Chapitre 6 : Environnement d'exécution

Schéma d'exécution, fichier de configuration, include,
portée durée de vie

Schéma d'exécution

- L'interpréteur PHP exécute **un et un seul** fichier à la fois.
  - Le fichier peut être modifié par l'instruction **include**
- Les chemin relatifs (PATH) sont :
  - par rapport à la position sur le disque de cet unique fichier
- Gestion de codes de grande taille :
  - Regrouper des codes dans des fichiers qui sont inclus **au moment de l'exécution** (cf. instruction 'include' plus loin)
  - ATTENTION: dans les langages compilés comme ADA, C, C++, l'inclusion de fichier se fait à la compilation, donc **avant** l'exécution !
- Chemins :
  - **Jamais** de chemins absolus dans un projet
  - Utiliser des chemins relatifs !
  - Car ... plus facile de relocaliser le projet.
  - Tous les chemins relatifs le sont par rapport à la position absolue de l'unique fichier qui s'exécute.

Accès avec un chemin

```
// Dans le même répertoire que le fichier PHP
$f = fopen("mon_fichier1");

// Dans le répertoire frère le fichier PHP
$f = fopen("../data/mon_fichier2");
```

- Le chemin relatif de "mon_fichier" est celui du fichier qui est en cours d'exécution

Fichier de configuration

```
chemin = ./data
valeur_limite = 100
tab[] = 10
tab[] = 20
tab[animal] = chien du voisin
info = "définir la variable $chemin"
path = ${PATH}
```

- Format du fichier de configuration de php (php.ini)
  - pas besoin de $ pour les variables
  - les valeurs ne sont que des strings
  - délimiteurs optionnels
  - Pas d'interprétation des caractères spéciaux sauf ...
    - Pour l'accès aux variables d'environnement
    - commentaire : ';'

Fichier de configuration

```
$config = parse_ini_file("exemple_config.ini");

array(5) {
["chemin"]=> string(6) "./data"
["valeur_limite"]=> string(3) "100"
["tab"]=> array(3) {
        [0]=> string(2) "10" [1]=> string(2) "20"
        ["animal"]=> string(15) "chien du voisin" }
["info"]=> string(28) "définir la variable $chemin"
["path"]=> string(29) "/usr/bin:/bin:/usr/sbin:/sbin"
}
```

- La fonction d'analyse produit un tableau
- Méthode :
  - aucune constante importante dans un programme PHP
  - Utiliser un fichier de configuration

Inclusion de fichiers

Objectif : découper le code trop long, structuration de projet

`include()` ou `require()`

- inclusion dynamique lors de l'exécution
- **réévaluation** à chaque passage (exemple $var dans une boucle)
- inclusion d'un fichier PHP à l'endroit désiré

```
include('fichier.inc.php');
require('fichier.inc.php');
```

- `include()` : **warning** sur une erreur
- `require()` : **arrêt** sur une erreur
- `include_once()` ou `require_once()`
  - doivent être utilisées dans les cas où le même fichier risque d'être inclus et évalué plusieurs fois durant l'exécution d'un script.

  => sûr de n'être inclus qu'une fois

  ⇒ évite les problèmes de re-déclarations de fonctions ou autres.
- Conseil : utiliser **`require_once()`** dans 99% des cas

L'inclusion est une instruction (dynamique)

```
$A = 1;
$B = 10;
print($B);
if ($A) {
    include("B20.php");
} else {
    include("B50.php");
}
print($B);
```

- Le choix de l'inclusion se fait à l'exécution !
- Le fichier est passé à l'analyseur PHP
- Le code du fichier inclus hérite de la portée des variables de la ligne où l'inclusion apparaît

L'inclusion est une instruction (dynamique)

```
$A = 1;
$B = 10;
print($B);
if ($A) {
    include("B20.php");
} else {
    include("B50.php");
}
print($B);
```

Si B20.php :

<?php $B=20; ?>

Affiche : 10, puis 20

Portée des variables

```
1  $a = 10;
2  echo $a;
3  $b = 20;
4  echo $a + $b;
5  function carre($x) {return $x*$x;}
6  echo carre($a)
```

- Portée d'une variable :
  - La zone de code ou la variable est accessible (i.e. visible)
- En PHP : la portée va de l'initialisation de la variable ou déclaration du paramètre :
  - à la fin du fichier pour les variables globales
  - à la fin du bloc marqué par un '}' pour les variables locales et paramètres
- La portée de $b va de la ligne 3 à la ligne 6

Durée de vie des variables

```
$a = 10;
$b = 20;
function carre($a) {
  $b = $a*$a;
 return $b;
}
$a = carre(3);
```

- Durée de vie des variables globales : durée de l'exécution du **programme**.
- Durée de vies des variables locales et paramètres : durée de l'exécution de la **fonction**
- Exception : variables "static"
- Dans l'exemple : $b locale **masque** $b globale qui garde sa valeur (20)

Tester l'existence d'une variable

```
var_dump(isset($a));
$a = NULL;
var_dump(isset($a));
$a = 10;
var_dump(isset($a));
```

- La fonction 'isset()' est vrai si :
  - la variable existe
  - son contenu est différent de NULL
- Résultat : false / false / true
- NB: pour savoir si une constante est définie utiliser 'defined()'

Durée de vie des objets

```
$a = new Personne("Martin");
$b = $a;
// On change $b et cela modifie $a
$b->nom = "Dupont";
$a = NULL;
// Ici l'objet existe encore car référencé par $b
$b = NULL;
// Ici l'objet peut être récupéré par le ramasse miette
```

- Pas de "delete" explicite sur les objets
- Tant qu'une variable référence un objet, il existe (comme en Java) !
- Sinon : le "ramasse miette" (garbage collector), le fait disparaitre
- Basé sur le comptage de références

Comptage de références

```
$a = new Personne("Martin");
debug_zval_dump($a);
$b = $a;
debug_zval_dump($a);
debug_zval_dump($b);
$a = NULL;
// Ici l'objet existe encore car référencé par $b
debug_zval_dump($b);
$b = NULL;
// Ici l'objet peut être récupéré par le ramasse miette
```

- debug_zval_dump : affichage plus détaillé de l'interpréteur Zend
- Il y a une référence de l'objet dans $a, mais aussi une référence dans l'appel de la fonction debug_zval_dump() car la référence à l'objet est copiée lors de l'appel

Comptage de références

Affichage :

object(Personne)#1 (1) refcount(2) { ["nom"]=> string(6) "Martin" refcount(1) }

object(Personne)#1 (1) refcount(3) { ["nom"]=> string(6) "Martin" refcount(1) }

object(Personne)#1 (1) refcount(3) { ["nom"]=> string(6) "Martin" refcount(1) }

object(Personne)#1 (1) refcount(2) { ["nom"]=> string(6) "Martin" refcount(1) }

- C'est bien le même objet partout le #1
- Attention : une référence de plus car passage du paramètre par copie de la référence.
- Quand la référence tombe à zéro : la place de l'objet peut être récupérée par le ramasse miette (garbage collector)
- Pour info : les chaines sont aussi gérée avec des références ...

Résumé

- Un seul fichier PHP s'exécute
  - chemins relatifs par rapport à ce fichier
  - inclusion pour structurer son code
- Utiliser un fichier de configuration pour les constantes
- Portée et durée de vies des variables
  - idem les autres langages
- Durée de vie des objets
  - dépend des références comptabilisée
  - Pas de 'delete' : le ramasse miette gère la mémoire