PHP CLÉS EN MAIN

76 scripts efficaces pour enrichir vos sites web

par William Steinmetz et Brian Ward

PEARSON

1

TOUT CE QUE VOUS AVEZ TOUJOURS VOULU SAVOIR SUR LES SCRIPTS PHP SANS JAMAIS OSER LE DEMANDER

Les scripts présentés dans ce chapitre répondent à plusieurs questions qui encombrent les forums et les groupes de discussions consacrés à PHP. Parmi elles, citons :

- Comment ajouter des liens *Précédent/ Suivant* à mon panier virtuel ?
- Existe-t-il un moyen simple d'utiliser une couleur différente pour chaque ligne de mon tableau ?
- Je veux trier un gros tableau et je ne sais pas comment faire !
- Quel système de templates puis-je mettre en place pour que mes données soient formatées de la même façon sur toutes les pages ?

Bien que ce livre contienne des scripts assez compliqués et que d'autres vous sembleront plus intéressants, ceux qui sont présentés ici répondent aux questions que nous rencontrons sans cesse. Ces scripts pour débutants représentent ce que tout le monde *devrait* savoir ou *voudrait* savoir. Faites lire ce chapitre à un programmeur en PHP que vous appréciez et il vous en sera reconnaissant.

NOTE *Si vous n'avez pas peur de jouer le rôle d'administrateur de votre serveur web, le Chapitre 2 vous aidera également à progresser si vous débutez en PHP ; il vous facilitera également la vie si vous avez déjà un peu programmé avec ce langage.*

Recette 1 : Inclure un fichier extérieur dans un script

La plupart des applications sérieuses utilisent un ensemble de variables et de scripts qui seront repris par quasiment toutes les pages. Si vous concevez, par exemple, un panier virtuel qui se connecte à une base de données MySQL, vous pouvez déclarer le nom d'utilisateur et le mot de passe MySQL dans chaque page du panier, mais que se passera-t-il si vous devez changer ce mot de passe ? La modification et la mise à jour de chaque fichier sur le serveur pourront alors devenir un gros problème.

Au lieu de déclarer le mot de passe dans chaque page de script, vous pouvez stocker le nom d'utilisateur et le mot de passe dans un fichier séparé. Vous pourrez ensuite inclure ce fichier dans votre script et toutes les variables déclarées dans le fichier seront automatiquement déclarées dans le script !

En outre, vous pouvez également stocker des scripts ou des fonctions dans un fichier et ne les inclure que lorsque vous en avez besoin. La fonction permettant de suivre en temps réel les expéditions UPS, par exemple, représente 24 Ko de traitement XML, mais elle ne sert que lorsqu'un client choisit UPS comme type d'expédition. Pourquoi ne pas alors la stocker dans un fichier *suivi_ups.php* et ne l'appeler que lorsque cela s'avère nécessaire ?

En réalité, quasiment toutes les applications PHP professionnelles emploient un fichier portant un nom comme *config.php*, qui contient les déclarations des variables essentielles utilisées par toutes les pages : le nom d'utilisateur et le mot de passe MySQL, par exemple. Ces applications stockent également les scripts utilitaires dans des répertoires distincts : les programmeurs peuvent alors faire de savants mélanges en prenant le script qui vérifie-si-un-utilisateur-est-connecté dans un répertoire et le script qui récupère-les-données-de-la-base dans un autre. Il leur reste à écrire un script qui adapte la vue des données en fonction de la connexion ou non de l'utilisateur. Voici comment faire :

```
<?php
require_once("/chemin/vers/fichier.php");
?>
```

Le fichier passé à `require_once()` fait désormais partie du script, exactement comme si vous aviez copié son contenu pour le coller dans le script. Vous pouvez même inclure des fichiers HTML pour créer un système de templates rudimentaire.

Quel que soit le nom du fichier, PHP tentera de lire son contenu comme du code PHP. Comme pour tout fichier PHP, vous devez donc entourer le code PHP contenu dans ce fichier par les balises `<?php` et `?>` ; sinon, l'interpréteur se

contentera d'afficher le contenu brut de tout le fichier (même s'il est codé en binaire).

La fonction `require_once()` s'utilisant comme n'importe quelle autre instruction, vous pouvez donc l'intégrer dans une structure de contrôle :

```
if ($fichier_necessaire === true) {
   require_once("fichier_necessaire.php");
}
```

Problèmes éventuels

Plusieurs choses peuvent mal tourner lorsque l'on inclut un fichier.

Le chemin vers le fichier est incorrect.

En ce cas, le script se termine en signalant une erreur fatale. Si vous préférez qu'il poursuive son exécution même s'il n'a pas trouvé le fichier à inclure, utilisez `include_once()` à la place de `require_once()`.

Le chemin vers le script est correct, mais il se trouve dans un répertoire inaccessible en lecture.

Ce problème peut survenir si vous avez configuré la variable `open_basedir` pour restreindre les répertoires auxquels PHP a accès. Pour des raisons de sécurité, les développeurs web limitent l'accès aux fichiers et aux répertoires importants. Nous expliquerons comment modifier les droits d'accès aux répertoires à la recette n°16, "Restreindre l'accès de PHP aux fichiers".

Le fichier inclus contient une ligne blanche ou une espace avant ou après le code du script PHP.

Si votre script met en place des cookies ou effectue un traitement qui n'est pas conforme à HTTP, il doit le faire *avant* d'envoyer quoi que ce soit au navigateur. N'oubliez pas que PHP affiche tout ce qui ne se trouve pas entre les balises `<?php` et `?>` dans un fichier inclus : si une ligne blanche se trouve avant ou après ces balises, elle sera donc envoyée au navigateur comme s'il s'agissait de code HTML, ce qui empêchera la mise en place des cookies et le démarrage des sessions. Si vous incluez un script, vérifiez qu'il n'y a pas d'espace à l'extérieur des balises PHP. Faites particulièrement attention aux espaces après la balise de fin `?>` car généralement ils restent invisibles dans un éditeur de texte.

NOTE *Les cookies servent à suivre la trace des utilisateurs et à stocker des informations cachées. Consultez le Chapitre 8 pour plus de détails.*

Le fichier inclus peut être lu par des méthodes non PHP.

Vous pouvez stocker des variables PHP dans n'importe quel fichier, quel que soit son nom, mais si vous n'indiquez pas à Apache que ces fichiers doivent être protégés en écriture, il enverra leur contenu en texte brut à quiconque lui demande. Si vous n'y prenez pas garde, toute personne connaissant le nom de vos fichiers inclus pourra donc les lire.

Il est inutile de préciser que le stockage des mots de passe et des noms de comptes MySQL dans un emplacement qui peut être lu par Internet Explorer n'est pas considéré comme une bonne mesure de sécurité.

Pour améliorer la sécurité, vous pouvez placer les fichiers inclus à l'extérieur des répertoires du serveur web (de préférence, dans un répertoire dont l'accès est protégé par un mot de passe), afin que les scripts ne puissent être accédés que par FTP. Si vous manipulez des données sensibles, comme des données bancaires, vous devriez vous sentir obligé de prendre ces mesures.

NOTE *Nous verrons comment valider un numéro de carte bancaire à la recette n°30, "Vérifier la validité d'une carte de crédit".*

Vous êtes perdu dans les inclusions.

Un jour, j'ai acheté un programme de panier virtuel parce qu'il était écrit en PHP et que je comptais adapter les scripts pour mes besoins professionnels. Imaginez ma surprise lorsque j'ai constaté que son module principal (celui où les visiteurs ajoutent, suppriment et modifient les articles) contenait 7 inclusions, 12 lignes de code et 3 templates Smarty. J'ai ouvert l'un des fichiers inclus et j'ai découvert – vous l'aurez deviné – qu'il contenait à son tour trois inclusions.

Les fichiers inclus permettent de rendre votre code très compact mais, croyez-moi, si vous tentez de déchiffrer un script faisant appel à de très nombreux fichiers inclus, c'est l'enfer. Pour la santé mentale des autres programmeurs et des générations futures, *n'incluez pas un fichier sans ajouter un commentaire indiquant aux autres ce que fait ce fichier,* merci.

Vous utilisez des variables non vérifiées comme noms de fichiers inclus.

Bien que vous puissiez écrire `include($fichier)` pour inclure un script en fonction du choix de l'utilisateur, cette pratique permet à un pirate d'inclure n'importe quel fichier du site avec un minimum d'effort ou, selon la configuration de votre serveur, un fichier de *son* site qui s'introduira donc sur votre serveur. En fait, quelques virus PHP utilisent ce genre de faute de programmation. En outre, ce genre de scripts sont bien plus sujets aux bogues et sont généralement impossible à relire.

Si vous devez inclure des fichiers dont le nom provient d'une variable, utilisez plutôt un script comme celui de la recette n°28, "S'assurer qu'une réponse fait partie d'un ensemble de valeurs", afin de vérifier que les noms de fichiers sont corrects et éviter ainsi qu'un pirate puisse lire votre fichier de mots de passe.

Recette 2 : Alterner les couleurs des lignes d'un tableau

Si vous devez présenter un grand nombre d'informations sous la forme de lignes d'un tableau, comme les sujets d'un forum de discussion ou les articles d'un panier virtuel, les différentes lignes seront bien plus faciles à lire si chacune d'elles a une couleur légèrement différente de celles des lignes qui l'entourent. La première étape consiste à définir les couleurs des lignes de tableau dans votre feuille de style.

```
tr.lig1 {
   background-color: gray;
}

tr.lig2 {
   background-color: white;
}
```

Ici, on a défini deux classes de style, `lig1` et `lig2`, pour les balises de lignes de tableau (`<tr>`). Vous pouvez les intégrer dans un fichier CSS inclus par votre document ou les placer entre les balises `<style>` et `</style>` dans la partie `<head>` du document.

Définissons maintenant une fonction qui retourne alternativement ces classes. Nous utiliserons une petite astuce consistant à passer par référence une variable entière à cette fonction, qui pourra ainsi modifier cette variable qui servira de bascule pair/impair :

```
function formater_ligne_tableau(&$cpteur_lig) {
  // Renvoie la classe de style pour une ligne
  if ($cpteur_lig  & 1) {
    $couleur_lig = "lig2";
  } else {
    $couleur_lig = "lig1";
  }
  $cpteur_lig++;
  return $couleur_lig;
}
```

Voyons maintenant comment utiliser cette fonction. On crée d'abord une requête SQL pour obtenir quelques lignes de données à partir de la table décrite dans l'annexe, puis on lance le formatage du tableau :

```
$sql = "SELECT nom_produit FROM infos_produits";
$resultat = @mysql_query($sql, $db) or die;

echo "<table>";
```

Le reste est assez simple ; il suffit d'initialiser la variable bascule `$i`, d'appeler `formater_ligne_tableau($i)` sur chaque ligne afin d'obtenir des classes de style alternées et d'associer à chaque ligne du tableau la classe ainsi obtenue. Enfin, il reste à fermer le tableau :

```
$i = 0;
while($lig = mysql_fetch_array($resultat)) {
  /* Affichage du résultat */
  $classe_lig  = formater_ligne_tableau($i);
  echo "<tr class=\"$classe_lig\"><td>$lig[nom_produit]</td></tr>";
}
echo "</table>";
```

Il est essentiel de comprendre comment fonctionne `formater_ligne_tableau()` : il faut initialiser une variable entière pour représenter l'état, mais la valeur en elle-même n'a aucune importance puisque la fonction s'en occupe pour vous.

Amélioration du script

Vous pouvez faire un million de choses avec les feuilles de style. Pour plus d'informations sur les CSS, consultez l'une des nombreuses ressources en ligne consacrées à ce sujet ou lisez le livre d'Éric Meyer, *CSS par Éric Meyer* (Pearson, 2005). En utilisant une approche objet, vous pouvez aisément convertir cette fonction en une méthode de formatage de tableau : au lieu de déclarer explicitement une variable d'état, il suffit de créer une classe avec une variable privée chargée de mémoriser cet état. Le constructeur et le destructeur ouvrent et ferment, respectivement, les balises du tableau. Voici à quoi ressemblera cette classe :

```
class TableauAlterne {
  function __construct() {
    $this->etat = 0;
    print "<table>";
  }
  function __destruct() {
    print "</table>";
  }
  function affiche_ligne($ligne) {
    if ($this->etat & 1) {
      $couleur_lig = "lig2";
    } else {
      $couleur_lig = "lig1";
    }
    print "<tr class=\"$couleur_lig\">";
    foreach ($ligne as $valeur) {
      print "<td>$valeur</td>";
```

```
    }
    print "</tr>";
    $this->etat++;
  }
}
```

Voici comment utiliser cette classe (en supposant que la requête SQL est identique à celle de l'exemple précédent) :

```
$montableau = new TableauAlterne;
while($ligne = mysql_fetch_row($resultat)) {
  /* Affichage du résultat. */
  $montableau->affiche_ligne($ligne);
}
unset($montableau);
```

Au premier abord, vous pourriez penser que c'est une belle amélioration car elle est un peu plus simple à utiliser – il n'est plus nécessaire de se préoccuper de la variable d'état, la méthode `affiche_ligne()` peut gérer un nombre quelconque de colonnes et vous n'avez plus non plus besoin d'écrire le moindre code HTML pour ouvrir et fermer le tableau. C'est, en effet, un avantage indéniable si tous vos tableaux se ressemblent, mais cette apparente simplicité se paye en termes de souplesse et d'efficacité. Vous pourriez bien sûr ajouter des méthodes et des attributs pour effectuer des traitements supplémentaires, comme l'ajout d'en-têtes aux tableaux, mais demandez-vous si cela en vaut réellement la peine.

Recette 3 : Créer des liens Précédent/Suivant

Si vous devez afficher un grand nombre d'articles sur une page, il peut être souhaitable de découper cet affichage en plusieurs pages contenant, chacune, un nombre limité d'articles. Vos résultats seront ainsi plus faciles à lire et vous améliorerez le temps de chargement des pages.

Une barre de navigation permet aux utilisateurs de contrôler le chargement de ces pages. Vous avez besoin d'une barre comportant des liens *Précédent* et *Suivant*, ainsi que des liens permettant d'atteindre une page précise d'après son numéro. Voici un script qui prend tout cela en charge :

```
<?php

function creer_navbar($num_deb  = 0, $articles_par_page = 50, $nbre) {
  // Création d'une barre de navigation
  $page_courante = $_SERVER["PHP_SELF"];

  if (($num_deb  < 0) || (! is_numeric($num_deb))) {
    $num_deb = 0;
  }
```

```
  $navbar = "";
  $navbar_prec = "";
  $navbar_suiv = "";

  if ($nbre > $articles_par_page) {
    $cpteur_nav = 0;
    $nb_pages = 1;
    $nav_passee = false;

    while ($cpteur_nav < $nbre) {
      // Est-on sur la page courante ?
      if (($num_deb <= $cpteur_nav) && ($nav_passee != true)) {
                $navbar .= "<b><a href=\"$page_courante?debut=$cpteur_nav\">
                           [$nb_pages]</a></b>";
       $nav_passee = true;
       // Faut-il un lien "Précédent" ?
       if ($num_deb != 0) {
         $num_prec = $cpteur_nav - $articles_par_page;
         if ($num_prec < 1) {
           $num_prec = 0;
         }
         $navbar_prec = "<a href=\"$page_courante?debut=$num_prec \">
                        &lt;&lt;Précédent - </a>";
      }

      $num_suiv = $articles_par_page + $cpteur_nav;

      // Faut-il un lien "Suivant" ?
      if ($num_suiv < $nbre) {
        $navbar_suiv = "<a href=\" $page_courante?debut=$num_suiv\">
                     - Suivant&gt;&gt; </a><br>";
      }
    } else {
      // Affichage normal.
      $navbar .= "<a href=\"$page_courante?debut=$cpteur_nav\">
                [$nb_pages]</a>";
    }

    $cpteur_nav += $articles_par_page;
    $nb_pages++;
  }

  $navbar = $navbar_prec . $navbar . $navbar_suiv ;
  return $navbar;
  }
}

?>
```

Supposons que vous utilisiez ce script pour traiter des informations provenant d'une base de données. Il y a deux points à respecter pour que l'affichage soit précis et efficace :

N'afficher qu'un sous-ensemble de lignes de votre base de données.

Si vous souhaitez n'afficher que 25 articles par page, écrivez une requête SQL qui ne renvoie que 25 lignes (vous *pourriez* tout récupérer, puis parcourir les milliers de lignes de résultats pour ne garder que les 25 qui vous intéressent, mais il existe des méthodes bien plus efficaces). En outre, vous voulez ne montrer que 25 articles *spécifiques*. Récupérer les 25 premiers articles de la base ne vous aidera pas si l'utilisateur veut consulter les produits correspondant aux articles 176 à 200.

Heureusement, la clause `LIMIT` de SQL permet d'atteindre aisément ces deux objectifs puisqu'elle permet de ne récupérer qu'un certain nombre de lignes dans une base de données. La requête SQL sera donc de la forme :

```
SELECT * FROM votre_table
WHERE conditions
LIMIT $num_deb, $articles_par_page
```

Lorsqu'elle est ajoutée à la fin d'une requête, la clause `LIMIT` n'extrait que le nombre indiqué de lignes de l'ensemble résultat. En mettant, par exemple, `LIMIT 75, 25` à la fin d'une requête, on ne récupère que 25 lignes de la base à partir de la 75e ligne.

NOTE *Si le numéro de la ligne de début est supérieur au nombre de lignes disponibles, MySQL ne renvoie aucune donnée. Si, par exemple, vous utilisez une clause `LIMIT 200, 25` alors que la table ne contient que 199 lignes, vous obtiendrez un ensemble résultat vide, pas une erreur. Les programmes bien écrits prennent en compte les ensembles vides en faisant un test `if (mysql_num_rows($resultat) > 0)`.*

Maintenant que vous n'avez que les lignes que vous souhaitiez, il vous reste une chose à faire.

Compter le nombre total de lignes de l'ensemble résultat.

Ensemble résultat est le terme employé par SQL pour signifier "toutes les données qui ont été renvoyées après le traitement de la clause WHERE". On a besoin du nombre total de lignes pour savoir sur quelle page on se trouve et combien il reste de pages avant la fin. Sans lui, nous pourrions afficher un lien *Suivant* alors qu'il n'y a plus d'articles à voir et nous ne pourrions pas indiquer à l'utilisateur le nombre de pages qu'il lui reste à consulter.

La requête SQL devrait être de la forme :

```
SELECT count(*) AS nombre FROM votre_table
WHERE conditions
```

Lorsque vous disposez de cette information, vous pouvez greffer les trois informations dans la fonction `creer_navbar()`. Celle-ci contient plusieurs variables :

`$page_courante` : La page courante qui contient la barre de navigation. Dans ce script, nous utilisons la variable spéciale `$_SERVER["PHP_SELF"]` qui contient toujours la page courante sans le nom d'hôte ni les éventuels paramètres GET. Dans un script accessible à l'URL **http://exemple.com/navbar.php?debut=0**, par exemple, `$page_courante` vaudrait */navbar.php*.

`$num_deb` : Le premier numéro de ligne. Si, par exemple, l'utilisateur examine les lignes 100 à 125, ce numéro vaudra 100. Il est passé à l'URL *via* un paramètre GET. Par défaut, il vaut 0.

`$articles_par_page` : Le nombre de lignes affichées sur chaque page. Si, par exemple, il y a 100 lignes de données, une valeur de 25 pour cette variable produira quatre pages de 25 lignes chacune. Ces mêmes 100 lignes avec une valeur de 50 pour `$articles_par_page` produirait deux pages de 50 lignes. Si cette variable vaut 200, un ensemble de 100 lignes n'aura pas de barre de navigation puisque tout tient sur une seule page. La valeur par défaut de cette variable est 50.

`$nbre` : Le nombre total de lignes. Nous avons déjà expliqué comment obtenir cette information dans "Compter le nombre total de lignes de l'ensemble résultat".

`$cpteur_nav` : Cette variable part de 0 et s'incrémente de `$articles_par_page` jusqu'à ce qu'elle devienne supérieure à `$nbre`, auquel cas le script a atteint la fin de l'ensemble résultat – et donc la fin de la barre de navigation.

`$nb_pages` : Le nombre de pages dans l'ensemble résultat.

`$nav_passee` : Une variable temporaire qui passe à vrai dès que `$cpteur_nav` dépasse `$num_deb` ; en d'autres termes, nous sommes sur la page courante. Cela permet de mettre en évidence le numéro de la page courante lors de l'affichage de la barre de navigation.

Utilisation du script

```
<?php

$num_deb  = intval($_GET("debut"));
$articles_par_page = 100;
$categorie_ventes = null;

if ($num_deb >= 0) {
```

```
  // Compte les articles de la catégorie.
  $sql = "SELECT count(*) AS nombre FROM infos_produits
          WHERE categorie = 'chaussures'";
  $resultat = @mysql_query($sql, $chaine_connexion)
       or die("Erreur :  " . mysql_error());
  while ($ligne = mysql_fetch_array($resultat)) {
    $nbre = $ligne['nombre'];
  }

  // Récupère le résultat à afficher pour le client.
  $sql = "SELECT num_produit, nom_produit FROM infos_produit
          WHERE categorie = 'chaussures'
          LIMIT $num_deb, $articles_par_page";
  $resultat = @mysql_query($sql, $chaine_connexion)
       or die("Erreur :  " . mysql_error());
  if (mysql_num_rows($resultat) > 0) {
    while($ligne = mysql_fetch_array($resultat)) {
      // Parcourt des lignes et ajoute des balises HTML
      // dans $categorie_ventes.
      $categorie_ventes .= "Données provenant de la requête SQL";
    }
  } else {
     $categorie_ventes = "Aucun article n'appartient à cette catégorie.";
  }

  $navbar = creer_navbar($num_deb, $articles_par_page, $nbre);
}

if (is_null($categorie_ventes)) {
  echo "Entrée incorrecte.";
} else {
 echo "$navbar<br />$categorie_ventes";
}

?>
```

Cet exemple montre comment utiliser la fonction `creer_navbar()` avec un ensemble de données provenant d'une table SQL (le contenu de cette table est détaillé en annexe). Ce script fonctionne de la façon suivante :

1. Le premier numéro de ligne est extrait d'un paramètre GET et placé dans la variable `$num_deb`.
2. Une première requête SQL permet d'obtenir le nombre total de lignes concernées dans la table.
3. Une seconde requête extrait au plus `$articles_par_pages` lignes de la table, à partir de la ligne `$num_deb`.
4. Les données de la ligne sont formatées.