Opérateurs PHP : la "grammaire" de PHP

Ryle
Modérateur PHPfrance | 10413 Messages
Modérateur

24 avr. 2008, 14:02

Bien que la liste des différents opérateurs soit disponible dans n'importe quel livre ou documentation aux chapitres correspondant, il semblait judicieux de les regrouper également pour avoir une vue d'ensemble de ceux-ci afin de savoir dans quel contexte ils s'utilisent. Voici donc un petit résumé des opérateurs les plus utilisés en php et de leur usage.

- les slashes (//), dièse (#) et slash-etoile (/* ... */) :
Ils servent à mettre en commentaire une partie du code. Le double slash et le dièse sont des commentaires de ligne : ils mettent en commentaire tout ce qui les suit sur la ligne. Le slash-étoile est un commentaire de bloc : il met en commentaire tout ce qui le suit jusqu'à ce qu'il trouve le étoile-slash fermant :

```
$var = 'xx'; // ceci est un commentaire de ligne, on peut aussi utiliser le #

/* ceci est un
commentaire
de bloc */
```

Nota : Vous trouverez parfois des commentaires de bloc commençant par /** au lieu de /*. Cette syntaxe provient de Java qui permet de générer une documentation technique à partir des informations contenues dans les commentaires qui débutent ainsi et qui respectent une syntaxe particulière.

- le point-virgule (;) :
Il sert à spécifier la fin d'une instruction. On en retrouve donc généralement au bout de chaque ligne, à l'exception des blocs d'instructions qui eux sont délimités par des accolades

```
$somme = 1 + 1; // fin de l'instruction un point-virgule
if ($somme == 2) { // pas de ; pour les if, for, while, etc. mais une paire d'accolades
  echo "Ca fait 2";
}
echo "abc"; echo "def"; // on peut avoir deux instructions sur une même ligne
```

- Les opérateurs arithmétiques (+ - * / %) :
Ils servent aux calculs : addition, soustraction, multiplication, division et modulo (reste de la division)

```
$total = 5 + (3 * $valeur) / 2;
```

- Les opérateurs d'affectation (= += -= ....) :
L'opérateur d'assignation le plus simple est le signe "=". Il permet d'affecter la valeur de droite dans la variable de gauche.

```
$a = 3; // affecte la valeur 3 à la variable $a
```

Il existe des "opérateurs combinés" pour tous les opérateurs arithmétiques. Ils permettent d'utiliser la valeur d'une variable dans une expression et d'affecter le résultat de cette expression à cette variable :

```
$a = 3; // affecte la valeur 3 à la variable $a
$a += 5; // équivaut à "$a = $a + 5" ce qui affecte la valeur 5+3 = 8 à la variable $a
```

- Les opérateurs d'incrémentation et décrémentation (++ et --) :
Ces opérateurs pré-incrémente ou post-incrémente (ou décrémente) la valeur d'une variable. Comprendre qu'ils incrémente (ou décrémente) la variable avant ou après l'avoir utilisé.

```
++$a; // incrémente $a de 1, puis retourne $a
$a++; // retourne $a, puis l'incrémente de 1
--$a; // décrémente $a de 1, puis retourne $a
$a--; // retourne $a, puis décrémente $a de 1
```

- Les opérateurs de comparaison (== != === ...)
Les opérateurs de comparaison, comme leur nom l'indique, vous permettent de comparer deux valeurs, voire deux type :

```
$a == $b // Cette comparaison retourne TRUE si $a est égal à $b
$a === $b // Retourne TRUE si $a est égal $b ET si elles ont le même type (String, boolean, int ..)
$a != $b // Retourne TRUE si $a est différent de $b
$a <> $b // Retourne TRUE si $a est différent de $b
$a !== $b // Retourne TRUE si $a est différent de $b OU si elles n'ont pas le même type
```

Il est également possible de comparer des valeurs numérique avec les opérateurs "supérieur ou égal", "supérieur", "inférieur ou égal" et "inférieur" , respectivement : "<=", "<", ">=" et ">".

- le point (.) :
Il sert à la concaténation de chaines :

```
$chaine = "chaine1" . "chaine2";
echo $chaine; // affiche : chaine1chaine2
$chaine .= "chaine3"; // équivaut à $chaine = $chaine . "chaine3"
echo $chaine; // affiche : chaine1chaine2chaine3
```

- la virgule (,) :
Elle sert à séparer les arguments passés en paramètre d'une fonction lors de l'appel et de la déclaration. Elle permet également de séparer les valeurs d'un tableau.

```
function additionne($param1, $param2) { // dans la déclaration
  $total = $param1 + $param2;
  return $total;
}
$resultat = additionne(123, 321); // lors de l'appel
echo $resultat; // affiche : 444
```

- Les opérateurs logiques (&& et ||) :
Ils servent à ajouter des conditions obligatoires (ET) ou facultatives (OU). Il est également possible d'utiliser les instructions "AND" et "OR", mais attention, la priorité de celles-ci est différentes (notamment par rapport à l'opérateur d'affectation) et vous pourriez avoir des résultats innatendues dans vos tests. Si vous avez un doute, privilégiez l'usage des premiers 😃

```
// cette condition est vraie si $val1 ET $val2 sont toutes les deux vraies OU BIEN si $val3 est vraie
if ( ($val1 && $val2) || $val3) {
  ...
}
```

Il existe également un opérateur "XOR" (le shériff de l'espace.. hum..) qui correspond à un OU exclusif. La condition sera remplie si l'un ou l'autre des tests est vérifié, mais pas si les deux le sont en même temps.

- le point d'exclamation (!) :
Il correspond à une négation. Utilisé dans un if, il indique le contraire de la condition spécifiée :

```
$condition= true;
if ($condition) {
  echo 'La variable $condition est à vrai';
}
if (!$condition) { // condition inverse, elle équivaut ici à un else.
  echo 'La variable $condition n\'est pas à vrai';
}
```

Cela peut par exemple s'avérer utile lorsque seul le "else" d'une condition vous intéresse, vous n'avez ainsi pas à implémenter le "if()" associé, mais directement le "if()" équivalent à votre "else".

- Guillemets et apostrophes (' ") :
Elles servent à délimiter une chaine de caractères. A l'intérieur de guillemets les variables sont interprétées, mais ce n'est pas le cas avec les apostrophes. D'une manière générale, il vaut mieux sortir les variables des chaines et utiliser le point pour les concaténer.

```
$chaine1 = "chaine 1";
$chaine2 = 'chaine 2';
echo "Ma chaine est : $chaine1"; // affiche : Ma chaine est : chaine 1
echo 'Ma chaine est : $chaine1'; // affiche : Ma chaine est : $chaine1 (la variable n'est pas interprétée)
echo 'Ma chaine est : ' . $chaine1; // affiche : Ma chaine est : chaine 1
```

- l'antislash (\ ) :
Ilpermet d'échapper un caractère pour souligner un comportement particulier. Pour une apostrophe (ou une guillemet) il signifie que celle-ci ne termine pas la chaine mais doit être interprétée comme le caractère qu'il représente. Dans une chaine délimitée par des guillemets, il n'est pas nécessaire d'échapper les apostrophes, mais les guillemets doivent l'être. De même, il ne faut pas échapper les guillemets d'une chaine délimitée par des apostrophes, mais les apostrophes doivent l'être.
Certains caractères échappés ont également d'autres fonctions et peuvent correspondre à certains caractères spécifiques. Ainsi :

```
"\n" : retour à la ligne
"\t" : tabulation
```

Attention toutefois, en php ceci ne fonctionne que si la chaîne est déclarée entre guillemets.

- le point d'interrogation (?) et les deux points (:) :
Il s'agit de "l'opérateur ternaire". C'est une syntaxe simplifiée de l'instruction if/else :

```
echo (($maCondition) ? 'Le test est vrai' : 'Le test est faux');
// cette syntaxe revient au même que d'écrire :
if ($maCondition)
  echo 'Le test est vrai';
else
  echo 'Le test est faux';

// on peut ainsi facilment l'utiliser pour cocher/décocher par défaut une checkbox :
echo '<input type="checkbox" name="case" '.(($maCondition) ? 'checked' : '').' />';
```

- La flèche simple (->) :
Elles est utilisée lorsque l'on travail avec des "objets", elle permet de faire appel à une méthode ou un attribut d'un objet instancié (pas la peine de te focaliser là dessus si tu débutes 😃)

```
$monObjet = new Objet();
$monObjet->maMethode(); // équivaudrait à appeler une fonction au sein d'un objet
$monObjet->monAttribut; // équivaudrait à appeler une variable au sein d'un objet
```

- La flèche double (=>) :
Elle est utilisée dans les tableaux associatifs, elle permet de définir l'index et la valeur qui lui est associée :

```
$monTableau = array (
  1 => 'Elément 1',
  8 => 'Elément 2',
  'toto' => '...'
)
```

On peut également l'utiliser dans un foreach() pour parcourir chacun des couples clé/valeur d'un tableau associatif :

```
foreach ($monTableau as $cle => $valeur) {
  echo $cle; // affiche 1, puis 8, puis toto
  echo $valeur; // affiche Elément 1, puis Elément 2, puis ...
}
```

- Les doubles deux points (::) :
Ils permettent de faire appel à une méthode ou un attribut statique d'un objet, sans avoir besoin de l'instancier (une fois encore, c'est pas ce qu'il y a de plus important pour commencer 😃)

```
Objet::maMethodeStatic();
```

- l'arobase (@) :
Il s'agit de l'opérateur de silence. Il permet d'empêcher une fonction de retourner un éventuel message d'alerte ou d'erreur :

```
// si le fichier n'existe pas, la fonction unlink() renverrait un message d'alerte
@unlink('dossier/fichier.dat');
/* grace à l'arobase, elle ne le fera pas, et ne polluera pas l'écran en informant l'utilisateur que le fichier qu'il veut supprimer n'existe pas et ne peut être supprimé.. ce dont il doit royallement se moquer ;p */
```

Attention : Si le fichier existe mais qu'il y a une erreur de la suppression (problème de droits ou autre) vous ne le saurez pas non plus... a manipuler avec précaution, voire à éviter en effectuant les tests adéquats pour éviter les erreurs.