Comment déboguer en PHP

par Kieran Masterton (Auteur) Yannick Komotir (Traducteur)

Dernière mise à jour :

Cet article est la traduction de : **How to Debug in PHP**.

I - Introduction

Personne n'apprécie le processus de débogage du code. Cependant, si vous voulez écrire des applications web mortel, il est essentiel de comprendre completement ce processus. Cet article décompose les principes fondamentaux du débogage en PHP, en vous aidant à comprendre les messages d'erreur PHP et en vous présentant quelques outils utiles d'aide qui rendent ce processus moins douloureux.

II - Préparer le terrain

Il est important de configurer PHP correctement et d'écrire le code de telle sorte qu'il produise des erreurs significatives au bon moment. Par exemple, en géneral c'est dans les bonnes habitudes d'activer un niveau bavard de reports d'erreurs sur votre plateforme de développement. Ce qui n'est pas une bonne idée de faire de même sur votre(vos) server(s) de production. Dans un environnement live vous ne voudriez pas confondre un véritable utilisateur d'un utilisateur malveillant en lui fournissant beaucoup trop d'informations sur le fonctionnement interne de votre site. Ainsi, avec cela à l'esprit, parlons de la célèbre phrase : "Je n'ai aucun message d'erreur". Ceci est normalement provoqué par une erreur de syntaxe sur une plateforme où le développeur n'a pas bien préparé le terrain correctement. D'abord, activer display_errors. Ceci peut être fait dans votre fichier php.ini ou au début de votre script comme ceci :

```
<?php
ini_set('display_errors', 'On');
```

Dans ces exemples j'omet la balise fermante PHP (?>). On le considère généralement comme bonne pratique de faire ainsi dans les dossiers qui contiennent seulement le code de PHP afin d'éviter l'injection accidentelle des espaces blanc et tous les erreurs commune □headers already sent□

Ensuite, determiner un niveau de rapport d'erreur. Par défaut PHP 4 et 5 n'affichent pas les notices PHP lesquelles peuvent être important dans le déboguage de votre code (plus d'informations plus loin). Les notices sont produites par PHP, qu'elles soient affichées ou pas, ainsi déployer un code produisant vingt notices a un impact sur les fonctionnement général de votre site. Pour s'assurer que les notices sont visible, placer votre niveau de rapport d'erreurs dans votre le fichier php.ini ou modifier votre script pour ressembler à ceci :

```
<?php
ini_set('display_errors', 'On');
error_reporting(E_ALL);
```

E_ALL est une constante, ne pas faire l'erreur de l'inserer dans les quotes.

Avec PHP 5 c'est aussi une bonne idée de fixer le niveau de rapport d'erreurs à E_STRICT. E_STRICT est utile pour s'assurer que le code suit les bonnes pratiques. Par exemple E_STRICT avertie quand vous employez une fonction dépréciée. Voici comment l'activer durant l'exécution :

```
<?php
ini_set('display_errors', 'On');
error_reporting(E_ALL | E_STRICT);
```

Il est également intéressant de faire de même sur votre plateforme de développement, c'est parfois une bonne idée de faire ces changements dans le fichier php.ini plutôt qu'au moment de l'exécution. Car si vous rencontrez une erreur de syntaxe avec ces valeurs fixées dans le code et pas dans le php.ini vous pouvez, selon l'installation, avoir une page blanche. De même, il vaut la peine de noter que si vous placer ces valeurs dans le code, une expression conditionnelle pourrait être une bonne idée afin d'éviter d'avoir accidentellement ces valeurs sur l'environnement de production.

III - A quel type d'erreur ai-je affaire ?

Comme dans la plupart des langages, les erreurs PHP peuvent sembler quelque peu ésotériques, mais il y a en fait seulement quatre principaux types d'erreur à ne pas oublier :

III-A - Erreurs de Syntaxe

Les erreurs de syntaxes ou parse erreurs sont généralement provoquées par une faute de frappe dans le code. Par exemple un point-virgule, une quote, un crochet ou des parenthèses absentes. Quand vous rencontrez une erreur de syntaxe vous recevrez une erreur semblable à celle-ci :

```
Parse error: syntax error, unexpected T_ECHO in /Document/Root/example.php on line 6
```

Dans ce cas il est important de verifier la ligne au-dessus de la ligne citée dans l'erreur (dans ce cas-ci la ligne 5) parce que PHP rencontre quelque chose d'inattendu sur la ligne 6, c'est courant que la ligne fautive causant l'erreur soit celle au-dessus. Voici un exemple :

```
<?php
ini_set('display_errors', 'On');
error_reporting(E_ALL);

$sSiteName = "Developpez"
echo $sSiteName;
```

Dans cet example j'ai omis le point-virgule de la ligne 5, cependant, PHP a rapporté q'une erreur s'est produite sur la ligne 6. En regardant la ligne au-dessus on repère et corrige le problème.

Dans cet exemple j'utilise la notation hongroise. Adopter cette manière de codage peut faciliter le déboguage du code dans le cadre d'un travail en groupe ou sur un morceau de code écris il y a une époque. La première lettre identifie le type de la variable cela signifie que la détermination du type de la variable est très rapide et simple. Ceci peut faciliter à repèrer les irrégularités et aussi toutes les erreurs potentielles de logique.

III-B - Les Alertes

Les alertes ne sont pas des fauteurs de trouble comme les erreurs de syntaxe. PHP prencontre parfois des alertes, vous avez fait une erreur quelque part et t'informe à son sujet. Les alertes apparaissent souvent pour les raisons suivantes :

- les en-têtes sont déjà envoyés. Vérifier les espaces blanc au début du code ou dans les fichiers inclus.
- vous passer un nombre incorrect de paramètre à une fonction.
- un chemin incorrect dans une inclusion de fichier.

III-C - Les Notices

Les erreurs NOTICE ne stoppent pas l'exécution du script, mais elles peuvent être très importantes dans la recherche d'un bug embêtant. Il arrive de voir qu'un code qui fonctionnait parfaitement en production commencer à afficher des notices quand on places error_reporting à E_ALL.

Une notifice courante rencontrer pendant le développement est :

```
Notice: Undefined index: FullName in /Document/Root/views/userdetails.phtml on line 55
```

Cette information peut être extrêmement utile dans le débogage de votre application. Dire avoir fait une simple requête sur une base de données et avoir extrait d'une table une colonne des données d'utilisateurs. Pour la présentation dans votre vue vous avez assigné les détails dans un tableau appelé $aUserDetails. Cependant, l'affichage de $aUserDetails['FirstName'] sur la ligne 55 n'a aucune valeur et PHP génère une erreur NOTICE ci-dessus. Dans ce cas l'erreur NOTICE reçu aide réellement. PHP nous a utilement dit que que la clé FirstName n'existe pas. Nous savons que nous ne somme pas dans le cas d'un enregistrement NULL. Nous devrons peut-être vérifier notre résultat SQL pour nous assurer avoir réellement recherché le prénom de l'utilisateur dans la base de données. Dans ce cas-ci la notifice nous a aidé à éliminer un problème potentiel qui alternativement nous a orienté vers la source probable de notre problème. Sans l'erreur NOTICE, nous nous serions probablement arrêté sur l'enregistrement, suivi d'un retour en arrière suivant notre logique pour trouver par la suite notre omission dans le SQL.

III-C-1 - Les Erreurs fatales

Les erreurs fatales sont le type d'erreur le plus douloureux, mais souvent elles sont les plus faciles à résoudre. En resumé, cela signifie que PHP comprend ce que vous lui avez demandé de faire mais ne peut pas executer la demande. Votre syntaxe est correcte, vous parlez son language mais PHP n'a pas ce qui lui faut pour s'exécuter. L'erreur fatale la plus commune est une classe ou une fonction non définie et l'erreur produite pointe normalement directement à la source du problème :

```
Fatal error: Call to undefined function create() in /Document/Root/example.php on line 23
```

Utilisation de var_dump() pour faciliter le débogage. var_dump() est une fonction native de PHP qui affiche de manière structuré, humainement lisible, les informations sur un (ou plus) d'expression (s). C'est particulièrement utile au traitemet des tableaux et d'objets car var_dump() montre leur structure en te donnant la meilleure visibilité à l'instant. Voici un exemple d'usage de var_dump() dans ce contexte : j'ai créé un tableau de scores réalisée par des utilisateurs mais une valeur du tableau se distingue des autres, var_dump() peut nous aider à découvrir cette distinction.

```php
<?php
ini_set('display_errors', 'On');
error_reporting(E_ALL);

$aUserScores = array('Ben' => 7,'Linda' => 4,'Tony' => 5,'Alice' => '9');
echo '<pre>';
var_dump($aUserScores);
echo '</pre>';
```

Mettre var_dump() dans la balise <pre> facilite la lisibilité.

L'affichage du var_dump() ressemblera à ceci :

```
array(4) {
  ["Ben"]=>
  int(7)
  ["Linda"]=>
  int(4)
  ["Tony"]=>
  int(5)
  ["Alice"]=>
  string(1) "9"
}
```

Comme on peux le voir le var_dump nous indique que $aUserScores est un tableau avec quatre paires de clé/valeur. Ben, Linda, et tony ont tous ont leurs valeurs (ou scores) stockées comme entiers. Tandisque Alice s'affiche comme une chaîne de caractère suivi de la taille. Si nous revenons à mon code, on peut voir que j'ai tort d'inclure le score 9 de Alice dans les guillemets dans les guillemets, PHP l'interpréte comme une chaîne. Or, cette erreur n'aura pas un effet extrêmement négatif, cependant, elle démontre la puissance du var_dump() pour nous aider à avoir une meilleure

visibilité de nos tableaux et objets. Bien que c'est un exemple très simple sur le fonctionnement de var_dump(), il peut également être utilisé pour inspecter des tableaux ou objets à plusieurs dimensions. Il est particulièrement utile à découvrir si vous avez les bonnes données renvoyées par une requête de base de données ou lors de l'exploration d'une réponse JSON, voyons Twitter :

```
<?php
ini_set('display_errors', 'On');
error_reporting(E_ALL);

$sJsonUrl = 'http://search.twitter.com/trends.json';

$sJson = file_get_contents($sJsonUrl,0,NULL,NULL);
$oTrends = json_decode($sJson);

echo '<pre>';
var_dump($oTrends);
echo '</pre>';
```

IV - Outils utiles à prendre en compte lors du débogage

Enfin, je tiens à souligner quelques outils utiles que j'ai utilisés pour m'aider dans le processus de déboguage. Je ne vais pas entrer dans les détails sur l'installation et la configuration de ces extensions et plugins, mais je tenais à les mentionner car elles peuvent réellement rendre la vie plus facile.

IV-A - Xdebug

Xdebug est une extension PHP qui a pour objectif de prêter main-forte dans le processus de débogage de vos applications. Xdebug offre des fonctionnalités telles que

- Le stockage automatique des traces d'erreur dans une pile
- L'appel de fonction d'authentification
- L'affichage des caractéristiques telles que la production d'un var_dump et des certaines informations sur le code.

Xdebug est hautement configurable et adaptable selon les besoins. Par exemple, la pile des tracese (qui est extrêmement utile pour suivre ce que fait votre application et à quel moment elle le fait) peut être configuré à quatre différents niveaux de détail. Cela signifie que vous pouvez régler la sensibilité de la production de Xdebug pour vous aider à obtenir des informations granulaire sur l'activité de votre application.

Les traces de la pile vous montre où les erreurs se produisent, vous permettent de retracer les appels de fonction et de détailler les numéros de ligne en provenance de ces événements. Tout cela est fantastique pour l'information de débogage de votre code.

Par défaut Xdebug limites var_dump() de sortie à trois niveaux de récursivité. Il est possible de modifier cette valeur dans le fichier xdebug.ini en fixant la valuer de xdebug.var_display_max_depth à un nombre qui a va à vos besoins.

Pour commencer sur Xdebug, Consultez **le guide d'installation**.

IV-B - FirePHP

Nombreux sont fans de FireBug, FirePHP est une petite bibliotheque PHP utile associé à un plugin Firefox qui aide au développement AJAX. FirePHP permet d'enregistrer les informations de débogage à la console Firebug en utilisant simplement un appel de méthode comme ceci:

```php
<?php
$sSql = 'SELECT * FROM tbl';
FB::log('Requête SQL: ' . $sSql);
```

Dans le cas où je fais une requête AJAX de recherche, par exemple, il pourrait être utile de passer en argument de la chaîne SQL formée afin que je puisse m'assurer que mon code se comporte correctement. Toutes les données enregistrées dans la console Firebug est envoyé via des en-têtes de réponse et donc n'affecte pas la page qui est rendu par le navigateur.

Comme pour toutes les informations de débogage, ces donnes ne doivent pas être destinées à un usage publique. L'inconvénient d'avoir à ajouter les appels de la méthode de FirePHP dans votre code est qu'avant de mettre en production, vous aurez soit à supprimer l'ensemble de ces appels ou paramètre l'environnement de sorte à envoyer ses informations que sous certaines conditions.

Vous pouvez installer le plugin Firefox FirePHP depuis son **site Web** et aussi sur **grab the PHP libs**. Oh, et n'oubliez pas d'installer FireBug si vous ne l'avez pas déjà.

V - Conclusion

Espérons qu'au cours de cet article, vous avez appris comment bien préparer le terrain en préparant PHP au processus de débogage; reconnaître et traiter les quatre types clé d'erreurs PHP et utilisez la fonction var_dump () pour votre bien. De même, j'espère que vous trouverez FirePHP et Xdebug utiles et qu'ils vous rendront la vie plus facile lors de votre cycle de développement.

Comme je l'ai déjà mentionné, et je ne peux vraiment pas le dire assez, n'oubliez pas de retirer ou supprimer l'affichage des informations rélatives au débogage lorsque vous mettez vos sites en production, après tout il n'y a rien de pire que de donner aux utilisateurs la possibilité de lire vos erreurs dans les moindres détails.

VI - Voir aussi

- **Documentation PHP sur la configurations des errors reporting**
- **Cours de PHP 5 par Guillaume Rossolini**