R207
Présentation du langage PHP

PHP: Hypertext Preprocessor

IUT de Béziers, dépt. R&T © 2005-2022
http://www.borelly.net/
Christophe.BORELLY@umontpellier.fr

Généralités

- PHP, est un acronyme récursif, qui signifie « PHP: Hypertext Preprocessor »
- C'est un langage de script. Sa syntaxe est empruntée aux langages C, Java et Perl.
- Son utilisation la plus courante est la génération de pages HTML dynamiques.
- Ce langage est apparu dans les années 1994 (Rasmus Lerdorf puis Zeev Suraski et Andi Gutmans 1997)

Différentes utilisations

- On peut utiliser PHP pour :
  - Faire du HTML dynamique (Server side scripting).
  - Faire des scripts en ligne de commande (Command line scripting)
  - Faire des applications graphiques avec l’extension PHP-GTK (Client-side GUI applications)

Syntaxe

- Les instructions PHP sont interprétées entre les balises `<?php` et `?>`.
- Les variables sont identifiées par le caractère `$` suivi d'une chaîne commençant par une lettre.
- Le nom des variables est sensible à la casse.
- Une instruction se termine par `;`

String (1)

- Utilisation de '
- Pas d'expansion des variables.
- Prise en compte de \' et de \\.

```
$x='12';
$a='Texte \'simple\' : $x\n';
// Texte 'simple' : $x\n
```

String (2)

- Utilisation de "
- Expansion des variables.
- Prise en compte des caractères d'échappement : \n (LF 0x0A), \r (CR 0x0D), \t (HT 0x09), \\, \$, \" , \0ooo (code ASCII en octal), \0xhh (code ASCII en hexadécimal).

```
$x='12';
$a="Texte 'simple' : $x\n";
// Texte 'simple' : 12
//
```

Expansion des variables

```
$x="cb";
$n="10.51";
$cb="x";
$v="cbSlack-$n:"; // cbSlack-10.51:
$v="$xSlack-$n:";  // Bug ! Var. xSlack ???
$v="${x}Slack-$n:"; // cbSlack-10.51:
$v="{$x}Slack-$n:"; // cbSlack-10.51:
$v="${$cb}Slack-$n:"; // cbSlack-10.51:
$v="{$$cb}Slack-$n:"; // cbSlack-10.51:
```

Les opérateurs

- Les opérateurs sont ceux du C :
  - = + - * / == != > >= < <= += -= ++ --
- Opérateur ternaire :
  - `condition`**`?`**`val_vrai`**`:`**`val_faux`
- Pour les chaînes : 'toto'=='titi'
- Deux opérateurs supplémentaires :
  - **===** même valeur et même type
  - **!==** non égal ou type différent

Exemples

```php
<?php
/* Exemple de commentaires
sur plusieurs lignes */
$a='cb';      # Crée la variable $a
$$a='Slack'; // Créé la variable $cb
$b=$a=='Slack'; // $b est faux
$A=12;    # Crée la variable $A != $a
$c=($A>10)?23:32; // $c vaut 23
?>
```

La concaténation de chaînes

- L'opérateur de concaténation est . ou .=

```
<?php
$a='Slack';
$b='cb'.$a; // $b vaut cbSlack
// $b vaut cbSlack-10.51
$b.='-'.(6+4).'.51';
?>
```

Opérateur <<<

- Utilisation de **<<<**

```
$s='string';
$str=<<<EOT
Example of $s
spanning multiple lines
using heredoc syntax.\n
EOT;
```

L’affichage

- Plusieurs fonctions d’affichage existent :
  - print(), echo, printf() et fprintf()

```
print('Hello');
echo('Hello');
echo('Hello',' world !!');
$a='world';
printf("Hello %s !!\n",$a);
fprintf(stderr,"Hello %s !!\n",$a);
```

Les fonctions (1)

- Une fonction se définie avec le mot clé function.
- Le mot clé return désigne la valeur de retour de la fonction qui est facultative.

```
function cb($i)
{
  printf("%s\n",$i);
}
$i=12;
cb(6);
// Affiche 6
```

```
function cb2($i,$y=3)
{
  // $y a une valeur par défaut
  return 2*$i+$y;
}
$c=cb2(6);   // $c=15
$d=cb2(6,4); // $d=16
```

Les fonctions (2)

- On peut passer une variable par référence avec le caractère **&** pour pouvoir la modifier dans la fonction.

```
function cb3($i,&$y) {
   $y=$i+2;
   return 2*$i+$y;
}
$z=4;
$c=cb3(6,$z); // $c=20, $z=8
```

Portée des variables

- Dans une fonction, on peut accéder a une variable externe avec le mot clé global ou le tableau associatif « superglobal » $GLOBALS.

```
function cb4() {
  global $y;
  $y+=2;
  $GLOBALS['z']++;
}
$y=2;$z=5;
cb4(); // $y vaut 4 et $z vaut 6
```

Variables static

- Utile en général pour les fonctions récursives. La variable static est unique.

```
function cb5() {
  static $a=0;
  echo "$a\n";
  $a++;
}
cb5(); // $a vaut 0
cb5(); // $a vaut 1
```

Les constantes

- On défini les constantes avec **define**(). Le nom de cette constante est sensible à la casse par défaut.

```
define('CBCONST','Hello world');
define('CB_VAL','cbSlack-1.51',true);
$x=CBCONST; // Accès direct
$y=constant('cb_VAL');
```

- La liste des constantes est obtenue avec **get_defined_constants**().

Les constantes « magiques »

- **__LINE__** : La ligne courante du script
- **__FILE__** : Le fichier courant
- **__FUNCTION__** : La fonction courante
- **__CLASS__** : La classe courante
- **__METHOD__** : La méthode courante

Les structures de contrôle (1)

```
if ($a>1) $a++;
if ($a<0)
{
   $x=2*$a;
   $a=1/$x;
}
if ($a>0) $a=12;
else {
   $x=2*$a;
   $a=1/$x;
}
```

```
if ($a<0)
{
   $x=2*$a;
   $a=1/$x;
}
elseif ($a==0)
{
   $a=3;
}
else $a--;
```

Les structures de contrôle (2)

```php
switch ($a) {
  case 1:$a++;break;
  case 2:$a=3;
  case 20:$a=33;break;
  default:$a=2;
}
switch ($b) {
  case "a":$a++;break;
  case "cb":$a=3;
  case "ab":$a=4;break;
  default:$a=2;
}
```

```php
switch ($a):
  case 1:$a++;break;
  case 2:$a=3;
  case 20:$a=33;break;
  default:$a=2;
endswitch;
```

Les structures de contrôle (3)

```php
$a=0;
do {
  $a+=3;
} while ($a<10);

for ($i=1;$i<=10;$i++) {
  $a+=3;
}
for ($i=0;;$i++) {
  if ($i>10) break;
  $a+=3;
}
```

```php
a=3;
while ($a<10) {
  $a+=2;
}
while ($a>0):
  if ($a==6) {
    $a=3;
    continue;
  }
  $a--;
endwhile;
```

break et continue

```php
for ($i=1;$i<=10;$i++):
  $a+=3;
  for ($j=0;;$j++) {
    // retourne au niveau 2
    if ($i>4&&$j==5) continue 2;
     // sort de la boucle du niveau 2
    if ($i>5&&$j>10) break 2;
    $a+=2;
  }
endfor;
```

Les types de données

- En général, PHP gère lui-même le type des variables suivant le contexte. Mais on peut faire du casting ou utiliser les fonctions de manipulation des variables.
- Les types suivants sont possibles :
  - NULL, boolean, integer, float
  - string, array, object, resource

Fonctions utiles (1)

- Le type d'une variable peut être testé avec une des fonctions :
  - is_null(), is_bool(), is_int(), is_float(), is_string(), is_array(), is_object(), etc...

```
$a=22;
if (is_int($a)) echo "A is int\n";
$b=gettype($a);
$c=settype($a,'string');
var_dump($a); // Affichage de la variable
```

Fonctions utiles (2)

- On peut vérifier si une variable existe avec la fonction **isset**() et effacer une variable avec **unset**() :

```
$a=22;
unset($a); // Efface la variable $a
if (isset($a)) echo "A existe\n";
else echo "A n'existe pas !!!\n";
```

Null et Boolean

- NULL est insensible à la casse et indique une variable sans valeur.

```
$a=null;
if (is_null($a)) echo "A est NULL\n";
```

- Les deux valeurs TRUE et FALSE sont insensibles à la casse.

```
$a=True;
$b=(3<5);
$c=(bool)56;
$d=(boolean)'false';
if ($a==TRUE) echo "Vrai 1\n";
if ($b) echo "3 est inférieur à 5\n";
```

Integer

- Maximum 2^{31}=2 147 483 648

```
$a=1234; // décimal
$a=-123; // décimal négatif
$a=0123; // base 8, octal (83)
$a=0x1A; // base 16, hexadécimal (26)
$b=(int)'12';
$c=intval('12.3 Kg');
$b=(integer)'12.3 Kg';
```

Float

- Maximum environ 1,8 10^{308} (IEEE 64 bits)

```
$a=1.234;
$a=1.2e3;
$a=7E-10;
```

- Casting avec (float), (double) ou (real)

```
$a=(float)'12.3 Kg';
$a=floatval('12.3 Kg');
```

Array (1)

```php
$a=array(4,'toto',6.22);
print_r($a); // Affiche le contenu du tableau
```

```
Array
(
    [0] => 4
    [1] => toto
    [2] => 6.22
)
```

- Tableau associatif (avec indice textuel)

```php
$b=array('user'=>'cb','pass'=>'pwd',4,3=>6.22);
print_r($b);
```

```
Array
(
    [user] => cb
    [pass] => pwd
    [0] => 4
    [3] => 6.22
)
```

Array (2)

```php
$c=array();
$c[]=6;
$c[]=7;
$c['val']=7;
$c[9]='toto';
$c[]=$b;
print_r($c);
unset($c[10]);
```

```
Array
(
    [0] => 6
    [1] => 7
    [val] => 7
    [9] => toto
    [10] => Array
        (
            [user] => cb
            [pass] => pwd
            [0] => 4
            [3] => 6.22
        )
)
```

Opérateurs sur les tableaux

- Union : $a **+** $b
- Égalité : $a **==** $b
  - TRUE si $a et $b ont les mêmes éléments.
- Inégalité : $a **!=** $b ou $a **<>** $b
  - TRUE si $a n'est pas égal à $b.
- Identité : $a **===** $b
  - TRUE si $a et $b ont les mêmes éléments dans le même ordre.
- Non identité : $a **!==** $b

Expansion des tableaux

```php
$x=array('user'=>'cb');
$a="User : $x[user]";       // User : cb
$a="User : {$x['user']}"; // User : cb
$a="User : {$x["user"]}"; // User : cb
// Erreur !!!
// Recherche la constante user !!!
$a="User : {$x[user]}";
```

Structure foreach

- Parcours rapide d'un tableau :

```
$b=array('user'=>'cb','pass'=>'pwd');

foreach ($b AS $val) printf('%s-',$val);

foreach ($b AS $key=>$val) {
  printf("%s : %s\n",$key,$val);
}
```

Fonctions utiles

- **count**($array), **sizeof**($array)
- **in_array**($search,$array,$strict=false)
- **array_key_exists**($key,$array)
- **extract**($array)
- **sort**($array), **asort**($array), **ksort**($array)
- **shuffle**($array)
- **array_merge**($array1,$array2,...)
- ...

Include et require

- Pour faciliter la programmation, on peut fractionner un source PHP en plusieurs fichiers et importer du code PHP avec **`include`**`($file)` ou **`require`**`($file)`.
- Si le fichier n'est pas trouvé (voir directive *include_path*), `include()` génère un warning et `require()` arrête le script (appel de `exit()`).
- Il existe les versions **`include_once`**`()` et **`require_once`**`()` qui vérifient que le fichier n'a pas déjà été importé.

Utilisation de PHP en ligne de commande

```
cb@pccb$ cat hello.php
<?php
printf("Hello %s !\n",$argv[1]);
?>
cb@pccb$ php hello.php cb
Hello cb !
cb@pccb$
```

Script PHP en ligne de commande

```
#!/usr/bin/php
<?php
var_dump($argv);
?>
```

```
cb@pccb$ chmod +x phpTest
cb@pccb$ ./phpTest -h foo
array(3) {
  [0]=>
  string(6) "./phpTest"
  [1]=>
  string(2) "-h"
  [2]=>
  string(3) "foo"
}
cb@pccb$
```

Fichier de configuration

- Le fichier php.ini contient les valeurs par défaut utilisées par les extensions PHP.
- On peut avoir des configurations particulières php-cli.ini ou php-apache.ini.
- On peut le spécifier en ligne de commandes avec l’option -c.
- On peut aussi utiliser la variable d’environnement PHPRC.

Modification dynamique de la configuration

```
$x=ini_get('html_errors');
ini_set('html_errors','off');
ini_set('error_prepend_string','ERREUR :');
ini_set('error_append_string','********');

error_reporting(E_ALL);
ini_set('error_reporting',E_ERROR|E_WARNING|
  E_PARSE|E_NOTICE);
ini_set('error_reporting',E_ALL&~E_NOTICE);
```

Classes et Objets

- Une classe permet de définir des objets.
- Les champs d'un objet sont les variables définies dans la classe.
- Les méthodes d'un objet sont les fonctions définies dans la classe.

```
class SimpleClass
{
 public $x=15;
 function display() {
  printf("X : %d\n",
      $this->x);
 }
}
$o=new SimpleClass();
$o->display();
```

Visibilité des champs et des méthodes

- En PHP4, les champs se définissait avec le mot clé var.
- A partir de PHP5, on peut utiliser les mots clés :
  - public : accès à tous (défaut).
  - private : accès seulement pour la classe.
  - protected : accès pour la classe et les sous-classes.

Utilisation de foreach sur un objet

- L'instruction **foreach** peut-être utilisée sur un objet pour parcourir ses champs visibles.
- Le rendu sera différent suivant la position de l'instruction :

```
x : 1
y : 2
```

```
Objet A :
a : 3
x : 1
y : 2
z : 3.14
```

```php
class A {
   public $x=1,$y=2;
   protected $z=3.14;
   private $a=3;
function disp() {
   echo "Objet A :\n";
   foreach ($this AS $k=>$v)
    echo "$k : $v\n";
}
}
$a=new A();
foreach ($a AS $k=>$v)
    echo "$k : $v\n";
$a->disp();
```

Constructeurs et destructeurs

- En PHP4, le constructeur porte le même nom que la classe.
- A partir de PHP5, le constructeur porte le nom prédéfini :
  - __**construct**().
- Pour des raisons de compatibilité, si cette méthode n'est pas trouvée, on utilise la notation PHP4.
- A partir de PHP5, le destructeur porte le nom :
  - __**destruct**().

Héritage

- Le mot clé extends permet de faire hériter une classe des propriétés d'une autre (une seule).

```
class ExtendClass extends SimpleClass {
  public $y=1;
}
$a=new ExtendClass();
$a->display(); // Affiche 15 cf. SimpleClass
```

Opérateur ::

- Cet opérateur permet d'accéder au membres surchargés, constants ou statiques d'une classe.
- Le mot clé **parent** est utilisé pour la classe supérieure et **self** pour la classe courante.
- On peut accéder aux champs et méthodes static ou const sans instance d'objet :

```php
class A {
   const MSG='Hello';
}
class B extends A {
   public static $nom='world';
   public static function
   display() {
    echo parent::MSG.' '
          .self::$nom."\n";
   }
}
```

```php
echo A::MSG."\n";
B::display();
```

Surcharge des méthodes

- Une sous-classe peut surcharger une méthode si elle n'a pas été déclarée `final`.
- On peut également appeler la méthode parente avec l'écriture `parent::`.

```
class ExtendClass2
 extends SimpleClass {
  public $y=1;
  function display() {
    parent::display();
    printf("Y : %d\n",
      $this->y);
  }
}
```

Méthode __toString()

- Cette méthode permet de définir la chaîne de caractères qui représente l'objet utilisé.
- L'exemple produit :

```
CB : 15
=> Object id #1
=> Object id #1
=> CB : 15
```

```php
class CB {
   private $x=15;
 function __toString() {
   return "CB : $x\n";
 }
}
$cb=new CB();
echo $cb;
echo "=> $cb\n";
echo '=> '.$cb."\n";
echo '=> '.$cb->__toString();
```

Méthode __call()

- Une classe peut utiliser la méthode **`__call()`** pour intercepter les appels à des méthodes non définies dans la classe.
- L'exemple produit :

```
Method : toto()
array(1) {
  [0]=>
  int(1)
}
```

```php
class C {
 function __call($method,$args
 {
echo "Method : $method()\n";
var_dump($args);
 }
}
$c=new C();
$c->toto(1);
```

Méthodes __set() et __get()

- Le même principe existe pour la gestion des champs d'un objet.
- L'exemple produit :

```
X 1
Field : y
Y 12
```

```php
class D {
public $x=1;
 function __get($name) {
echo "Field : $name\n";
return 12;
 }
 function __set($name,$value) {
echo "Field : $name\n";
var_dump($value);
 }
}
$d=new D();
echo 'X '.$d->x."\n";
echo 'Y '.$d->y."\n";
```

Interfaces

- Une interface permet d'imposer la définition de méthodes particulières.
- Une interface ne peut pas contenir :
  - de déclaration de champs.
  - de définition de méthode.
- Une classe peut implémenter plusieurs interfaces.

```
interface Editable {
public function getValue($key);
public function setValue($key,
   $val);
}
class A implements Editable
{
public function getValue($key)
 {...}
public function setValue($key,val)
 {...}
}
```

Les exceptions

- PHP5 propose un système d'exceptions qui permet de gérer les erreurs possibles lors de l'exécution d'un programme utilisant des objets.
- Un bloc try/catch doit être utilisé quand un morceau de code peut envoyer une exception.

```
try {
  // Génère une exception
  throw new
   Exception('Error');
  // Le code suivant une
  // exception n'est pas
   exécuté
  echo 'Never executed';
} catch (Exception $e) {
  echo 'Exception : ',
   $e->getMessage(),"\n";
}
// Suite du prog.
// Même si exception
```

La classe Exception

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

   function __construct($message=null,$code=0);
   function __toString(); // formated string for display
final function getMessage();       // message of exception
   final function getCode();          // code of exception
   final function getFile();          // source filename
   final function getTrace();         // an array of the backtrace()
   final function getTraceAsString(); // formated string of trace
}
```

Le clonage d’objets

- On peut dupliquer les propriétés d’un objet avec le mot clé **clone** : $b=clone $a;
- On peut définir la méthode __**clone**() pour modifier la façon de dupliquer un objet.
- La méthode __**clone**() ne peut pas être appelée directement.

Type hinting

- A partir de PHP5, on peut forcer l'utilisation d'un argument d'une classe donnée dans une méthode ou une fonction.

```
class MyClass {...
function MyMethod (MyClass $foo) {...}
}

function MyFunction (MyClass $foo) {...}
```

Références

- http://www.php.net/
- http://www.w3schools.com/php/
- http://www.php.net/manual/fr/