (**if-then**, **while**, **for**, **switch,** ecc.) è analoga. Il costrutto **if** però ammette anche la parola chiava **elseif**.

```
if ($a > $b) {
        echo "a è maggiore di b";
} elseif ( $a < $b ) {
        echo "a è minore di b";
}
```

Tutti i costrutti ammettono inoltre una sintassi alternativa che consiste nel sostituire alla prima parentesi graffa il carattere "duepunti" (:) ed alla seconda le pèarole chiave **endif**, **endwhile** , **endfor**, **endforeach**, oppure **endswitch**.

```
<?php if ($a == 5): ?>
a è uguale a 5
<?php endif; ?>
```

Esistono le parole chiave **break** (interrompe l'iterazione corrente) e **continue** (passa al ciclo successivo).

il ; (puntoevirgola) è richiesto per separare le istruzioni (finelinea).

I commenti " //" (commenti in linea) e " /* .... */ (più linee). Il Php, come moltri linguaggi di scripting, ammette inoltre l'uso di " #" .

Type casting.

Tutti gli operatori: di confronto (" ==" , " <" , " >" , " !=" , ecc.), aritmetici (+,*,++, %, ecc), sui bit (<<,>>,&,ecc) logici (&&,||,!,ecc) sono utilizzati in modo analogo.

L'operatore booleano "**or**" può essere utilizzato, in modo simile allo shell-scripting, come alternativa sintetica ad una operazione condizionale:

```
function_call() or die("function_call error");
```

Esiste l'operatore condizionale : " test ? primovalore : secondovalore" .

Inclusione

Si utilizzano le istruzioni **include** e **require** . La differenza è che require produce un errore in caso di file mancante. Il file specificato viene incluso e valutato (cioè eseguito).

Quando un file viene incluso l'interprete si porta in modalità HTML, quindi i file PHP dovrebbero sempre essere inclusi nei tag di apertura e chiusura.

Il codice del file incluso eredita lo scope del punto di inclusione.

```
/*
questo è un errore (solo la prima istruzione del file incluso
sarà  condizionata dall'if) In questi casi vanno sempre
utilizzati i blocchi {}
*/
if ($condizione) include $file;
else include $un_altro;
```

Se il codice incluso termina con un'istruzione **return** allora la funzione include assumerà quel valore (altrimenti ritorna 0/1):

Le funzioni di inclusione prevedono le varianti **require_once** ed **include_once** , che evitano l'inclusione multipla dello stesso file, cioè il file verrà incluso una sola volta, alla prima chiamata.

La funzione get_included_files() ritorna un array contenente i nomi dei file inclusi al momento della chiamata.

Funzioni

La dichiarazione di funzioni utilizza una sistassi simile a quella del linguaggio C (a meno delle definizioni di tipo che non servono). La definizione di funzione inoltre non ha un " posto" privilegiato. Può addirittura essere condizionale o interna ad una funzione:

```
if ($makefoo) {
        function foo(){
                echo "foo";
        }
}
```

Se la definizione non è condizionale e non è interna ad una funzione può essere posta anche dopo il codice che la utilizza:

```
<?php
//... chiamata
bar();
//... altro codice
// definizione
function bar()
{
echo "bar\n";
}
?>
```

Naturalmente le funzioni definite all'interno di altre funzioni esistono solo dopo la chiamata della funzione che le contiene:

```
<?php
function foo(){
        function bar(){
                echo "bar\n";
        }
}
/* qui bar() non esiste ancora */
foo();
/* adesso bar() esiste */
bar();
?>
```

Gli argomenti di una funzione sono passati per valore, a meno che non venga specificato tramite " &"

```
function (&$ref_val)
```

In tal caso i cambiamenti al valore effettuati dalla funzione restano validi anche dopo la chiamata

Sono ammessi parametri opzionali con specifica del valore di default:

```
function foo($tipo = "alpha")
```

Sono ammessi nomi di funzioni variabili (anche per la chiamate ai metodi di una classe):

```
function foo() {
```

```
        echo "foo()<br />\n";
}
$func = 'foo';
$func();   // questa chiama foo()
```

Funzioni builtin

La disponibilità delle funzioni builtin dipende dalla configurazione e dalle modalità di installazione di PHP sul sistema in uso. Molte delle funzionalità (database, grafica, ecc) sono infatti disponibili come moduli.

La funzione

```
bool function_exists ( string function_name )
```

verifica se la funzione specificata è definita. Analogamente

```
bool method_exists ( object object, string nome_metodo )
```

verifica se il metodo specificato è definito per la classe specificata. Invece

```
array get_extension_funcs ( string module_name )
```

restituisce un array contenente i nomi di tutte le funzioni definite all'interno del modulo indicato da module_name. Mentre

```
array get_loaded_extensions ( void )
```

Restituisce una matrice con il nome di tutti i moduli compilati e caricati nell'interprete PHP in uso.

Per caricare un modulo runtime è disponibile la funzione **dl(nome_modulo)**. La disponibilità della funzione "dl" è soggetta a restrizioni in funzione della configurazione utilizzata.

Per l'elenco e la descrizione delle funzioni e dei moduli disponibili si veda la Guida alle Funzioni (http://www.php.net/manual/it/funcref.php).

Classi e oggetti

Definizione ed implementazione di una classe avvengono contestualmente:

```
class foo{
        function do_foo(){
                echo "Doing foo.";
        }
}
```

inizializzazione di un oggetto (istanza):

```
$bar = new foo;
```

richiamo di una proprietà o metodo:

```
$bar->do_foo();
```

L'operatore **instanceof** permette di determinare se un dato oggetto appartiene ad una data classe:

```
$oggetto instanceof classe (restituisce true/false)
```

La definizione di classe non ha un "posto" privilegiato. Può addirittura essere interna ad una funzione.

Le definizioni di classe non andrebbero spezzate in più moduli.

Per ogni classe è possibile definire una funzione avente lo stesso nome (**costruttore**) che viene chiamata automaticamente all'istanza di un oggetto di quella classe e serve per inizializzare l'oggetto.

L'ereditarietà è fornita dallo specificatore **extends**. Ed esempio:

```
class NamedCart extends Cart {
  .... }
```

In questo caso la classe NamedCart eredita metodi e proprietà di "Cart" e ne aggiunge di propri. All'interno della classe NamedCart è possibile riferirsi a "Cart" tramite l'operatore **parent**.

Cookies

I cookies sono un meccanismo per memorizzare dati nel computer client. La funzione **setcookie** invia un cookie al client. Ogni cookie inviato dal client sarà automaticamente trasformato in una variabile PHP. Il vettore auto-globale **$_COOKIE** sarà sempre impostato con qualsiasi cookie inviato dal client. I cookie diventano disponibili soltanto dalla pagina successiva a quella che li ha generati, o dopo il ricaricamento di questa.

I cookies sono parte dell'intestazione HTTP, quindi devono essere impostati prima che qualsiasi output sia inviato al browser. Per impostare un cookie all'interno del codice è però possibile usare l'output bufferizzato. La funzione **ob_start** inizia la bufferizzazione dell'output. La funzione **ob_end_flush** invia tutto il buffer.

Serializzazione

La serializzazione consiste nella conversione di un dato, di tipo arbitrario, in una stringa. Questa possibilità si rivela molto comoda per trasmettere o memorizzare dati, in particolare considerando le tipologie di comunicazione web.

Le funzioni **serialize** e **userialize** permettono la conversione a stringa, ed il successivo recupero, di qualsiasi tipo di dato.

Ogni classe può definire le funzioni **__sleep()** e **__wakeup()** che vengono richiamate in automatico all'atto della serializzazione e del ripristino. Tali funzioni possono essere utilizzate per svolgere eventuali operazioni di chiusura o di ripristino.

Gestione della sessione

Consiste nel mantenere dati attraverso accessi successivi, cioè di conservare un rapporto esclusivo tra il browser e il programma. Permette di registrare un numero arbitrario di variabili che vengono preservate.

La funzione **session_start** crea una sessione (o riprende quella corrente basata sull'id di sessione che viene passato attraverso una variabile GET o un cookie). Per assegnare un nome alla sessione è necessario chiamare **session_name** prima di session_start. Utilizzando una sessione basata sui cookie, la chiamata a session_start() va effettuata prima di qualsiasi altro output al browser.

Le variabili di sessione vengono salvate nella array associativo **$HTTP_SESSION_VARS** oppure **$_SESSION**:

```
...TODO
```

Il controllo errori e gestione delle eccezioni

In PHP esiste un operatore di soppressione degli errori: @. Quando viene posto all'inizio di un'espressione gli errori generati dalla stessa non verranno mostrati:

```
// il programma termina senza messaggi
$file = @file ('filechenonesiste') or die;
```

...TODO

trak_errors

$php_errormsg

La funzione "**die($string)**" termina l'esecuzione mandando in output la stringa passata.

try/throw/catch

Interazione con database

Una delle potenzialità maggiori del PHP sta nella facilità di interazione con un database, generalmente tramite SQL (Linguaggio di Interrogazione Standard).

Le istruzioni seguenti si riferiscono al DB MySQL, ma ne esitono di sostanzialmante analoghe per ogni motore DB (PostgreSQL, Oracle, ODBC, ecc.).

Apertura e chiusura di una connessione:

```
$id_db=mysql_connect(host,login,password);
mysql_close();
```

Invio di una interrogazione SQL:

```
$result=mysql_db_query(nome_database,query,$id_db);
```

Per l'utilizzo del risultato dell'interrogazione sono disponibili varie funzioni:

```
// numero di record ottenuti
$num_rows=mysql_num_rows($result);

// numero di campi ottenuti
$num=mysql_num_fields($result);

// parsing dei campi ottenuti
// successivamente disponibili nell'array $row ($row[1], $row[2],ecc.)
$row=mysql_fetch_row($result);
```

Interazione con XML

Ci sono due standard per gestire dati XML in PHP:

- SAX (Simple API for XML)
- DOM (Document Object Model).

L'approccio DOM, più intuitivo e sintetico, ha il difetto di richiedere il caricamento completo del documento in RAM, ed è quindi inadatto a documenti di grandi dimensioni.

Negli esempi ci riferiremo ad un foglio XML avente la seguente struttura (contenuta nel file rubrica.xml):

```
<rubrica>
        <contatto>
                <nome> Mario Rossi </nome>
                <indirizzo> Via verdi,5 </indirizzo>
                <telefono> 0123456789 </telefono>
        </contatto>
        <contatto>
                ...
        </contatto>
        ...
</rubrica>
```

Esempio di parsing con DOM:

```
$doc = new DOMDocument();
$doc->load( 'rubrica.xml' );

$contatti = $doc->getElementsByTagName( "contatto" );
foreach( $contatti as $contatto ){
        $nomi = $contatto->getElementsByTagName( "nome" );
        $nome = htmlentities($nomi->item(0)->nodeValue);

        $indirizzi = $contatto->getElementsByTagName( "indirizzo" );
        $indirizzi = htmlentities($indirizzi->item(0)->nodeValue);

        $telefoni = $contatto->getElementsByTagName( "telefono" );
        $telefono = htmlentities($telefono->item(0)->nodeValue);

        echo "NOME: $nome INDIRIZZO: $indirizzo TELEFONO: $telefono\n";
}
```

Esempio di parsing con SAX:

```
// funziona chiamata all'apertura di un tag
function start_element ($parser, $element_name, $element_attrs) {
        switch ($element_name) {
                case 'nome': echo 'NOME: '; break;
                case 'indirizzo': echo 'INDIRIZZO:'; break;
                case 'telefono': echo 'TELEFONO'; break;
        }
}

// funzione chiamata alla chiusura di un tag
function end_element($parser, $element_name) {
        switch ($element_name) {
                case 'contatto': echo '<br>'; break;
        }
}

// funzione a cui vengono passati i dati tra i tag
function character_data($parser, $data) {
        echo htmlentities($data);
```

```
}

$parser = xml_parser_create();
xml_set_element_handler ($parser, 'start_element', 'end_element');
$fp = fopen('rubrica.xml', 'r');
while ($data = fread($fp, 4096)) {
        xml_parse($parser, $data, feof($fp));
}
xml_parser_free($parser);
```

Esempio di scrittura (avendo i dati in array associativi):

```
<rubrica>
<?php
foreach( $contatti as $contatto ){
        $nome = htmlentities( $contatto['nome'], ENT_QUOTES );
        $indirizzo = htmlentities( $contatto['indirizzo'], ENT_QUOTES );
        $telefono = htmlentities( $contatto['telefono'], ENT_QUOTES );
        ?>
        <contatto>
        <nome><?php echo( $nome ); ?></nome>
        <indirizzo><?php echo( $indirizzo ); ?> </indirizzo>
        <telefono><?php echo( $telefono ); ?> </telefono>
        </contatto>
        <?php
}
?>
</rubrica>
```