PHP - Méthodes de débogage

Juin 2014

- Informations sur une variable
  - echo
  - print_r()
  - var_dump() / var_export()
  - Rediriger la sortie (ne pas afficher à l'écran)
- "Backtrace" ou rapport de contexte
- Bien utiliser les rapports d'erreur
- Le mot clé global
- Le debuggueur APD
- Compléments

Le langage PHP est bien pourvu en fonctions de débogage, il sera difficile après cela de trouver une excuse pour ne pas essayer de corriger vos erreurs par vous même ;-) Toutes ces méthodes se rapportent à l'affichage d'informations sur des variables ou sur le contexte dans un script. Elles se rapportent également aux mauvaises habitudes qu'il faut adopter avec minutie...

Informations sur une variable

echo

La première fonction, la plus basique pour afficher la valeur d'une variable est bien sûr "*echo*". Elle peut afficher un nombre entier, une chaine de caractères ou un nombre réel. C'est une habitude à prendre : une requête SQL ne fonctionne pas ? Alors stockez là dans une variable puis affichez-la avec un echo avant de l'exécuter. Vous pourrez savoir si le problème vient d'une variable qui possède une valeur incohérente voire une valeur **null**. Vous pourrez également tester votre requête plus directement en copiant/collant celle qui est affichée sur votre page web. Par exemple si vous utilisez MySQL, vous pouvez tester votre requête à l'aide de PhpMyAdmin ou avec le client MySQL officiel (en passant par une console Linux ou une fenêtre Dos sous Windows).

print_r()

La fonction echo est pratique mais insuffisante dans certains cas. Elle ne peut pas afficher un tableau ni les caractéristiques d'un objet. Pour ça, "*print_r*" est plus avancé. Cette fonction affichera la valeur pour un entier, une chaine ou un réel et vous présentera de façon structurée

un tableau ou un objet.

var_dump() / var_export()

Plus verbal encore que son précédent: la fonction *var_dump()* vous donnera le type d'une variable ou des champs d'un tableau etc... ainsi que leur contenu exact et, pour les chaines, le nombre de caractères. Si la variable contient une valeur "vide ou fausse" comme 0, '' (chaine vide), Null ou False, vous aurez le type et la valeur exacte. *Var_export()* est à peu près similaire mais présente ces informations sous forme de code PHP.

Rediriger la sortie (ne pas afficher à l'écran)

Ce que j'appellerai ici la "sortie" sont les données qui s'affichent normalement à l'écran. Il se peut que vous ayez besoin de consigner les données rapportées par var_dump ou print_r dans un fichier plutôt qu'à l'écran ou dans une page web. Imaginons que vous vouliez consigner ces infos dans un fichier nommé test.log Vous devrez alors garder en mémoire les données de sortie à l'aide de la fonction *ob_start()* puis, à la fin de votre script, mettre cette sortie dans une variable avec *ob_get_contents()* puis effacer ces données destinées à l'écran avec *ob_end_clean()* Il suffira enfin d'écrire ces données dans le fichier. Voici donc ce que ça donne: J'ai un tableau nommé *$tab*, je veux consigner sa description dans le fichier test.log plutôt qu'à l'écran:

```
<?
ob_start();
$tab=array(1=>'test',
 2=>'test2');
var_export($tab);

$tab_debug=ob_get_contents();
ob_end_clean();

$fichier=fopen('test.log','w');
fwrite($fichier,$tab_debug);
fclose($fichier);
?>
```

A présent, les détails du tableau $tab sont sauvegardés dans le fichier et rien n'a été affiché à l'écran ou sur la page web.

"Backtrace" ou rapport de contexte

Un backtrace, ou rapport de contexte (si quelqu'un trouve mieux comme traduction, n'hésitez pas...) vous permettra de retracer le chemin à travers les appels de fonctions et les inclusions de fichiers pour arriver à un point donné dans le code. Prenons un exemple inspiré du site de PHP: fr.php.net/manual/fr/function.debug-print-backtrace.php Nous avons une première page (fonctions.php)

```
<?php

function b()
{
 var_dump(debug_backtrace());
}
function a()
{
 b();
}

a();

?>
```

Puis un deuxième fichier qui appellera fonctions.php (on l'appellera main.php):

```
<?php
include 'fonctions.php';
?>
```

Voici ce que l'on obtient:

```
array(3) {
  [0]=>
  array(4) {
    ["file"]=>
    string(29) "/fonctions.php""
    ["line"]=>

    int(10)
    ["function"]=>
    string(1) "b"
    ["args"]=>
    array(0) {
    }
  }
  [1]=>
  array(4) {
    ["file"]=>
    string(29) "/fonctions.php""
    ["line"]=>
    int(13)
    ["function"]=>
    string(1) "a"
    ["args"]=>
    array(0) {
    }
  }
  [2]=>
  array(4) {
    ["file"]=>
    string(30) "/main.php"
    ["line"]=>
    int(3)
    ["args"]=>
    array(1) {
      [0]=>
      string(29) "/fonctions.php"
    }
    ["function"]=>
    string(7) "include"
  }
}
```

Nous avons trois tableaux, l'analyse se fait en partant du dernier pour arriver jusqu'au premier si l'on veut retracer le cheminement du code depuis le début. Dans chaque tableau nous avons: **function** : la fonction qui est appelée **args**: les arguments passés à cette fonction (ici, le chemin vers fonctions.php) **line**: la ligne concernée dans le code **file**: le chemin vers le fichier php en cours de traitement. Dans cette longue enfilade, pour arriver jusqu'au point de traçage (debug_backtrace()), on remarque que l'on passe par la fonction include dans le fichier principal

(main.php) puis par a() et enfin par b() dans le fichier fonctions.php. Ça peut être utile pour voir comment se comporte le code en suivant l'ordre d'appel des fonctions et la valeur des paramètres qui lui sont passés en arguments. La fonction debug_backtrace() retourne un tableau contenant le rapport de contexte depuis l'endroit ou vous l'avez appelée. Je l'ai affichée directement à l'aide de var_dump() mais vous pouvez l'utiliser comme vous le voulez, le stocker dans une variable, etc. La fonction debug_print_backtrace (PHP 5) vous permettra d'afficher directement le rapport.

Bien utiliser les rapports d'erreur

PHP détaille bien les erreurs sur les pages web (ou à l'écran si vous utilisez PHP pour une application qui n'est pas destinée au web). N'hésitez pas à les regarder en détail. Il se peut que vous ayez désactivé ce rapport d'erreur en utilisant l'opérateur @ Utilisez-le avec parcimonie, il est utile car il peut par exemple vous permettre de ne pas donner de renseignements sur votre base de données dans un rapport d'erreur, mais il faut tout de même penser à gérer cette erreur qui ne s'affichera pas. De même, signaler à PHP de vous rapporter toutes les erreurs (configuration par défaut généralement), grâce à la fonction error_reporting, vous permettra de mieux vous y retrouver:

```
error_reporting(E_ALL);
```

Le mot clé global

Il existe en PHP un mot clé qui s'appelle "*global*". Il vous permet d'utiliser une variable de portée globale à l'intérieur d'une fonction. **Fuyez ce mot-clé!** Ne l'utilisez pas, à moins que votre vie soit en jeu.... Cette fonctionnalité rend le code illisible, et son débuggage difficile à suivre.

Le debuggueur APD

PHP possède également des fonctions de débuggage avancé regroupées sous le nom de APD.

Compléments

- Article sur le débuggage PHP par "LePhpFacile.com"