Migration vers php 5.3

Quelques astuces pour automatiser la migration

par Maxime Varinard - Vaisonet (Vaisonet : Création de site e-commerce) (Blog)

Date de publication : 30 septembre 2009

Dernière mise à jour :

Voici quelques astuces permettant d'automatiser la migration vers php 5.3 à partir des versions précédentes, grâce à des expressions régulières.

1 - Présentation de l'auteur

Maxime Varinard est chef de projet chez Vaisonet ( ), société utilisant notamment php5 dans le développement d'application e-commerce. A l'occasion de la migration de notre application principale de php 5.2.9 à php 5.3, nous avons collecté dans cette article les astuces qui nous ont permis d'automatiser le plus possible cette migration.

2 - Introduction

L'objectif de cet article n'est pas de ré-écrire le guide de migration de php 5.2 à php 5.3 que l'on peut trouver ici :

Le but est d'en faciliter la mise en œuvre, en donnant les astuces permettant d'automatiser le plus possible la ré-écriture du code pour ne plus utiliser de fonctions dépréciées. Bien sûr, suivre à la lettre ces informations peut ne pas suffire à avoir une application 100% compatible php 5.3, mais le plus gros du travail sera fait.

3 - Préalable

Un IDE qui permet de rechercher et de remplacer des expressions régulières dans tous les fichiers d'un projet est nécessaire. NetBeans, Eclipse le font, mais aussi des éditeurs plus simples comme PsPad.

4 - Les fonctions à remplacer

call_user_method()

Info : Remplacer par call_user_func avec en tableau l'objet

call_user_method_array()

Info : Il faut ré-écrire proprement le code. Un foreach ne serait pas suffisant, la fonction call_user_method() étant déprécié. Il faudra utiliser désormais call_user_func.

define_syslog_variables()

Info : Suppression simple, les variables syslog n'ont plus besoin d'être initialisées.
Chaine de recherche : define_syslog_variables();
Chaine de remplacement : //define_syslog_variables()

ereg()

Info : Utiliser preg_match.
ereg("regex", devient preg_match("#regex#",
Précautions : penser à mettre les délimiteurs. Pas de remplacement automatique car il faut souvent corriger les expressions régulières (par exemple : "\." devient "."). Une relecture attentive s'impose.

ereg_replace()

Info : Utiliser preg_replace.
ereg_replace("regex", devient preg_replace("#regex#",
Précautions : penser à mettre les délimiteurs. Pas de remplacement automatique car il faut souvent corriger les expressions régulières (par exemple : "\." devient "."). Une relecture attentive s'impose.

eregi()

Info : Utiliser preg_match.
eregi("regex", devient preg_match("#regex#i",
Précautions : penser à mettre les délimiteurs et le i. Pas de remplacement automatique car il faut souvent corriger les expressions régulières (par exemple : "\." devient "."). Une relecture attentive s'impose.

eregi_replace()

Info : Utiliser preg_replace.
eregi_replace("regex", devient preg_replace("#regex#i",
Précautions : penser à mettre les délimiteurs et le i. Pas de remplacement automatique car il faut souvent corriger les expressions régulières (par exemple : "\." devient "."). Une relecture attentive s'impose.

set_magic_quotes_runtime()

Info : Pas de remplacement automatique possible. Il faut passer en revue ses requêtes et les échapper, par exemple avec mysqli_escape_string().

session_register()

Info : Utiliser $_SESSION['ma_variable'] = 'toto';
Chaine de recherche : [^_]session_register\((.*?)\)
Chaine de remplacement :\$_SESSION\[$1\]
Précautions :le [^_]est utile si vous avez des éléments type monframework_session

session_unregister()

Info : Utiliser unset($_SESSION['ma_variable'])
Chaine de recherche : [^_]session_unregister\((.*?)\)
Chaine de remplacement : unset\(\$_SESSION\[$1\]\)
Précautions :le [^_] est utile si vous avez des éléments type monframework_session

session_is_registered()

Info : Utiliser isset($_SESSION['ma_variable'])
Chaine de recherche : [^_]session_is_register\((.*?)\)
Chaine de remplacement : isset\(\$_SESSION\[$1\]\)
Précautions :le [^_]est utile si vous avez des éléments type monframework_session

set_socket_blocking()

Info : Fonction à remplacer
Chaine de recherche : set_socket_blocking(
Chaine de remplacement : stream_set_blocking(

split()

Info : Utiliser preg_split
split("regex", devient preg_split("#regex#",
Précautions : mettre les délimiteurs. Pas de remplacement automatique car il faut souvent corriger les expressions régulières (par exemple : "\." devient "."). Une relecture attentive s'impose. Explode() peut-être utile s'il n'y a pas besoin d'expression régulière.

spliti()

Info : Utiliser preg_split

spliti("regex", devient preg_split("#regex#i",
Précautions :mettre les délimiteurs et le i. Pas de remplacement automatique car il faut souvent corriger les expressions régulières (par exemple : "\." devient "."). Une relecture attentive s'impose. Explode() peut-être utile s'il n'y a pas besoin d'expression régulière.

sql_regcase()

Info : Devient inutile avec les expressions régulières POSIX.

dl()

Info : Charger les extensions via php.ini à la place.

extension mysql

Infos : Je conseille de passer à mysqli, qui donne de meilleures performances. Inconvénient, on a souvent mysql*(A,B) qui devient mysqli*(B,A). Une relecteure attentive du code est préférable à changement automatisé.

mysql_db_query()

Info : Fonction à remplacer par mysql_select_db() + mysql_query()
Chaine de recherche : set_socket_blocking(
Chaine de remplacement : stream_set_blocking(
Précautions : Un remplacement automatique se fera généralement entre mysql_db_query et mysql_query. mysql_select_db pouvant être placé après mysql_connect.

mysql_escape_string()

Info : Fonction à remplacer
Chaine de recherche : mysql_escape_string(
Chaine de remplacement : mysql_real_escape_string(