PHP, un langage très polyvalent - 1ère partie

- Les articles -

Date de mise en ligne : lundi 6 septembre 2010

Description :

On pense généralement à PHP comme à un langage dédié à la génération de pages webs dynamiques. Or PHP est beaucoup plus que cela : c'est un vrai langage de scripting, capable de piloter toutes sortes de processus batchs, des plus simples aux plus complexes.

Environnement iSeries

On pense généralement à PHP comme à un langage dédié à la génération de pages webs dynamiques. Or PHP est beaucoup plus que cela : c'est un vrai langage de scripting, capable de piloter toutes sortes de processus batchs, des plus simples aux plus complexes.

Dans le livre blanc que j'avais consacré à l'utilisation de PHP en environnement i5, j'avais surtout parlé de l'exécution de scripts PHP à l'intérieur d'un navigateur internet, mais il est important de savoir que ce n'est pas le seul moyen d'exécuter un script PHP.

Vous pouvez en effet exécuter vos scripts PHP en mode ligne de commande, sans passer par un navigateur. Vous obtenez ainsi de bien meilleures performances d'exécution. Vous pouvez également planifier l'exécution de vos scripts PHP de manière à ce qu'ils s'exécutent périodiquement. Pour ce faire, vous pouvez utiliser le système de planification de travaux de votre système, qu'il s'agisse d'un serveur Linux, Windows, ou d'un serveur i5.

J'ajoute que, si vous utilisez PHP en mode ligne de commande, vous n'avez aucun besoin d'installer Apache (ou tout autre serveur d'application) sur votre système. En effet, l'interpréteur PHP se suffit à lui même et n'a pas besoin d'un serveur d'application pour fonctionner. J'ajoute également que vous n'avez aucune obligation d'installer MySQL, si vous n'en avez pas réellement besoin. Par exemple, lors d'une mission chez un client, j'avais besoin d'exécuter sur un serveur Windows un programme dont l'unique fonction consistait à scruter le contenu d'un répertoire, et à récupérer des fichiers d'un certain type se trouvant dans ce répertoire, afin de les zipper et de les transférer dans un autre répertoire (qui était en fait un répertoire de l'IFS d'un serveur i5). J'avais écrit ce programme sous la forme d'un script PHP lancé plusieurs fois par heure via le planificateur de travaux du serveur Windows, et ceci avec d'excellentes performances. Seul PHP était installé sur ce serveur, il ne s'appuyait sur aucun serveur d'application.

A titre d'exemple, le code ci-dessous est une version remaniée et simplifiée du script que j'avais écrit pour mon client. Cette version est écrite dans un but purement pédagogique, mais elle est pleinement opérationnelle. Il s'agit bien ici de code PHP 5, même s'il est écrit dans un style de programmation procédural, et non pas objet. Vous pouvez l'adapter à vos besoins si vous le souhaitez.

```
<?php
 /**
 * Cette fonction extrait l'extension d'un fichier dont le nom a été
 * transmis en paramètre
 * @param $fichier
 * @return
 */
 function recupExtension($fichier) {
 // on sépare toutes les parties du nom du fichier en utilisant
 // le point comme séparateur et on stocke ces parties dans un tableau
 $decoupe = explode(".", $fichier) ;
 // l'extension est contenue dans le dernier poste du tableau, on la
 // stocke à part
 $extension  = array_pop($decoupe) ;
 // le nom du fichier est contenu dans les autres éléments du tableau
 // on les reconcatène
 $nom_fichier = implode(".", $decoupe);
 return array($nom_fichier, $extension) ;
```

```
}

/**
* Cette fonction scrute le contenu d'un répertoire et retourne un tableau
* contenant la liste de tous les fichiers ayant l'extension spécifiée
* dans le second paramètre d'entrée
* @param $repert
* @param $sel_ext
* @return
*/
function scrute_repertoire($repert, $sel_ext) {
try {
$files = array();
$dir = new DirectoryIterator($repert);

foreach ($dir as $entry) {
$filename = $entry->getFilename();
if ((!$entry->isDot()) &&
($filename != '~') &&
($filename != basename($_SERVER['PHP_SELF']))) {
if ($entry->getType() == 'file') {
list($nom_base, $extension) = recupExtension($filename);
} else {
$nom_base = '';
$extension = '';
}
// on ne traite que les fichiers dont l'extension
// correspond à celle reçue en paramètre
if (strtolower($extension) == strtolower($sel_ext)) {
$files[] = array(
"name" => $filename,
"size" => $entry->getSize(),
"type" => $entry->getType(),
"mtime" => $entry->getMtime(),
"path" => $entry->getPathname(),
"read" => $entry->isReadable(),
"write" => $entry->isWritable(),
"exec" => $entry->getType() == 'file' ? $entry->isExecutable() : '',
"basename" => $nom_base,
"ext" => $extension
);
}
}
}
// Release the resource.
unset($dir);
return $files;
} catch (Exception $e) {
return null;
}
}
```

```
/*
* forçage de la limite de temps, car la limite standard de 60
* secondes peut se révéler trop juste dans certains cas
*/
set_time_limit(600);

/**
* Analyse des paramètres en entrée pour en extraire les répertoires
* d'origine et de destination
* */
$repert_ori = null;
$repert_des = null;
if (isset($_GET["repert1"]) && isset($_GET["repert2"])) {
// transmission des paramètres via requête HTTP
$repert_ori = trim($_GET["repert1"]);
$repert_des = trim($_GET["repert2"]);
} elseif (isset($argv[1])) {
// transmission des paramètres en mode ligne de commande
$repert_ori = trim($argv[1]);
$repert_des = trim($argv[2]);
}
if ($repert_ori == null || $repert_ori == '' ||
$repert_des == null || $repert_des == '') {
die("chemins d'accès incorrects, arrêt du traitement");
}
if (!is_dir($repert_ori) || !is_dir($repert_des)) {
die("chemins d'accès incorrects, arrêt du traitement");
}

/*
* extraction de la liste des fichiers XML du répertoire spécifié
*/
$files = scrute_repertoire($repert_ori, 'xml');

/*
* déplacement des fichiers XML du répertoire d'origine vers le
* répertoire de destination
*/
foreach ($files as $file) {
$ancien = $repert_ori . '/' . $file['name'];
$nouveau = $repert_des . '/' . $file['name'];
if (copy($ancien, $nouveau)) {
unlink($ancien);
}
}
?>
```

Le script ci-dessus ne présentait pas de difficulté particulière. La version que j'ai utilisée en production était découpée un peu différemment, les fonctions se trouvant dans un autre script, et étant invoquées via la commande PHP "require_once", mais pour la simplicité de lecture de l'exemple ci-dessus, j'ai tout regroupé dans un seul script. Les 2 fonctions utilisées ici auraient pu être regroupées dans une classe abstraite, et invoquées comme des méthodes statiques, mais ce n'était pas indispensable, aussi je me suis limité volontairement à un style de programmation strictement procédural. J'y reviendrai certainement dans un prochain article.

Il y a un autre cas d'utilisation pour lequel je recours fréquemment à la ligne de commande PHP : c'est l'exécution de programmes de reprises de données. Dans l'exemple ci-dessous, j'extrais des données d'un fichier CSV pour les recopier dans une table MySQL ayant quasiment la même structure que le fichier d'origine. Le code est incomplet, par souci de simplification, j'ai omis de présenter le code d'ouverture de la connexion base de données (si vous avez lu mon livre blanc sur PHP, vous ne devriez avoir aucun mal à combler ce manque).

```
/*
 * augmentation du temps limite d'exécution
 */
 set_time_limit (1800) ;

 /*
 * vidage de la table MySQL de destination (reprisedata)
 */
 $nomtable = 'reprisedata' ;
 $st = $db->query('delete from '.$nomtable) ;

 /*
 * extraction des noms de colonnes de la table MySQL
 * qui seront utilisées pour préparer la requête d'insertion
 */
 $tab_nomcols = array() ;
 $st1 = $db->query('describe '.$nomtable) ;
 foreach ($st1->fetchAll(PDO::FETCH_ASSOC) as $row) {
 $tab_nomcols []= mb_strtolower($row['Field']) ;
 }

 /*
 * suppression des colonnes "cod_statut" et "cod_modif" qui n'existent
 *  pas dans le fichier CSV d'origine
 */
 $poste_supprime = array_pop($tab_nomcols);
 $poste_supprime = array_pop($tab_nomcols);

 /*
 * préparation des jokers utilisés dans la requête d'insertion
 */
 $jokers   = array() ;
 $nbcols = count($tab_nomcols) ;
 for ($i = 1; $i <= $nbcols; $i++) {
 $jokers[] = "?" ;
 }

 $nomcols_join = implode(",", $tab_nomcols) ;
 $jokers_join = implode(",", $jokers);

 // requetes d'insertion dans les tables
 $req1  = "INSERT INTO $nomtable ($nomcols_join) VALUES ($jokers_join)";

 $fichier = 'data.csv';
 $fichier_csv = dirname(__FILE__) . '/'.$fichier ;
 $file = fopen($fichier_csv, "r");
```

```
$inc_id = 0 ;
$nblig_traitees = 0 ;
$nblig_reprises = 0 ;

echo "reprise du fichier {$fichier} dans la table {$nomtable}". PHP_EOL;

try {
$st1 = $db->prepare($req1) ;

while (!feof($file) ) {
$nblig_traitees++;

$dtacsv = fgetcsv($file, 0, ';');
if (count($dtacsv) <= 1) {
echo "nombre de colonnes incorrect sur ligne $nblig_traitees". PHP_EOL ;
} else {
/*
* incrémentation de l'ID
*/
$inc_id++ ;
/*
* fusion de l'ID (calculé à la volée) et du tableau contenant
* les données provenant du fichier CSV
*/
$tableau = array_merge(array($inc_id), $dtacsv) ;
/*
* insertion de la ligne dans la table destinataire
*/
$st1->execute($tableau) ;

$nblig_reprises++;
}
}
} catch (PDOException $e) {
echo 'Error : ' . $e->getMessage() . '';
echo 'Code  : ' . $e->getCode() . '';
echo 'File  : ' . $e->getFile() . '';
echo 'Line  : ' . $e->getLine() . '';
echo 'Trace : ' . $e->getTraceAsString() . '';
}

fclose($file) ;
echo "nombre de lignes traitees : $nblig_traitees". PHP_EOL;
echo "nombre de lignes reprises : $nblig_reprises". PHP_EOL;
```

Le code ci-dessus ne présente pas de difficulté particulière, je me suis montré volontairement peu généreux au niveau des commentaires, pour vous obliger à l'étudier en profondeur (il faut bien que vous travailliez un peu ;).

Il y a bien évidemment plusieurs manières d'écrire un programme de reprise tel que celui ci-dessus. J'ai pu écrire le programme de reprise sous cette forme car le fichier CSV d'origine et la table MySQL de destination ont quasiment la même structure (les champs sont placés dans le même ordre). C'est bien évidemment une situation idéale qui ne se produit pas toujours. Je suis néanmoins assez content de ce programme, car j'ai pu le réutiliser à plusieurs

reprises, en le faisant évoluer au cas par cas en fonction des données que j'avais à reprendre.

Dans une seconde partie de cet article, je vous montrerai que PHP est un excellent outil pour l'administrateur de bases de données, qui a besoin de manipuler, comparer, et déployer des bases de données. Mais je ne vais pas trop en dire maintenant, je préfère ménager le suspense.

En attendant, si vous souhaitez approfondir l'utilisation du mode ligne de commande de PHP, je vous invite à lancer une recherche sur internet avec les mots clés suivants : "ligne de commande php". Vous trouverez de nombreux tutoriaux de qualité sur le sujet.

Si vous souhaitez en savoir davantage sur le pilotage - sur système Unix et Linux - de processus batchs écrits en PHP, je vous invite à lire l'excellent article de Jack Herrington : Batch processing in PHP