Méthodes et techniques de développement

Olivier Dommange
2015

Sommaire

Développer avec PHP 1

Le mode de développement 1

La nomenclature (type de nommage) 2

Le patron de conception (« design pattern ») 4

L'organisation des fichiers 4

La librairie de code (« framework ») 5

Quelques rappels 5

Le processus d'échange d'informations avec le serveur.................................... 5

Les variables 6

Les constantes 8

Les tableaux 9

Les fonctions 11

Les classes 13

Les requêtes à la base de données en MySQLi 20

Connexion à la base de données 20

La sélection 22

Insertion, mise à jour et suppression 23

Une classe de gestion des requêtes 26

Le processus de traitement (1ère partie) 29

Les 6 étapes du processus de traitement 29

Le processus de traitement (2ième partie) : introduction au MVC...............35

Le contrôleur, le modèle et la vue (MVC) dans la pratique 36

L'URL rewriting 41

.htaccess 41

La gestion des urls avec PHP 42

La gestion de l'affichage 44

Le processus de transmission des données à la vue 44

Les espaces de nom 49

PHP est un langage serveur utilisé dans de nombreux outils tels que les CMS (content manager system – gestionnaire de contenus) comme Wordpress, Drupal ou Typo3 pour ne cités de ceux-ci. Près de 80% des sites Web utilisent ce langage (Wikipedia, http://fr.wikipedia.org/wiki/PHP – avril 2015). Les possibilités qu'offre ce langage aux développeurs en font une référence de choix dans le développement de sites et applications Web.

Ce document propose des méthodes et techniques de développement avec ce langage. Il a pour but de présenter des méthodes d'utilisation du langage dans le cadre de développements. Bien qu'un bref retour sur les principes de bases soit proposé, il est encouragé, voire très utile, de se familiariser avec les notions de base de la programmation (conditions, boucles et fonctions) et des outils serveurs (sessions et phpMyAdmin) de PHP pour mieux appréhender les outils et méthodes qui seront proposés.

Développer avec PHP

L'évolution qu'a connu le langage a permis d'introduire de nouvelles méthodes de programmation permettant ainsi au langage de gagner en maturité. Certains outils et méthodes sont toutefois devenus au fil des versions désuets et peu recommandables. Les outils et principes qui seront présentés sont ceux applicables depuis la version 5.4 du langage, introduite en 2012 (5.5 est apparue en 2013 et 5.6 en 2014). Utiliser ces nouveautés permet de développer des applications plus performantes et mieux sécurisées.

Le développement consiste à conjuguer l'utilisation d'outils aux méthodes de programmation du langage. Maîtriser les outils permet de rendre un code performant et rapide à exécuter. Qui dit méthode sous-entend également structure et organisation du code Certaines méthodes sont utiles et rendent le développement efficace par sa clarté et pour la facilité de son entretien. On y retrouve le mode de développement, la nomenclature (des variables, fonctions, objets ou fichiers), l'organisation des fichiers ou encore le processus de traitement des informations (appelé patron de conception ou « model pattern » en anglais).

Le mode de développement

Trois modes ou manières de concevoir du code sont répandues avec PHP. Il en existe d'autres mais il est déjà important de distinguer ceux-ci, puisque plus courants. Celui utilisé permettra de plus ou mois dissocier les opérations PHP et isolé le code HTML. L'objectif étant de faciliter l'utilisation, la maintenance et la pérennité du code. Certaines nécessitent toutefois une expérience et maîtrise du code et des concepts de développement.

Modes

Intégré

Comme son nom l'indique le code PHP est directement intégré au HTML. Les opérations sont rédigées au fur et à mesure des besoins. Le PHP colonise littéralement le code HTML.

Plus facile pour un débutant, il existe cependant peu d'avantages à ce mode de développement car le code est difficile à maintenir puisqu'il ne dispose d'aucune structure. Les mêmes opérations sont répétées à plusieurs reprises de pages en pages.

```
1  $nb = 6;
2  $result = $nb * 2;
3  echo '<span>' . $result . '</span>';
```

Procédural

Le mode procédural consiste à intégrer dans des fonctions des opérations récurrentes.

Ce mode permet de dissocier le code PHP du HTML et d'organiser les fichiers en fonction de leur utilité. Sont ainsi créées des fonctions utilitaires et spécifiques (format de dates, gestion d'e-mails, format de chaînes de caractères, requêtes à la base de données, création d'un fichier XML, ...).

```
1  $nb = 6;
2  $result = myFunction( $nb );
3  echo $result;
```

Orienté objet

Le mode orienté objet permet de créer des systèmes de gestion. Chaque système (classe) dispose de fonctions propres à un type d'application. La combinaison d'objets permet d'intégrer différents systèmes à priori indépendant les uns des autres.

Il se distingue par sa modularité, sa facilité d'utilisation, son entretien et la pérennité du code. Il s'avère cependant plus complexe à maîtriser.

```
1  $nb = 6;
2  $obj = new MyClass();
3  $result = $obj->myFunction( $nb );
4  echo $result;
```

La nomenclature (type de nommage)

Afin de faciliter la reconnaissance des variables, fonctions, objets, nom de fichiers, tables et champs de la base de données un type de nommage peut être définit. Les règles établies sont déclinées de ce que propose le langage. Il existe plusieurs méthodes de nomenclature. Voici quelques règles pouvant être intéressant d'appliquer.

Éléments

Variables

Inscrit en minuscule. Une majuscule est ajoutée au début de chaque mot afin de faciliter la lecture de la variable. Le nom d'une variable peut commencer par une majuscule pourvu qu'il s'agisse d'une correspondance avec le nom d'un autre élément facilitant son identification (nom du

```
1  // Standard
2  $nbArticles = 6;
3
4  // Correspond au nom du champ d'une
5     table de la base de données
6  $IdArticle = $_POST['IdArticle'];
```

	champ d'une base de données, par exemple).
	
	Un tiret bas simple (_) est employé devant une variable servant d'attribut privé ou protégé d'une classe.
		
		// Attribut public d'une classe
		public $article;
		
		// Attribut privé ou protégé d'une classe
		private $_article;

Constantes	Inscrit en majuscule. Un tiret bas simple (_) sépare les mots d'une même constante.	define('SITE_PATH', realpath(dirname(__FILE__))); include SITE_PATH . '/includes/Db.php';

Fonctions	Inscrit en minuscule. Une majuscule est ajoutée au début de chaque composant le nom de la fonction.
	
	Un tiret bas simple (_) est employé devant une fonction servant de méthode privée ou protégée d'une classe.
		// Standard
		function mailSend()
		{
		}
		// Méthode public d'une classe
		public function mailSend()
		{
		}
		// Méthode privée ou protégée d'une classe
		private function _mailSend()
		{
		}

Classes	Une majuscule en début de chaque mot composant le nom de la classe.	class Articles()
		{
		}

MySQL	Le langage MySQL inscrit dans le PHP est en majuscule.	$sql = 'SELECT * FROM articles';

Tables (BD)	Le nom d'une table est en minuscule et au pluriel. Une table associative s'inscrit au singulier séparé par un tiret bas simple (_).
	
	Un tiret bas simple (_) sépare les mots d'un même nom de table.
		// table standard
		$sql = 'SELECT * FROM articles';
		
		// table associative
		$sql = 'SELECT * FROM article_motcle';

Champ (BD)	Une majuscule est ajoutée au début de chaque mot composant le nom du champ. Un rappel du nom de la table au singulier est introduit.	$sql = 'SELECT IdArticle FROM articles';
Fichiers, répertoires et espaces de nom	Une majuscule est ajoutée au début de chaque nom de fichier et répertoire. Le nom de fichiers et de répertoires en minuscules sont tolérés.	namespace Application\Articles; include SITE_PATH . '/includes/Db.php';

Le patron de conception (« design pattern »)

Le patron de conception (ou « design pattern » en anglais) définit le processus de traitement des données et leur transmission à l'interface. Le patron de conception souvent employé en PHP est le MVC, pour Model, View et Controller. Il s'agit d'un concept répandu dans les librairies de code (« Framework », en anglais) PHP, tels que Zend, Symfony ou Laravel. Une rubrique entière est consacrée au patron de conception MVC dans ce document.

L'organisation des fichiers

Plusieurs méthodes peuvent s'appliquer en fonction du volume de l'application, du patron de conception, du mode de développement et/ou de l'éventuel « framework » utilisé.

Voici la proposition d'une organisation de fichier intégrant un développement orienté objet en MVC pour un petit projet.

Répertoires		
application		Contient les modules qui seront appelés tour à tour lors de la consultation des pages
	articles	Module spécifique
	contacts	Module spécifique
	menus	Module spécifique
css		
includes		Contient les outils de mise en route du système
	Db.php	Gère le chargement des pages en fonction de l'URL
	init.php	Initie l'assemblage des outils
	load.php	Gère le chargement des éléments dans la page.
view		Contenus visibles du site
	404.html	
	articles	Module spécifique
	contacts	Module spécifique
	menus	Module spécifique
.htaccess		Paramètre le fonctionnement du serveur. S'occupe du URL rewriting
index.php		

La librairie de code (« Framework »)

La maîtrise des méthodes de développement permet d'exploiter ou de concevoir des « Framework ». Ceux-ci proposent des solutions facilitant le développement et offrent l'avantage de permettre de gagner en efficacité, de structurer le développement, d'éviter de coder des opérations récurrentes tout en disposant d'un outil sécurisé et à jour.

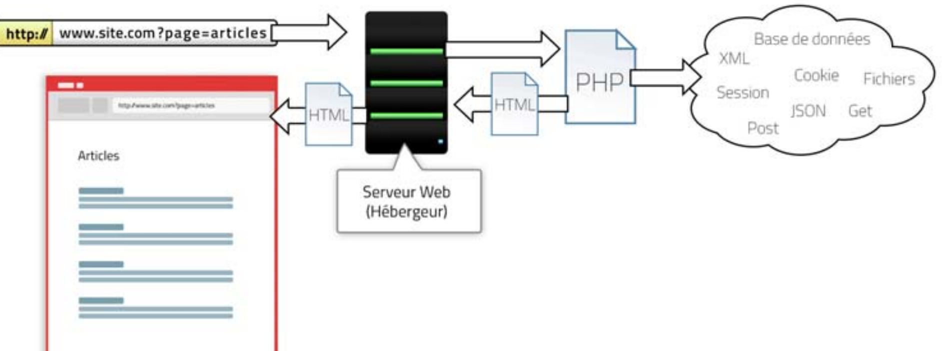
Il existe des dizaines de « Frameworks » PHP. Sont souvent cités comme références : Zend Framework, Symfony, Cake PHP, Yii, CodeIgniter, Laravel, ... Les principes et méthodes présentés dans ce document permettent d'aborder l'utilisation de ces outils.

Quelques rappels

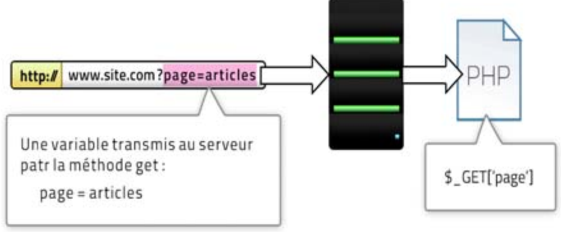
Petit retour sur quelques principes de base d'utilisation du langage. Ces principes et outils sont utiles et essentiels dans l'application des méthodes présentés dans ce document.

Le processus d'échange d'informations avec le langage serveur

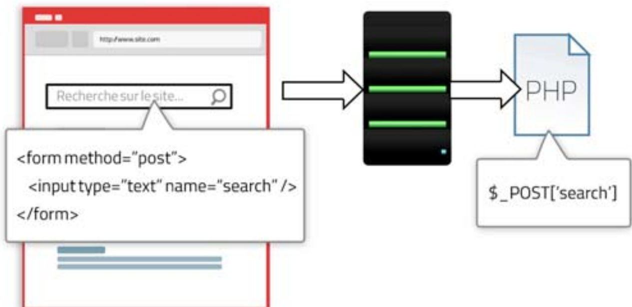
Grâce au protocole http un échange d'information se fait entre le client (navigateur) et le serveur. A la réception, PHP peut effectuer différentes opérations, telles que de collecter de données, avant de retourner une page HTML formatée (avec des contenus dynamiquement ajoutées).



Le transfert d'information entre le client et le serveur qui se fait par la méthode GET consiste à passer des informations dans l'adresse URL (variables et valeurs).



Le transfert d'information entre le client et le serveur qui se fait par la méthode POST est souvent utilisé dans des formulaires.



Les variables

Une variable est une sorte de contenant permettant le transfert de données et son altération au fil de déroulement des opérations. Une variable peut contenir différents types de contenus. La distinction des types est déterminant pour savoir employer une variable correctement.

La déclaration d'une variable

Les variables s'initient par le symbole dollars ($). Elles se composent de caractères alphanumériques. Seul le symbole tiret bas simple (_) peut être utilisé dans le nom qu'on lui donne.

Les variables locales

Elles ont une portée limitée, celle de la page ou d'une fonction.

```
1  $MyVar = 7;
2
3  function MyFunction()
4  {
5          $MyVar = 8;
6          echo $MyVar; // Affiche 8
7  }
   echo $MyVar; // Affiche 7
```

Les variables globales

Permet a une variable de transmettre sa valeur au-delà d'une fonction.

```
1  $MyVar = 7;
2
3  function MyFunction()
4  {
5          globale $MyVar ;
6          $MyVar = 8;
7  }
   echo $MyVar; // Affiche 8
```

Les variables super globales

Les variables super globales débutent par le symbole dollars et la ligne de base simple ($_) puis s'accompagnent du nom de la globale. Ces variables sont générées par PHP dans les cas suivants :

- lorsque des données sont transmises par les méthodes get $_GET ou post $_POST
- lorsqu'un fichier est transmis via un formulaire $_FILE
- pour obtenir des informations provenant du serveur $_SERVER
- pour initier ou récupérer des valeurs de session $_SESSION ou de cookie $_COOKIE.

Les valeurs des variables super globales sont inscrites dans un tableau de données. Raison pour laquelle les crochets ([]) accompagnent de la valeur et permettent d'indiquer le nom d'une clé. Les variables super globales ont une portée qui va au-delà de la fonction et sont accessibles à tout endroit dans le code.

```
1  echo $_SERVER['HTTP_HOST'] // Affiche le nom de domaine
```

Les types de contenus des variables

Bien que PHP ne soit pas strict dans la façon de déclarer une variable ou modifier sa valeur, une utilisation adéquate des variables passent par la compréhension des types de valeur.

<table>
<tr><th colspan="3">Types</th></tr>
<tr><td>Chaîne de caractères</td><td>La chaîne de caractère (« string » en anglais) dispose d'un contenu composé de caractères, chiffres et symboles. Les guillemets simples ou doubles peuvent être employés.</td><td><pre>1 $myVar1 = 'Ceci est du texte.';
2 $myVar2 = "Ceci est du texte.";
3 $myVar3 = 'L\'autre texte.';</pre></td></tr>
<tr><td>Valeur numérique</td><td>Une valeur numérique peut être intégrée à une opération mathématique.</td><td><pre>1 $myN1 = 5;
2 $myN2 = 10;
3 $result = $myN1 + $myN2;</pre></td></tr>
<tr><td>Tableau de données</td><td>Multiples valeurs inscrites dans un tableau. Chaque valeur dispose d'une clé permettant d'y faire référence.

Les variables globales ($_POST, $_GET, $_SESSION, $_FILE, $_COOKIE ou $_SERVER) générées par PHP entreposent leurs données sous la forme d'un tableau de données.</td><td><pre>1 $jours = array('lundi', 'mardi',
   'mercredi', 'jeudi', 'vendredi');
2 // Affiche mardi
3 echo $jours[1];
4
5 $pers = array('prenom'=>'Jonh',
   'nom'=>'Doe');
6 // Affiche John
7 echo $pers['prenom'];
8
9 // Alternative depuis 5.4
10 $pers = ['prenom'=>'Jonh',
11 'nom'=>'Doe'];
12 // Affiche John
13 echo $pers['prenom'];</pre></td></tr>
<tr><td>Objets</td><td>Le chargement d'un objet dans une variable est ce qu'on appelle l'instanciation d'une classe. Ceci permet, par l'appel de la variable déclarée, en</td><td><pre>1 $math = new Math();
2</pre></td></tr>
</table>

ajoutant les symboles trait d'union et plus petit que (->) d'accéder aux attributs et méthodes publiques de la classe.
Deux fois deux points (::) sont utilisés pour les attributs et méthodes publiques et statiques.

```
3  // Accès a un attribut
4  $math->num;
5
6  // Accès a une méthode
7  $math->result();
```

La concaténation

La concaténation est un processus par lequel les valeurs d'une ou plusieurs variables s'associent entre elles ou à une chaîne de caractères. En PHP, le symbole point (.) permet cette association.

```
1  $jours = 7;
2  echo 'Il y a ' . $jours . 'dans une semaine';
3
4  $pers = array( 'prenom'=>'Jonh', 'nom'=>'Doe' );
5  echo 'Votre nom est ' . $pers['prenom'] . ' ' . $pers['nom'] . '. Est-ce juste ?';
```

L'utilisation des guillemets doubles permet une interprétation des variables pendant la lecture. Ceci ralentit sensiblement le processus mais permet d'éviter la concaténation.

```
1  $jours = 7;
2  echo "Il y a $jours dans une semaine";
3
4  $pers = array( 'prenom'=>'Jonh', 'nom'=>'Doe' );
5  echo "Votre nom est $pers['prenom'] $pers['nom']. Est-ce juste ?";
```

Les constantes

Une constante permet de définir de manière définitive une valeur. L'utilisation d'une constante est similaire à celle d'une variable. Il est convenu d'utiliser les constantes pour des valeurs fixes. Comme l'adresse URL du site. La création d'une constante se fait par l'appel de la fonction PHP define().

```
1  define ('SITE_PATH', realpath( dirname(__FILE__) ) );
2  define ('SITE_URL', str_replace(realpath( $_SERVER['DOCUMENT_ROOT'] ), '', SITE_PATH) );
3
4  include SITE_PATH . '/includes/init.php';
```

Les tableaux de données (array)

Les tableaux de données permettent de cumuler plusieurs valeurs dans un seul ensemble. Chaque valeur dispose d'une clé de référence ce qui facilite leur accès. Les clés de références s'initient par le chiffre 0 et s'incrémente automatiquement (lorsqu'elles ne sont pas déclarées).

```php
$jours = array( 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi' );
/*
Le tableau $jours est conçu comme ceci
|-cle-|--valeur--|
|  0  | lundi    |
|  1  | mardi    |
|  2  | mercredi |
|  3  | jeudi    |
|  4  | vendredi |
*/

// Affiche mardi
echo $jours[1];
```

La clé de référence peut être également déclarée et nommée.

```php
$pers = array( 'prenom'=>'Jonh', 'nom'=>'Doe' );
/*
Le tableau $jours est conçu comme ceci
|---cle---|-valeur-|
| prenom  | John   |
| nom     | Doe    |
*/

// Affiche John
echo $pers['prenom'];
```

Depuis la version 5.4 de PHP un tableau de données peut également s'initier en remplaçant la fonction array() par des crochets ([ ]).

```php
$pers = [ 'prenom'=>'Jonh', 'nom'=>'Doe' ];
```

Un tableau peut disposer de plusieurs niveaux. En d'autres termes, un tableau peut en contenir d'autres.

```
$pers = array(
        array( 'prenom'=>'Jonh', 'nom'=>'Doe' ),
        array( 'prenom'=>'Jane', 'nom'=>'Dew' )
);
/*
Le tableau $jours est conçu comme ceci
|-cle-|----------valeur--------|
|  0  |  |---cle---|-valeur-|  |
|     |  | prenom  | John   |  |
|     |  | nom     | Doe    |  |
|  1  |  |---cle---|-valeur-|  |
|     |  | prenom  | Jane   |  |
|     |  | nom     | Dew    |  |
*/

// Affiche Jane
echo $pers[1]['prenom'];
```

Les tableaux de données et les boucles

L'exploitation d'un tableau de données peut se faire avec les boucles « for », « while » et « foreach ». La boucle « foreach » est très souvent la plus indiquée car elle permet d'extraire également la clé de référence.

```
$pers = array( 'prenom'=>'Jonh', 'nom'=>'Doe' );

foreach( $pers as $cle => $per )
{
        echo $cle; // Affiche prenom, puis nom
        echo $per; // Affiche Jonh, puis Doe
}
```

Les variables super globales sont des tableaux de données

Les variables super globales telles que $_POST, $_GET, $_FILE, $_COOKIE, $_SESSION, $_SERVER sont inscrites dans un tableau dont les clés de références sont prédéfinies. Considérant ceci l'exploitation des valeurs qu'elles contiennent se fait comme expliqué précédemment.

Astuce : Afficher tout ce que contient un tableau de données

Il n'est pas rare que l'on souhaite consulter un tableau pour connaître sa structure et les valeurs qu'il dispose. Deux fonction PHP peuvent être très utiles dans ce cas. Il s'agit de print_r() et var_dump() qui affichent dans le navigateur le contenu du tableau de données inscrit dans une variable. La fonction var_dump() exploite n'importe quel type de contenu d'une variable et donne des informations complémentaires telles que son type. Elle est souvent préférée pour le débogage pour ces raisons.

```
1  $pers = array( 'prenom'=>'Jonh', 'nom'=>'Doe' );
2
3  print_r( $pers );
4  // ou
5  var_dump( $pers );
6
7  // Cette forme donnera un aperçu beaucoup plus lisible des contenus dans le navigateur.
8  ?><pre><?php print_r( $pers ); ?></pre><?php
9  // ou
10 ?><pre><?php var_dump( $pers ); ?></pre><?php
```

Les fonctions

PHP met à disposition une panoplie de fonctions prête à l'emploi. Celles-ci sont utiles pour des opérations communes. A titre d'exemple, la fonction str_replace() permet de remplacer des caractères par d'autres dans une chaîne, la fonction count() indique le nombre d'éléments disposés dans un tableau de données ou encore la fonction ceil() arrondi à l'entier le plus élevé un nombre décimal.

Il est également possible d'ajouter à cette librairie des fonctions conçues sur mesure. Une fonction permet de grouper une série d'opérations qui ne seront déclenchées que lorsque la fonction sera appelée. Elle se conçoit comme suit :

```
1  function checkMessageSent()
2  {
3      if( isset( $_POST[ 'email' ] ) )
4      {
5          $datas = $_POST;
6          mail( 'mymail@domain.net', 'Subject', 'Message', 'From:'.$datas[ 'email' ] );
7      }
8  }
9
10 // Exécute la fonction (et les opérations qu'elle contient)
11 checkMessageSent();
```

Des valeurs peuvent être transmises lors de l'appel de la fonction. Ceci permet d'altérer le fonctionnement d'une fonction et de la rendre plus modulable. Ces valeurs passent par des paramètres prédéfinis. Les paramètres ont la forme d'une variable.

```
1  function somme( $param1, $param2 )
2  {
3      echo $param1 + $param2;
4  }
5  // Affichera 9
6  somme( 4, 5 );
7  // Affichera 12
8  somme( 6, 6 );
```

Le renseignement des paramètres est obligatoire du moment où ils ont été déclarés. Il est toutefois possible d'attribuer des valeurs par défaut aux paramètres ce qui rend accessoire leur renseignement.

```
1  function somme( $param1, $param2 = 10 )
2  {
3      echo $param1 + $param2;
4  }
5  // Affichera 14
6  somme( 4 );
7  // Affichera 12
8  somme( 6, 6 );
```

Les fonctions peuvent également retourner un résultat. Le type de format des valeurs retournées sont les mêmes que pour les variables (chaîne de caractères, valeur numérique, tableau de données ou objet).

```
1  function somme( $param1, $param2 = 10 )
2  {
3      return $param1 + $param2;
4  }
5  // Affichera 14
6  echo somme( 4 );
7  // Affichera 12
8  echo somme( 6, 6 );
```

Depuis la version 5.3 de PHP, il est possible d'utiliser des fonctions anonymes. Cette nouvelle forme élargit les possibilités d'utilisation.

```
1  $somme = function somme( $param1, $param2 = 10 )
2  {
3      return $param1 + $param2;
4  }
5
6  // Affichera 14
7  echo $somme( 4 );
```

Les classes

Les classes sont les éléments utilisés dans la programmation orienté en objet (POO). Le concept et la manière de travailler avec les classes demande un effort particulier pour celui s'en est jamais servi. Pourtant, les classes se composent d'outils connus : de variables et de fonctions. Elles font respectivement références aux attributs (variables) et aux méthodes (fonctions) d'une classe.

A titre d'exemple, une classe peut servir à gérer un système de fichiers. Pour chaque fichier, sont conservées leurs caractéristiques dans une version spécifique de la classe qu'on appelle un objet. Ainsi chaque fichier devient un objet. La classe prévoit une série d'attributs (variables) et de méthodes (fonctions) qui se rapportent aux informations et manipulations possibles avec un fichier. Les attributs pouvant se rapporter à des informations telles que le nom, le poids, le format ou encore la taille (largeur et hauteur) dans le cas d'une image. Les méthodes permettent elles de dupliquer le fichier, d'archiver le fichier dans un répertoire ou encore de modifier son contenu.

Ce qui peut s'avérer complexe à démystifier sont les termes de l'orienté objet. Voici un petit lexique et quelques exemples à l'appui.

Termes

Classe

Elle est le système complet disposant des informations et manipulations prévues.

```
1 class MyFileClass
2 {
3         // Attributs
4
5         // Méthodes
6 }
```

Objet

A ne pas confondre avec la classe. Pourtant ils sont intimement liés. Un objet est en fait défini pour chaque élément utilisant une classe. Lorsque qu'un objet est créé, il s'agit d'une instance de la classe.

```
1 // $myFile1 est un objet de MyFileClass
2 $myFile1 = new MyFileClass('img.jpg');
3 // $myFile2 est autre un objet de MyFileClass
4 $myFile2 = new MyFileClass('file.pdf');
5
6 // Accède à l'attribut concernant le poids du
     fichier instancié avec $myFile1
7 echo $myFile1->fileWeight;
```

Instance

Instance et objet font référence à la même chose. Il s'agit de synonymes. On parle d'instance d'une classe lorsqu'un objet est créé.

Attribut (ou Propriété)

Il s'agit de variables disposés dans la classe qui recensent des informations.

```
1 class MyFileClass
2 {
3         public $fileWeight;
4         public $fileFormat;
5         public $filePath;
6
```

		`7 // Méthodes` `}`
Méthode	Il s'agit de fonctions qui permettent d'effectuer des opérations spécifiques.	(code below)

```
1  class MyFileClass
2  {
3          // Attributs
4
5          public function fileChangeFormat()
6          {
7                  // Opérations
8          }
9  }
```

L'appel des attributs et des méthodes

Un attribut ou une méthode ne seront pas exactement appelés de la même façon par un objet ou une classe. L'appel par un objet se fait via la variable qui a servi à créer l'instance. C'est cette variable qui fait appel directement aux attributs et méthodes de la classe.

```
1  // $myFile1 est un objet de MyFileClass
2  $myFile1 = new MyFileClass('img.jpg');
3
4  // Accède à un attribut de la classe MyFileClass
5  $myFile1->fileWeight;
6
7  // Accède à une méthode de la classe MyFileClass
8  $myFile1->fileChangeFormat();
```

L'appel à l'intérieur d'une classe d'un attribut ou d'une méthode se fait avec la variable $this qui fait allusion à la classe même.

```
1  class MyFileClass
2  {
3          public $fileWeight;
4
5          public function getFileWeight()
6          {
7                  return $this->fileWeight;     // Accède à un attribut de la classe
8          }
9
10         public function otherMethod()
11         {
12                 $this->getFileWeight();       // Accède à une méthode de la classe
13         }
14 }
```

L'état statique

L'état statique permet de disposer des attributs et méthodes d'une classe sans créer d'instance. Ce type d'appel est moins fréquent mais s'avère utile dans le cas où il s'agit de consulter une information qui ne change pas. Un exemple concret est la connexion à la base de données. Une fois la connexion établie elle ne change pas. Nous avons toutefois besoin de l'état de la connexion à chaque fois qu'une nouvelle requête doit être initiée.

Un attribut ou une méthode est considéré comme statique dès le moment où le mot-clé « static » accompagne leur déclaration.

```
1  class Db
2  {
3      // Attribut statique de la classe
4      public static $port = 3360;
5
6
7      // Méthode statique de la classe
8      public static function db(){
9          return new mysqli( 'localhost', 'root', '', 'atelierphp', self::$port );
10     }
11 }
```

L'appel des attributs et méthodes statiques

Comme précédemment une distinction est faite du moment où l'appel se fait à l'extérieur ou à l'intérieur d'une classe.

Comme indiqué précédemment aucune instanciation n'est nécessaire pour appeler un attribut ou une méthode à l'extérieur de la classe. Il suffit d'appeler la classe accompagnée de la méthode séparés par deux fois deux points (::).

```
1  $db = Db::db();
2
3  $results = $db->query( 'SELECT * FROM articles' );
```

A l'intérieur d'une classe, le mot-clé self:: permet d'accéder aux attributs et méthodes de la classe.

```
1  class Db
2  {
3      // Attribut statique de la classe
```

```
4       public static $port = 3360;
5
6       // Méthode statique de la classe
7       public static function db(){
8           return new mysqli( 'localhost', 'root', '', 'atelierphp', self::$port );
9       }
10  }
```

Le constructeur

Le constructeur est une méthode (fonction) qui à pour rôle d'initier des opérations dès que la classe est instanciée (lors de la déclaration d'un objet). Le constructeur n'est pas indispensable à une classe.

La déclaration d'une méthode constructeur se fait par la déclaration dans la classe de la méthode __construct().

```
1   class MyClass
2   {
3           function __construct(){
4                   // Opérations prévues à l'appel de la classe MyClass
5           }
6   }
```

Il peut notamment s'avérer utile de transmettre certaines valeurs lors de la création d'un objet. Ceci facilite la transmission de paramètres essentiels pour la suite. Les paramètres ont la forme d'une variable et intègrent la classe via le constructeur.

```
1   class MyOtherClass
2   {
3           function MyOtherClass( $param ){
4                   // Opérations prévues à l'appel de la classe MyClass
5           }
6   }
7
8   $myObj = new MyOtherClass( 'value' );
```

L'héritage

Une classe peut hériter d'une autre... ce qui signifie qu'il est possible d'utiliser des attributs et méthodes dans une classe sans créer d'instance alors qu'ils proviennent d'une autre classe.

Prenons l'exemple d'une classe qui servirait à gérer l'opération de chargement (importation) de fichiers. Cette classe disposerait des attributs et méthodes pouvant servir à l'opération. La classe de gestion de fichiers précédemment créée (MyFileClass) pourrait servir à cette classe puisqu'il sera sans doute utile d'obtenir des informations telles que le nom, le poids ou le format du fichier en plus d'utiliser les méthodes de transformation déjà prévues. Ainsi, la classe servant au chargement du fichier héritera des fonctionnalité de la classe MyFileClass. L'héritage s'établit pas l'appel de la classe lors de la déclaration de la nouvelle classe par le mot-clé « extends ».

```
1   class MyUploadFile extends MyFileClass
2   {
3           // Attributs de la classe
4
5           // Méthodes de la classe
6   }
7
8   $myFile1 = new MyUploadFile('img.jpg');
9
10  // Accède à l'attribut fileWeight provenant de la classe MyFileClass
11  echo $myFile1->fileWeight;
```

La portée (visibilité) des attributs avec les modes (public, private et protected)

Les attributs et méthodes peuvent être appelés à l'intérieur comme à l'extérieur d'une classe. Ils peuvent de plus être utilisés par une autre classe grâce à l'héritage. La déclaration d'un attribut ou d'une méthode s'accompagne d'un indicateur quant à sa portée définit par les mots-clés « public », « protected » ou « private ». La portée « public » rend un attribut ou une méthode accessible dans tous les cas de figure. Cela s'avère cependant peut prudent, voire peu rigoureux. Certains attributs et méthodes peuvent ne pas servir à l'extérieur d'une classe. Il n'est pas nécessaire de les rendre inutilement accessibles en permanence.

Voici une représentation des portées « public », « protected » et « private ».

	Public	Protected	Private
A la classe (interne)	✔	✔	✔
Par héritage	✔	✔	○
Par instanciation (externe)	✔	○	○

Sur la base du précédent tableau voici quelques exemples d'utilisation. Il est d'usage d'ajouter le tiret bas simple (_) devant les noms des attributs et méthodes protégés « protected » et pivés « private ».

```
1   class MyClass
2   {
3           public    $propPublic;        // Pourra être utilisé dans tous les cas de figure
4           protected $_propProtected;    // Pourra être utilisé dans la classe courante et
                                             la/les classe/s enfant/s
5           private  $_propPrivate;        // Pourra être uniquement utilisé dans la classe
                                              courante
6
7           function __construct(){
8                   echo $this->propPublic;       // OK
9                   echo $this->_propProtected;   // OK
10                  echo $this->_propPrivate;     // OK
11          }
12  }
13
14  class MyOtherClass extends MyClass
15  {
16          function __construct(){
17                  echo $this->propPublic;       // OK
18                  echo $this->_propProtected;   // OK
19                  echo $this->_propPrivate;     // Erreur
20          }
21  }
22
23  $myObj = new MyOtherClass();
24
25  echo $myObj->propPublic;      // OK
26  echo $myObj->_propProtected;  // Erreur
27  echo $myObj->_propPrivate;    // Erreur
```

Recommandations pratiques de la programmation orientée objet

Voici quelques recommandations qui figurent comme des bonnes pratiques dans la programmation orientée objet.

Limitation de l'accès des attributs

Les attributs sont des informations considérées comme étant uniquement accessibles et utiles par les méthodes de la classe dans laquelle ils sont déclarés. Il est par conséquent recommandé de définir la portée des variables en « private » uniquement.

Cette pratique à l'avantage de faciliter la maintenance de la classe et limiter ses dépendances en plus de limiter le renseigner des attributs d'une autre classe accessible par héritage.

Pour accéder aux valeurs des attributs privés depuis un objet, il existe les « getter » et « setter ». Il s'agit en fait de méthodes qui ont pour but de renseigner ou de récupérer la valeur d'un attribut.

```
1   class MyClass
2   {
3           private  $_prop1;      // Attribut privé
4
5           public function setProp1( $valueToSet )
6           {
7                   $this->_prop1 = $valueToSet;
8           }
9
10          public function getProp1()
11          {
12                  return $this->_prop1;
13          }
14  }
15
16  $myObj = new MyClass();
17
18  $myObj->setProp1( 'John Doe' );
19
20  // Affichera John Doe
21  echo $myObj->getProp1();
```

Offrir un retour à toutes les méthodes publiques

Lorsqu'une méthode est appelée depuis l'instance d'un objet, il est pratique d'avoir à disposition un retour (« return ») que ce soit pour transmettre des données ou établir un verdict sur le succès (true) ou échec (false) des opérations.

```
1   class MyClass
2   {
3           public function myMethod1()
4           {
5                   // [...] série d'opérations préalablement
6                   return ( isset( $process ) ) ? true : false;
7           }
8
9
10          public function myMethod2()
11          {
12                  // [...] récupération de données
13                  return ( count( $datas ) > 0 ) ? $datas : null;
14          }
15  }
```

Entretenir les classes génériques au fil des projets

L'un des objectifs de la programmation orientée objet est de limiter le temps de développement. Une classe utile dans un projet le sera sans aucun doute dans un autre. Le but est d'utiliser les classes d'un projet pour un autre et assurer une mise à jour de celles-ci en fonction des évolutions des technologies et des techniques du langage.

Aller plus loin...

Les constantes, les méthodes magiques, l'abstraction des classes, l'autochargement de classes, les interfaces, pour ne nommer qu'eux, sont des outils utiles à la programmation orientée objet en PHP qui permettent d'aller plus loin dans l'utilisation des possibilités du code.

Il peut être intéressant de consulter la rubrique réservée à la POO du site de php.net (http://php.net/manual/fr/language.oop5.php) ou encore de suivre le cours proposé par le site OpenClassRooms sur le sujet (http://openclassrooms.com/courses/programmez-en-oriente-objet-en-php).

Les requêtes à la base de données en MySQLi

Depuis la version PHP 5.5.0 les fonctions MySQL sont considérées comme désuètes et ne seront plus supportées dans un futur proche. Ce sont les extensions PDO (PHP Data Object) ou MySQLi qui les remplacent. MySQLi offre une approche intéressante qui transite entre MySQL et PDO. Il existe à peu de choses près les mêmes noms de fonctions que dispose MySQL mais elles s'emploient légèrement différemment. MySQLi peut s'utiliser en mode procédural (appel de fonctions) mais aussi en orienté objet. Quoique quelques explications seront données sur le mode procédural, le second mode est privilégié dans cette documentation.

Connexion à la base de données

L'opération de la connexion à la base de donnée doit dorénavant être associée à chaque requête. Ainsi la méthode query() (ou fonction mysqli_query() en mode procédural) fera toujours appel à la connexion à la base de données. Ceci est la principale différence avec l'extension MySQL.

Isoler les configurations de connexion

Afin de faciliter la configuration d'un projet l'isolement des paramètres utiles à la connexion peut s'avérer pratique. Un fichier « config.ini » situé à un endroit stratégique dans l'architecture des fichiers créés.

```
1  [database]
2  dbhost = localhost
3  dbuser = root
4  dbpass =
5  dbname = atelierphp
6  dbport = 3306
```

Connexion en mode procédural

La différence importante avec le procédé connu de MySQL se situe dans la connexion de la base de données et l'utilisation de cette connexion dans les requêtes.

```
1  function dbConnect()
2  {
3      static $connect; // static permet à la variable de conserver la valeur de conn.
4
5      if( !isset( $connect ) )
6      {
7          $config = parse_ini_file( SITE_PATH . '/includes/config.ini' );
8
9          $connect = mysqli_connect( $config['dbhost'], $config['dbuser'],
                                      $config['dbpass'], $config['dbname'],
                                      $config['dbport'] );
10     }
11
12     if( !$connect )
13     {
14           return mysqli_connect_error();
15     }
16
17     return $connect;
18 }
```

De cette façon, l'appel à la connexion dans la fonction mysqli_query() se fera par l'appel de la fonction dbConnect() qui vient d'être créé.

```
1  $connect = dbConnect();
2
3  $query = 'SELECT * FROM articles';
4
5  $results = mysqli_query( $connect, $query );
```

Connexion en mode orienté objet

En mode orienté objet cela signifie qu'un objet est créé de la classe MySQLi. Les requêtes seront effectuées depuis cet objet.

Pour assurer un accès permanent à la connexion de la base de données quelque soit l'endroit dans le code où une requête se déclare une classe peut être créée. Puisque pour cette opération est commune et fixe pour toutes les requêtes et que la connexion n'a qu'à se faire qu'une seule fois (et non pas à chaque fois qu'une requête à lieu) la classe utilisera une méthode statique pour la connexion.

```
Class Db{

    private static $_connect;

    public static function connect()
    {
        if( !isset( self::$_connect ) )
        {
            $config = parse_ini_file( SITE_PATH . '/includes/config.ini' );

            self::$_connect = new mysqli( $config['dbhost'], $config['dbuser'],
                                          $config['dbpass'], $config['dbname'],
                                          $config['dbport'] );
        }

        return self::$_connect;
    }
}
```

Ainsi lors de la création d'une requête la transmission de la connexion à la base de donnée par l'appel de la méthode statique suffira. Puisqu'il s'agit d'une classe l'appel pourra se faire dans une fonction comme dans n'importe quelle classe.

```
$db = Db::connect();

$results = $db->query( 'SELECT * FROM articles' );
```

La sélection

L'opération de sélection de données dans la base de données s'effectue sur la base d'une requête MySQL « select ». MySQLi retourne le résultat à travers l'objet initié à la connexion. Le résultat dispose d'attributs et de méthodes qui permettent d'exploiter les données qui ont été repêchées dans la base de données ou d'afficher le nombre de résultats trouvés, par exemple.

```
1   $db = Db::db();
2
3   $results = $db->query( 'SELECT * FROM articles' );
4
5   if( !$db->errno && $results->num_rows > 0 )
6   {
7       if( isset( $results ) )
8       {
9           while( $row = $results->fetch_array() )
10          {
11          ?>
12              <article>
13                  <h2><?php echo $row[ 'TitleArticle' ]; ?></h2>
14              </article>
15          <?php
16          }
17      }
18  }
```

Insertion, mise à jour et suppression

Les procédés d'insertion, de mise à jour et de suppression se font respectivement par les requêtes MySQL « insert », « update » et « delete ». Les requêtes peuvent s'établir avec l'appel de l'objet MySQLi ayant été initié à la connexion à la base de données. Exemple avec la suppression.

```
1   $db = Db::db();
2
3   $db->query( 'DELETE FROM articles WHERE IdArticle = \''.$id.'\'');
```

Vérification des données

Une attention particulière est à apporter à la sécurité lorsqu'il est question de composer une requête car elle est susceptible d'être la proie à une injection SQL qui consiste à dissimuler une requête SQL dans une autre par l'entremise d'une variable transmise par la méthode get ou post. Pour cette raison, le processus de traitement doit inclure une vérification des données qui seront introduites dans les requêtes. Deux méthodes peuvent être utilisées dans la vérification des données.

1. La méthode real_escape_string()

Cette méthode (mysqli_real_escape_string() en mode procédural) vérifie que les valeurs des variables ne disposent pas de requêtes cachées en échappant notamment les guillemets des chaînes de caractères.

Voici un exemple de l'utilisation de la méthode real_escape_string() avec l'insertion de données.

```
1  $db = Db::db();
2
3  $TitleArticle   = $db->real_escape_string( $_POST['TitleArticle'] );
4  $IntroArticle   = $db->real_escape_string( $_POST['IntroArticle'] );
5  $ContentArticle = $db->real_escape_string( $_POST['ContentArticle'] );
6
7  $db->query( 'INSERT INTO articles( TitleArticle, IntroArticle, ContentArticle )
   VALUES ( \''.$TitleArticle.'\', \''.$IntroArticle.'\', \''.$ContentArticle.'\' )');
```

Un second exemple, cette fois pour la mise à jour de données.

```
1  $db = Db::db();
2
3  $TitleArticle   = $db->real_escape_string( $_POST['TitleArticle'] );
4  $IntroArticle   = $db->real_escape_string( $_POST['IntroArticle'] );
5  $ContentArticle = $db->real_escape_string( $_POST['ContentArticle'] );
6
7  $db->query( 'UPDATE articles SET
                TitleArticle = \''.$TitleArticle.'\',
                IntroArticle = \''.$IntroArticle.'\',
                ContentArticle = \''.$ContentArticle.'\'
                WHERE IdArticle = \''.$id.'\''
   );
```

2. Les requêtes préparées

Cette seconde méthode va un peu plus loin et offre certains avantages. Elle s'apparente à la manière dont fonctionne l'extension PDO. Une requête préparée permet de vérifier les données et d'exécuter la requête dans le même mouvement. Il s'agit d'une autre classe MySQLi qui porte le nom de mysqli_stmt. Cela peut s'avérer pratique lorsque la requête doit être exécutée plus d'une fois et avec différentes valeurs.

La préparation de la requête consiste à définir un modèle qui permettra ensuite d'assigner des valeurs. Des symboles (?) indiquent dans la requête que des valeurs seront introduites à cet endroit.

```
1  $db = Db::db();
2
3  $statement = $db->preprare( 'INSERT INTO articles( TitleArticle, IntroArticle,
   ContentArticle ) VALUES ( ?, ?, ? )');
```

Lors de l'assignation des données (qui se fera avec la méthode bind_param()) un type doit être spécifié pour chaque valeur. Cette vérification est effectuée par l'attribution d'une lettre en référence à chaque type prévu dans la requête.

Lettre	Type
i	Entier (interger)
d	Nombre décimal (decimal)
s	Chaîne de caractères (string)
b	Blob (blob)

La méthode bind_param() effectue également une vérification des valeurs afin d'éviter les injections SQL. Ce qui se fait avec la méthode real_escape_string() précédemment présentée.

L'ordre d'apparition des valeurs doit correspondre à ce qui a été déclaré dans la préparation de la requête.

```
5   $statement->bind_param( 'sss',$TitleArticle ,$IntroArticle, $ContentArticle );
```

L'opération se termine par l'exécution de la requête par l'appel de la méthode execute().

```
7   $statement->execute();
```

Voici un exemple complet pour l'insertion de données utilisant la méthode de la requête préparée.

```
1   $db = Db::db();
2
3   $statement = $db->prepare( 'INSERT INTO articles( TitleArticle, IntroArticle,
    ContentArticle ) VALUES ( ?, ?, ? )');
4
5   $statement->bind_param( 'sss', $_POST['TitleArticle'], $_POST['IntroArticle'],
    $_POST['ContentArticle'] );
6
7   $statement->execute();
```

Pour la mise à jour

```
1   $db = Db::db();
2
3   $statement = $db->prepare( 'UPDATE articles SET
                         TitleArticle = ?,
                         IntroArticle = ?,
                         ContentArticle = ?
                         WHERE IdArticle = ?'
             );
4
```

```
5  $statement->bind_param( $stmt, 'sssi', $_POST['TitleArticle'],
6  $_POST['IntroArticle'], $_POST['ContentArticle'], $id );
7
8  $statement->execute();
```

Une classe de gestion des requêtes

L'un des avantages de l'orienté objet est qu'il permet de créer des librairies de codes utiles dans l'exécution d'opérations récurrentes. Les requêtes sont des opérations qui se font maintes et maintes fois et ont souvent un schéma similaire, voire identique. Prévoir une classe qui gère la connexion en plus des requêtes de sélection, d'insertion, de mise à jour et de suppression, s'avère très utile et efficace.

Voici un exemple de classe qui pourrait servir de base pour une librairie.

```
1  Class Queries{
2
3      private $_connect;
4
5      function __construct()
6      {
7          $this->_connect = Db::connect();
8      }
9
10     /**
11     * Effectue la requete
12     *
13     * @param string $query la requete
14     * @return mixed le resultat de mysqli::query()
15     */
16     private function _query( $query )
17     {
18          $result = $this->_connect->query( $query );
19          return $result;
20     }
21
22     /**
23     * Selectionne les donnees
24     *
25     * @param string $query la requete
26     * @return array les donnees | false en cas d'échec
27     */
28     public function select( $query )
29     {
30          $results = $this->_query( $query );
31          if( !$results )
32          {
```

```
33                  $rows = false;
34              }
35              else
36              {
37                  $rows = array();
38                  while( $row = $results->fetch_array() )
39                  {
40                      $rows = $row;
41                  }
42              }
43              return $rows;
44          }
45  
46          /**
47          * Verifie la valeur destinée à une insertion ou mise a jour
48          *
49          * @param string $value la valeur a vérifier
50          * @return string la valeur vérifiée et échapée
51          */
52          public function escape( $value )
53          {
54              return ' \''.$this->_connect->real_escape_string( $value ).'\' ';
55          }
56  
57          /**
58          * Insertion de donnees
59          *
60          * @param string $table la table d'insertion
61          * @param array $values les champs et valeurs respectivement key=>value
62          */
63          public function insert( $table, $values )
64          {
65              $strField = '';
66              $strValue = '';
67              $n = 0;
68              foreach( $values as $field => $value )
69              {
70                  $strField .= ( $n > 0 ) ? $field.', ' : $field.'';
71                  $strValue .= ( $n > 0 ) ? this->escape( $value ).', ' : this->escape(
    $value ).'';
72                  $n++;
73              }
74              $query = 'INSERT INTO' .$table. '(' .$strField . ') VALUES (' .$strValue .
    ')';
75              $this->_query( $query );
76          }
77  
78          /**
79          * Mise à jour de donnees
80          *
```

```
81         * @param string $table la table d'insertion
82         * @param array $values les champs et valeurs respectivement key=>value
83         * @param string $id l'identifiant pour l'insertion
84         */
85        public function update( $table, $values, $id )
86        {
87            $str = '';
88            $n = 0;
89            foreach( $values as $field => $value )
90            {
91                $str .= ( $n > 0 ) ? ', ' : '';
92                $str .= $field. '=' .this->escape( $value );
93                $n++;
94             }
95             $query = 'UPDATE ' .$table. 'SET ' .$str. ' WHERE ' .$id ')';
96             $this->_query( $query );
97        }
98
99        /**
100       * Suppression de donnees
101       *
102       * @param string $table la table d'insertion
103       * @param string $id l'identifiant pour la suppression
104       */
105       public function delete( $table, $id  )
106       {
107            $query = 'DELETE FROM ' .$table. ' WHERE ' .$id;
108            $this->_query( $query );
109       }
110   }
```

L'utilisation de cette classe pourrait se faire comme ceci pour la création d'une sélection.

```
1   $query = new Query()
2   $query->select( 'SELECT * FROM articles' );
3
4   foreach( $rows as $row )
5   {
6   ?>
7       <article>
8           <h2><?php echo $row[ 'TitleArticle' ]; ?></h2>
9       </article>
10  <?php
    }
```

Dans le cas d'une insertion.

```
$values = array(
    'IdArticle' => null,
    'TitleArticle' => $_POST['TitleArticle'],
    'IntroArticle' => $_POST['IntroArticle'],
    'ContentArticle' => $_POST['ContentArticle']
);

$query = new Query()
$query->insert( 'articles', $values );
```

Dans le cas d'une mise à jour.

```
$values = array(
    'TitleArticle' => $_POST['TitleArticle'],
    'IntroArticle' => $_POST['IntroArticle'],
    'ContentArticle' => $_POST['ContentArticle']
);

$query = new Query()
$query->update( 'articles', $values, 'IdArticle' = $id );
```

Enfin la suppression.

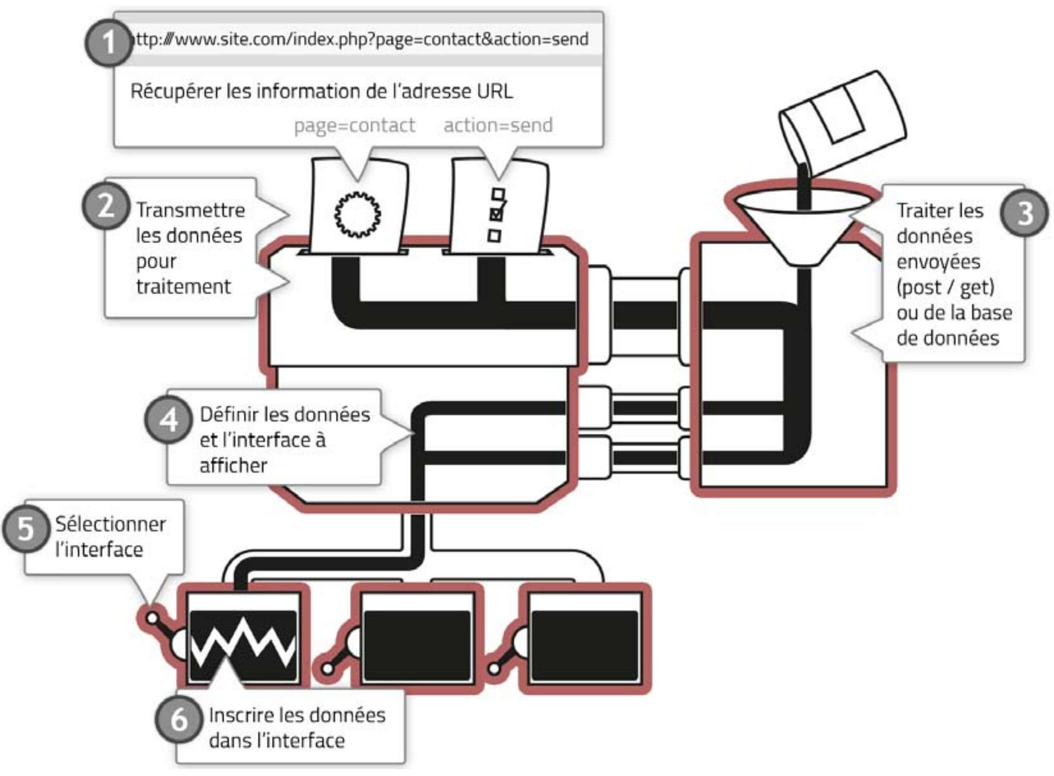
```
$query = new Query()
$query->delete( 'articles', 'IdArticle' = $id );
```

Le processus de traitement (1ère partie)

La transmission de données du navigateur au serveur, l'analyse de ces données et l'affichage d'une interface en guise de réponse comporte toute une série d'opérations qui ont à s'effectuer dans un ordre précis pour qu'ils puissent s'avérer efficaces. Analysons le traitement et les opérations à effectuer pour mener à bien ce processus qui se définit en **6 étapes**.

Afin d'illustrer l'opération, un exemple de traitement d'un formulaire de contact sera utilisé dont le but serait d'envoyer un message par e-mail. Le même processus s'applique aux autres types de traitement que ce soit pour l'insertion de contenus dans une base de données ou la récupération d'un flux RSS, par exemple.

Les 6 étapes du processus de traitement



1. **L'adresse URL**

   Objectif : Définir le format de l'adresse URL et utiliser les informations transmises.

   Le but est de définir un format à l'adresse URL de façon à indiquer la page du site consultée et l'action à mener. En lisant une adresse URL on devrait être en mesure de comprendre les opérations prévues par le site. Ainsi, l'URL peut être composée de deux variables, **« page »** et **« action »**. Cette règle permet d'établir de façon claire les opération prévues par le site.

   Avec l'adresse suivante, http://monsite.com/index.php?page=contact&action=send, on figure que la page concernée prévoit l'envoi des données d'un formulaire de contact sans doute dans un message envoyé par e-mail.

   Dans le code, la première étape consiste à récupérer les valeurs des variables « page » et « action » transmises par l'adresse URL, c'est-à-dire par la méthode « get ».

```
1  $page   = ( empty( $_GET[ 'page' ] ) )     ? '' : $_GET[ 'page' ];
2  $action = ( empty( $_GET[ 'action' ] ) )  ? '' : $_GET[ 'action' ];
```

2. La transmission des données pour traitement

Objectif : Passer les informations à l'outil qui se chargera du traitement des données.

Cette étape à pour but de transmettre l'action demandée par l'utilisateur à la page concernée. Cette étape est déterminante car elle initie une série d'opérations, dont celle du traitement (étape 3) qui retournera des données utiles à l'affichage (étape 4). Le résultat du traitement des données aura une influence sur l'interface à afficher et les données qui seront transmises dans la page.

En reprenant l'exemple du formulaire de contact, il sera question d'appeler une fonction qui servira au traitement des données du formulaire et lui transmettre comme paramètre l'action récupérée à l'étape précédente. Cette fonction pourrait s'appeler « checkMessageSent ». Un résultat est attendu et sera logé dans la variable « $datas ». Comme présenté dans ce qui suit.

```
1  include SITE_PATH . '/application/'.$page.'.php';
2
3  $datas = checkMessageSent( $action );
```

3. Le traitement des données

Objectif : Gérer le traitement des données.

Cette étape comportera des opérations spécifiques qui seront déterminées par l'action à mener. Il peut s'agir d'insérer des données dans une base de données, à l'inverse de récupérer des données pour les afficher dans une page ou comme dans le cas de figure du formulaire de contact d'envoyer un message par e-mail. Dans tous les cas, il est question de données provenant d'une source, d'analyse et de traitement de ces données et de gestion des erreurs.

Source des données

Dans le cas du formulaire de contact les données ont été envoyées par la méthode « post ». Il sera ainsi possible de les récupérer par la variable super globales concernées ($_POST). Dans d'autres cas, la source des données sera la base de données ou encore le contenu d'un fichier...

Analyse et traitement des données

Instaurer des systèmes de vérification qui permettent de vérifier si un champ est vide, si le format, poids ou encore la taille du contenu correspond à ce qui est prévu par le système. La sécurité sera traitée car les données transmises peuvent contenir des informations ou codes volontairement transmises par une personne de mauvaise intention...

Gestion des erreurs

Au fil du traitement des indications sont récupérées concernant l'échec des opération de vérification. Ces erreurs serviront à indiquer s'il est possible ou pas de mener l'opération jusqu'au bout et permettront d'informer l'utilisateur des problèmes rencontrés.

```
1  function checkMessageSent( $action )
2  {
3      $datas = array();
4
5      if( $action === 'send' )
6      {
7          if( empty( $_POST[ 'email' ] ) )
8          {
9              $datas[ 'error' ][ 'emailempty' ] = true;
10         }
11         else if( !filter_var( $_POST[ 'email' ], FILTER_VALIDATE_EMAIL ) )
12         {
13             $datas[ 'error' ][ 'emailformat' ] = true;
14         }
15
16         if( empty( $_POST[ 'message' ] ) )
17         {
18             $datas[ 'error' ][ 'messageempty' ] = true;
19         }
20
21         $datas = $_POST;
22 }
```

Compte tenu de la possibilité que dispose la fonction de ne retourner qu'une seule information, les valeurs (contenus et erreurs) seront inscrites dans un tableau de données *(en occurrence $datas dans le code qui précède)*.

4. L'exécution de l'opération résultante

Objectif : Définir et exécuter l'opération souhaitée par l'utilisateur retourner les informations quant au succès ou l'échec du processus.

L'indicateur déterminant pour cette finalité provient de la gestion des erreurs de l'étape précédente. Grâce à lui, il sera possible de déterminer si l'opération peut être menée jusqu'à son terme ou pas. Dans notre exemple du formulaire de contact, il s'agirait d'envoyer le message par e-mail. Dans le cas où une erreur est détectée pendant le traitement de l'opération, la finalité sera d'afficher de nouveau le formulaire de contact pour que l'utilisateur puisse apporter les corrections qui s'imposent. Dans le cas de la réussite de l'opération, l'utilisateur verra apparaître un message l'informant du succès rencontré.

En fonction de la finalité, **les informations suivantes seraient à retourner** pour mener à bien le reste du processus.

Les données

Elles seront utiles dans l'affichage des contenus dans l'interface telles que les données inscrites dans un formulaire par l'utilisateur (en cas d'erreurs trouvées).

Les erreurs

Dans le cas où il en a. Ceci permettra d'afficher des messages adéquats qui informeront l'utilisateur des problèmes rencontrés.

L'interface à afficher

Cette information est déterminante pour la suite du processus. En fonction du traitement un choix d'interface à afficher est défini. Dans le cas du formulaire de contact, dans le cas où une erreur est trouvée, l'interface choisie sera le formulaire. Dans le cas où le message a pu être envoyé, l'interface choisie sera plutôt celui d'un message informant l'utilisateur du succès de l'opération. Une information définissant l'interface sera également transmise.

```
1  function checkMessageSent( $action )
2  {
3      $datas = array();
4
..     [...]  // Traitement des données (étape 3)
20
21         $datas = $_POST;
22
23         if( !isset( $datas[ 'error' ] ) )
24         {
25             mail('mail@dom.net', 'Subject', $datas['message'], 'From:'.
26 $datas['email']);
27
28             $datas[ 'view' ] = 'contact_sent';
29         }
30         else
31         {
32             $datas[ 'view' ] = 'contact';
33         }
34     }
35     else
36     {
37         $datas[ 'view' ] = 'contact';
38     }
39
40     return $datas;
41 }
```

5. La sélection de l'interface

Objectif : Sélectionner le fichier de l'interface adéquate.

Cette étape est déterminée par le résultat de l'étape précédente. Une interface en HTML est prête à être chargée avec le contenu préalablement préparé à cet effet.

```
1  include SITE_PATH . '/application/'.$page.'.php';
2
3  $datas = checkMessageSent( $action );
4
..   [...]
19
20     <body>
21         <div id="page">
22             <main>
23                 <?php include SITE_PATH . '/view/'.$datas[ 'view' ].'.php'; ?>
24             </main>
25         </div>
26     </body>
27 </html>
```

6. Gestion de l'affichage des données dans l'interface

Objectif : Inscrire les données récupérées dans l'interface.

Des erreurs ainsi que des données ont été transmises (provenant de l'étape 4). Ces informations sont inscrites dans la variable récupérant ce qu'a retourné la fonction servant au traitement des données.

Dans l'exemple du formulaire de contact ceci correspondrait aux cas possible correspondant à l'échec ou au succès de l'opération. Dans le cas de la présence d'une erreur et de l'affichage du formulaire, il sera alors question de tester et afficher les contenus récupérés du formulaire lors de l'envoi par l'utilisateur.

```
1  <h1>Contact</h1>
2
3  <form action="<?php echo SITE_URL; ?>/index.php?page=contact&action=send"
   method="post">
4      <label for="email">
5          <?php echo ( isset( $datas[ 'error' ][ 'emailempty' ] ) ) ? '<span
   class="alert">Aucune adresse n\'a été indiquée</span><br />' : ''; ?>
6          <?php echo ( isset( $datas[ 'error' ][ 'emailformat' ] ) ) ? '<span
   class="alert">Le format de l\'adresse n\'est pas conforme.</span><br />' : ''; ?>
7          <input type="text" name="email" id="email" value="<?php echo ( isset(
   $datas['email'] ) ) ? $datas['email'] : ''; ?>" placeholder="Adresse E-mail" />
```

```
8       </label>
9
10      <label for="message">
11          <?php echo ( isset( $datas[ 'error' ][ 'messageempty' ] ) ) ? '<span class="alert">Aucune message n\'a été indiqué.</span><br />' : ''; ?>
12          <textarea id="message" name="message" placeholder="Votre message"><?php echo ( isset( $datas['message'] ) ) ? $datas['message'] : ''; ?></textarea>
13      </label>
14
15      <button class="btn">Envoyer</button>
16
17  </form>
```

Dans le cas du succès de l'opération, aucun traitement ne serait à envisager.

```
1  <h1>Contact</h1>
2
3  <p>Le message a été envoyé.</p>
```

Introduction au MVC : le processus de traitement (2ième partie)

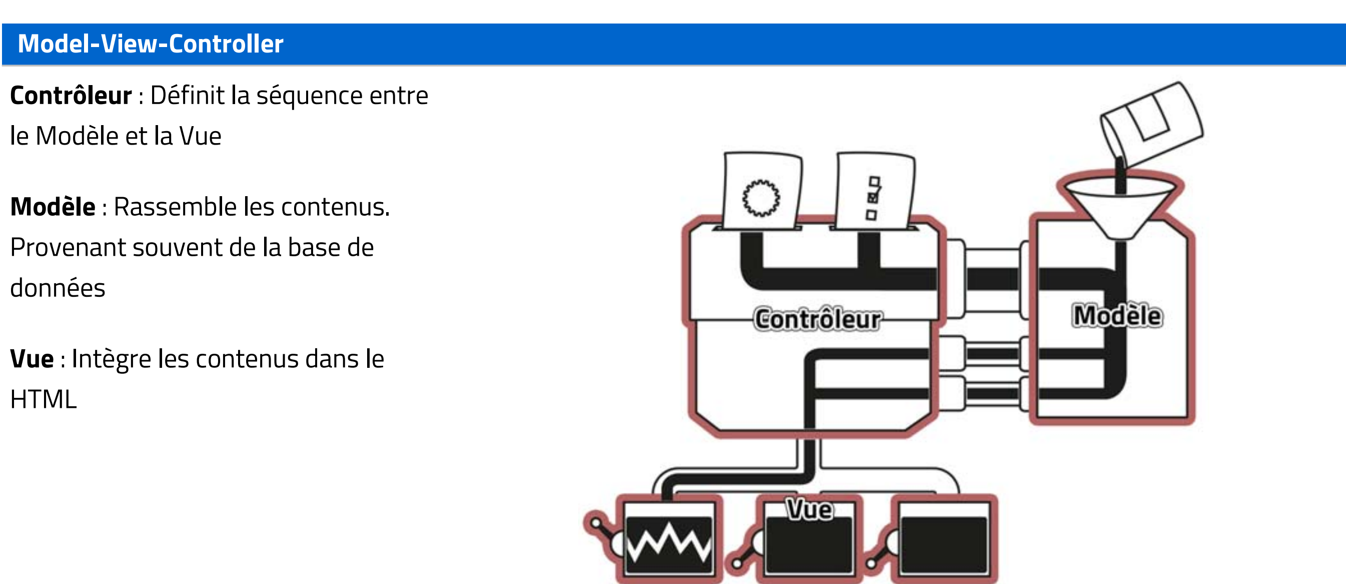
Le patron de conception (ou « design pattern » en anglais) MVC, pour Model, View et Controller est un concept répandu dans les langages de programmation tels que PHP. Le but est de simplifier les opérations courantes de traitement de l'information qui ont été présentées dans le chapitre précédent. L'un des avantages d'un tel concept concerne la méthode de travail qu'il impose. Les informations sont traitées dans un ordre défini et le processus distingue les étapes de traitement très clairement.

Le concept du MVC a également pour but de dissocier les interactions des éventuels modules dont un projet serait composé. Un module de gestion d'actualités et un autre module de gestion des contacts peuvent coexister et être visible dans la même page tout en se gérant l'un indépendamment de l'autre. Pour cela, le projet devra disposer d'un gestionnaire d'affichage ce qui sera traité plus loin dans cette documentation.

Le démarrage et la finalité du processus sont les mêmes que ce qui a été présenté dans le chapitre précédent. Ce qui se précise sont les étapes intermédiaires qui seront groupés dans des opérateurs portant le nom de Model, View et Controller.

Au démarrage Il y a l'URL et le traitement des informations qu'il dispose. Bien qu'il n'y ait aucune allusion concernant le traitement de l'URL dans le nom MVC, il s'agit pourtant bien du déclencheur du mécanisme. Il transmettra les informations pour la suite des opérations. C'est-à-dire la valeur de **page** et celle d'**action**.

Au centre du mécanisme il y a le *Contrôleur*. Il interroge le *Modèle* qui lui transmettra les données qu'il transmettra à son tour à la *Vue* pour l'affichage.



Le contrôleur, le modèle et la vue (MVC) dans la pratique

Dans la pratique cela s'applique à travers la structure des fichiers qui permet de dissocier par modules la gestion des Contrôleurs de modules distincts. Voici un petit rappel de la structure du projet qui permet de localiser les différents éléments.

Répertoires	
application	Les modules
articles	Module spécifique **disposera d'un Contrôleur**
contacts	Module spécifique **disposera d'un Contrôleur**
includes	Contient les outils de mise en route du système
view	Contenus visibles du site
articles	Module spécifique **disposera d'une Vue**
contacts	Module spécifique **disposera d'une Vue**
index.php	

1. Contrôleur

Objectif : Coordonner le traitement des données avec l'affichage.

Cette étape à pour but de coordonner les opérations des deux étapes suivantes (traitement des données et de l'affichage). Le résultat du traitement des données influera sur l'interface à afficher et les contenus transmis à la page.

Le traitement des opérations que fait le **Contrôleur** consiste à transmettre les informations provenant de l'URL (la page à consulter et l'action à mener) et d'attendre le résultat du Modèle pour ensuite transmettre les informations à l'interface adéquate, c'est-à-dire la Vue.

Concrètement chaque module dispose d'un Contrôleur, il s'agit d'une classe qui définira les données et la vue du module. Voici un modèle de Contrôleur « de base » qui illustre sa composition.

```php
1  <?php
2  class Controller {
3
4      private $_action;
5      private $_page;
6      private $_datas;
7      private $_view;
8
9      public function __construct( $page, $action )
10     {
11         $this->_page     = $page;
12         $this->_action   = $action;
13     }
14
15     public function datas()
16     {
17         [...]   // Defines value of $this->_view and sets $this->_datas
18
19         return $this->_datas;
20     }
21
22
23     public function view()
24     {
25         return 'moduleName/'.$this->_view;
26     }
27 }
```

La méthode datas() va traiter les informations et transmettre les données à transmettre à l'interface et la méthode view() va informer de l'interface à afficher. La méthode datas() doit toujours être exécuté avant la méthode view() car elle aura une influence sur l'interface à afficher.

A terme, l'utilisation de la classe Controller() sera définit au démarrage du processus comme présenté dans l'exemple qui suit.

```
1  $page    = ( empty( $_GET[ 'page' ] ) )     ? 'contact' : $_GET[ 'page' ];
2  $action = ( empty( $_GET[ 'action' ] ) )  ? '' : $_GET[ 'action' ];
3
4  include SITE_PATH . '/application/' . $page . '/Controller.php';
5
6  $controller = new Controller( $page, $action );
7
8  $datas       = $controller->datas();
9  $file_name   = $controller->view();
```

Les valeurs des méthodes datas() et view() sont récupérées. Elles serviront à l'affichage des données dans l'interface. Tous les contrôleurs devront donc avoir à disposition une méthode datas() et une méthode view().

2. Modèle

Objectif : Traiter et retourner des données.

La méthode datas() de la classe Controller() est prévue pour gérer le traitement des données. Pour bien comprendre le mécanisme, l'exemple du formulaire de contact sera repris. Ceci permettra de constater comment les codes évoluent à travers ce mécanisme. Le Modèle est ce que fait la méthode _checkMessageSent(). Il récupère les données, les traite et gère les erreurs. La méthode retourne une valeur qui informe de l'interface à afficher.

```
1  <?php
2  class Controller {
3
4      private $_action;
5      private $_page;
6      private $_datas;
7      private $_view;
8
9      public function __construct( $page, $action )
10     {
11         $this->_page      = $page;
12         $this->_action    = $action;
13         $this->_datas     = [];
14         $this->_view      = 'contact';
15
16         if( $this->_action === 'send' )
17         {
18             $this->_checkMessageSent();
```

```
19          }
20      }
21
22
23      private function _checkMessageSent()
24      {
25          if( isset( $_POST[ 'email' ] ) )
26          {
27              $datas = $_POST;
28
29              if( empty( $_POST[ 'email' ] ) )
30              {
31                  $datas[ 'error' ][ 'emailempty' ] = true;
32              }
33              else if( !filter_var( $_POST[ 'email' ], FILTER_VALIDATE_EMAIL ) )
34              {
35                  $datas[ 'error' ][ 'emailformat' ] = true;
36              }
37
38              if( empty( $_POST[ 'message' ] ) )
39              {
40                  $datas[ 'error' ][ 'messageempty' ] = true;
41              }
42
43              $this->_datas = $datas;
44
45              if( !isset( $datas[ 'error' ] ) )
46              {
47                  mail('mail@dom.net', 'Subject', $datas['message'], 'From:'.
$datas['email']);
48
49                  $this->_view = 'contact_sent';
50              }
51          }
52          $this->_view = 'contact';
53      }
54
55
56      public function datas()
57      {
58          return $this->_datas;
59      }
60
61
62      public function view()
63      {
64          return 'articles/'.$this->_view;
65      }
66
67  }
```

3. Vue

Objectif : Afficher l'interface et les données.

Cette étape est identique à celle expliquée dans le processus de traitement. Les informations transmises du Contrôleur permettent de définir une interface et d'afficher les données.

D'abord le choix de l'interface.

```
..  $controller = new Controller( $page, $action );
6
7   $datas      = $controller->datas();
8   $file_name  = $controller->view();
9
10      <body>
11          <div id="page">
12              <main>
13                  <?php include SITE_PATH . '/view/' . $file_name . '.php'; ?>
14              </main>
15          </div>
16      </body>
17  </html>
```

Dans l'exemple du formulaire de contact, il est question de gérer l'affiche des erreurs ainsi que les contenus récupérés du formulaire lors de l'envoi par l'utilisateur.

```
1   <h1>Contact</h1>
2
3   <form action="<?php echo SITE_URL; ?>/index.php?page=contact&action=send"
    method="post">
4       <label for="email">
5           <?php echo ( isset( $datas[ 'error' ][ 'emailempty' ] ) ) ? '<span
    class="alert">Aucune adresse n\'a été indiquée</span><br />' : ''; ?>
6           <?php echo ( isset( $datas[ 'error' ][ 'emailformat' ] ) ) ? '<span
    class="alert">Le format de l\'adresse n\'est pas conforme.</span><br />' : ''; ?>
7           <input type="text" name="email" id="email" value="<?php echo ( isset(
    $datas['email'] ) ) ? $datas['email'] : ''; ?>" placeholder="Adresse E-mail" />
8       </label>
9
10      <label for="message">
11          <?php echo ( isset( $datas[ 'error' ][ 'messageempty' ] ) ) ? '<span
    class="alert">Aucune message n\'a été indiqué.</span><br />' : ''; ?>
12          <textarea id="message" name="message" placeholder="Votre message"><?php
    echo ( isset( $datas['message'] ) ) ? $datas['message'] : ''; ?></textarea>
13      </label>
14
15      <button class="btn">Envoyer</button>
16
17  </form>
```

L'URL rewriting

Le principe de l'URL rewriting est d'offrir une URL lisible autrement de qu'une série de variables et de valeurs lisibles que par le développeur qui a conçu l'application Web.

URL rewriting
Ceci : http://www.monsite.com/index.php?page=articles&action=details&id=*(N)*
Devient : http://www.monsite.com/articles/details/*(N)*

L'ordre des valeurs dans la composition de l'adresse est déterminant. En respectant cette condition les informations pourront être correctement récupérées et utilisées. Voici la règle qui pourrait s'appliquer :

1. La première valeur (suivant le symbole slash (/) qui le séparant du nom de domaine) indique la **page**. Dans l'adresse indiquée dans le précédent exemple, la page appelée serait « articles ». Il correspond au module qui traitera les informations transmises.
2. La seconde valeur indique **l'action** à mener dans la page. Plus précisément dans le module défini. Dans l'exemple, « details » pourrait supposer que le détail d'un article serait à afficher (celui dont l'identifiant sera indiqué après le dernier slash).
3. Les valeurs qui suivent seront transmises via la page au module concerné. Celui-ci (par l'entremise de son Contrôleur) traitera ces valeurs et mènera les opérations en conséquence. Il s'agira donc de données spécifiques et connues par le module.

.htaccess

La disposition d'une adresse telle que http://www.monsite.com/articles/details/*(N)* ne permet pas de traiter son contenu en PHP. Aucune valeur ne peut être récupérée par la méthode « get » avec une adresse dans cet état.

Le fichier .htaccess, communique avec le logiciel http Apache et permet de lui transmettre des indications et réglages. Le fichier .htaccess permet de transmettre au PHP l'adresse sous une nouvelle forme sera cette fois adaptée au traitement.

Les indications suivantes inscrites dans le fichier .htacces transformera les valeurs de l'adresse URL de manière à les rendre accessibles à PHP.

.htaccess

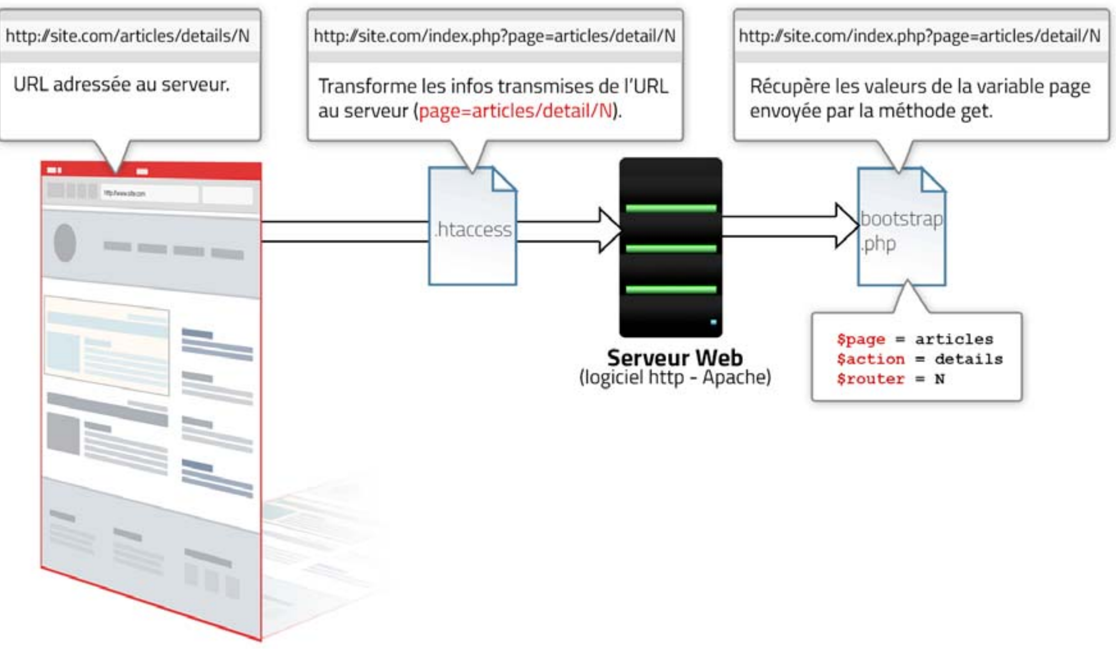
```
1 RewriteEngine on
2
3 RewriteCond %{REQUEST_FILENAME} !-f
4 RewriteCond %{REQUEST_FILENAME} !-d
5
6 RewriteRule ^(.*)$ index.php?page=$1 [L,QSA]
```

Le code ci-dessus du fichier « .htaccess » indique par la ligne `RewriteRule ^(.*)$ index.php?page=$1 [L,QSA]` d'ajouter `index.php?page=` devant les données transmises à PHP.

La gestion des URLs avec PHP

Les informations contenues dans l'adresse URL sont dorénavant accessibles par la variable $_GET['page']. Le format de la valeur n'est cependant pas exploitable dans l'état. Il faudra donc que PHP traite le contenu de $_GET['page'] en isolant les valeurs séparées par le symbole slash (/).

Un outil, nommé « Bootstrap » a la fonction d'exploiter la valeur de l'URL et définir la page du site à charger. Il s'agit d'un contrôleur qui amorce le processus de chargement des outils de la page.



Un fichier « **Bootstrap.php** » contenant une classe du même nom peut ainsi être créé dans le dossier « Includes ». Cette classe a comme mission de :

- Récupérer les valeurs de page et action.
- Récupérer les valeurs complémentaires éventuellement utiles aux outils de la page (modules).
- Définir la page à afficher par défaut (dans le cas où aucune information n'est transmise dans l'adresse URL).
- D'afficher une page 404 dans le cas où l'adresse ne correspond à aucune page du site et de modifier l'entête du fichier afin qu'une erreur 404 soit transmise en réponse au navigateur.
- Charger le Contrôleur de la page choisie.

includes/bootstrap.php

```
1 <?php
2 class Bootstrap{
3
4     public static $_page;
5     public static $_action;
6     public static $_router;
7
8     public static function url()
9     {
10         $router = ( empty( $_GET['page'] ) ) ? 'articles' : $_GET['page'];
11
12         if( !empty( $router ) )
13         {
14             $parts = explode( '/', $router );
15             self::$page    = ( isset( $parts[0] ) ) ? $parts[0] : '';
16             self::$action  = ( isset( $parts[1] ) ) ? $parts[1] : '';
17             self::$router  = ( isset( $parts[2] ) ) ? $parts[2] : '';
18         }
19
20         if( !file_exists( SITE_PATH. '/application/' .self::$_page.
21 '/Controller.php' ) )
22         {
23             header('HTTP/1.0 404 NOT FOUND');
24             include SITE_PATH . '/view/404.html';
25             exit;
26         }
27     }
28 }
```

Cette classe est prête à initier l'opération de gestion de chargement de la page.

includes/init.php

```
1 <?php
2 include SITE_PATH . '/includes/Bootstrap.php';
3 include SITE_PATH . '/includes/Db.php';
4
5 $controller = Bootstrap::url();
6
7 $controller = new Controller( Bootstrap::$page, Bootstrap::$action,
  Bootstrap::$router );
8
9 $datas      = $controller->datas();
10 $file_name  = $controller->view();
```

La gestion de l'affichage

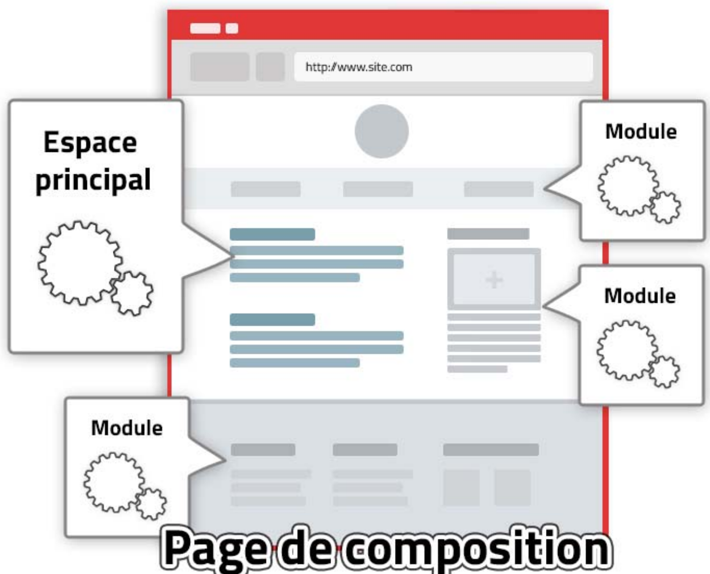
L'opération de la création du thème implique plusieurs étapes. L'ordre d'exécution de ces étapes est déterminant et assure la cohérence de l'opération.

Le processus de transmission des données à la vue

Le processus consiste à assembler les informations visuelles provenant de différents modules qui disposent d'interfaces qui seront jointes, parfois plus d'une fois, dans la même page. Un module peut par exemple gérer les menus de la page. Le module sera appelé autant de fois qu'il y a de menus présents dans la page.

L'assemblage démarre par l'initiation d'une page de composition. Elle dispose d'un contenu principal et de modules complémentaires. L'opération sera dictée par les valeurs des variables $page et $action qui indiquent le module principale à charger.

La page de composition sert en fait de « template » du site. Cette opération d'assemblage sera orchestrée par une classe spécialement conçue pour gérer la composition des rendus de la page et des modules qui la composent.



Pour déclencher l'utilisation de la future classe Template() celle-ci sera appelée juste après la récupération des valeurs de $page, $action et $router afin que ces informations soient transmises à la page de composition.

init.php

```
 1 <?php
 2 include SITE_PATH . '/includes/Bootstrap.php';
 3 include SITE_PATH . '/includes/Db.php';
 4 include SITE_PATH . '/includes/Template.php';
 5
 6 Bootstrap::router();
 7
 8 $urlInfos = array( 'page'=>Bootstrap::$page, 'action'=>Bootstrap::$action,
 9 'router'=>Bootstrap::$router );
10
11 Template::page( $urlInfos );
```

Une classe de gestion des vues

L'opération disposera d'une forme de récursivité car elle implique qu'un fichier effectue des appels de modules qui transmettront des vues disposant eux-mêmes de contenus assemblés. L'orchestration de ceci se fera par la classe « Template » qui aura à :

- effectuer les rendus et les afficher dans la page de composition
- opérer les inclusions des modules, appeler leur contrôleur et récupérer les données et la vue
- définir le contenu par défaut (page d'accueil)

La structure de la classe disposera de 2 méthodes. Une gérera les inclusions et la seconde les rendus pour l'affichage.

Template.php

```
 1 <?php
 2 class Template {
 3
 4     public static function page( $urlInfos )
 5     {
..         // Déclenche l'assemblage avec le rendu de la page principale.
 7     }
 8
 9     private static function _includeInTemplate($page, $action='', $router='')
10     {
..          // Opère les inclusions des modules et transmet pour le rendu
25     }
26
27     private static function _render($filepath, $datas='')
28     {
..          // Effectue les rendus pour l'affichage
38     }
39
40 }
```

1. **Rendu : effectuer les rendus et les afficher dans la page principale**

   La méthode _render() combine les informations qui sont les noms des vues et les données à afficher dans les vues. L'opération de rendu consiste à intégrer les données dans la vue puis afficher le résultat. Afin d'effectuer ceci, l'utilisation de la mémoire tampon sera nécessaire. Les fonctions ob_start(), ob_get_contents() et ob_end_clean() permettent respectivement de faire appel à la mémoire tampon, d'y récupérer des données en mémoire et enfin de libérer la mémoire.

```
Template.php
 1 <?php
.. [...]
27     private static function _render($filepath, $datas='')
28     {
29         if( file_exists( SITE_PATH.'/view/'.$filepath.'.php' ) &&
   is_readable( SITE_PATH.'/view/'.$filepath.'.php' ) )
30         {
31             ob_start();
32             require(  SITE_PATH.'/view/'.$filepath.'.php' );
33             $template = ob_get_contents();
34             ob_end_clean();
35
36             echo $template;
37         }
38     }
39 [...]
```

2. **Constructeur : Appel de la page** de composition

Le constructeur déclenche l'opération de rendu avec l'appel de la page de composition, appelée « page.php » et située dans le dossier des vues (view).

```
Template.php
 1 <?php
.. [...]
 4     public static function page( $urlInfos )
 5     {
 6        self::_render( 'page', $urlInfos );
 7     }
.. [...]
```

La page de composition récupère les données qui lui sont transmises lors du rendu. Etant donné que l'exécution de l'affichage se fait lors de l'opération de rendu, l'appel de la méthode _includeInTemplate() est possible puisque présent dans la même classe.

```
view/page.php
 1 <!DOCTYPE html>
 2 <html>
 3     <head>
 4         <meta charset="UTF-8">
 5         <title>Articles</title>
 6         <link rel="stylesheet" type="text/css" href="<?php echo SITE_URL; ?
```

```
   >/css/style.css" />
 7      </head>
 8      <body>
 9          <div id="page">
10              <main>
11                  <?php self::_includeInTemplate( $datas['page'],
12 $datas['action'], $datas['router'] ); ?>
13              </main>
14          </div>

15      </body>
16 </html>
```

3. Inclusion : Opérer les inclusions des modules et récupérer les données et la vue

L'inclusion consiste à faire appel au contrôleur d'un module afin de récupérer de celui-ci le nom de la vue et les données à intégrer dans celle-ci et transmettre ces informations à l'outil de rendu.

Une vérification est fait afin d'assurer un retour dans le cas où aucun module n'est trouvé. Dans un tel cas, la page home (home.php - page d'accueil) est affichée par l'appel à la méthode _render('home');. Un fichier « home.php » doit par conséquent être présent dans le dossier « view ». Celui-ci disposera des contenus à afficher dans l'espace principal devant figurer sur la page d'accueil du site.

Template.php

```
 1 <?php
.. [...]
 9     private static function _includeInTemplate($page, $action='', $router='')
10     {
11         if( file_exists(SITE_PATH.'/application/'.$page.'/Controller.php') )
12         {
13             include_once SITE_PATH.'/application/'.$page.'/Controller.php';
14
15             $controllerPath = '\application\\'.$page.'\Controller'; //nspace
16
17             $controller = new $controllerPath( $page, $action, $router );
18
19             self::_render( $controller->view(), $controller->datas() );
20         }
21         else
22         {
23             self::_render( 'home' );
24         }
25     }
.. [...]
```

Les espaces de nom

La gestion des espaces de noms est un moyen que dispose PHP depuis sa version 5.3 d'isoler les opérations de différents fichiers. Ainsi des opérateurs et modules pourraient disposer de noms de constantes de classes, de fonctions ou de classes identiques sans qu'un conflit et une erreur PHP ne soit générée.

La définition d'un espace de nom se fait en début de fichier par la déclaration « namespace » suivi du nom qu'on souhaite donner à cet espace.

applications/elements/MyClass.php

```
 1 <?php
 2 namespace application\Elements;
 3
 4 class MyClass {
 5
..          [...]
88
89 }
```

Le nom qui est attribué à l'espace de nom doit être unique dans tout le projet. Il doit être composé de « backslash » ce qui permet de définir une structure par le nom attribué. Ceci permet de créer ainsi une hiérarchie qui peut se référer par exemple à l'organisation des fichiers du projet.

Dans l'exemple qui précède, l'accès à la classe se fera comme suit.

```
 1 <?php
 2 include 'MyClass.php';
 3
 4 $object = new application\Elements\MyClass();
```

Accès à l'extérieur d'un espace de nom

Une fois qu'un espace de nom est défini, les appels de classes ne se feront qu'en référence à cet espace. Une classe situé dans d'autres fichiers ne sera plus trouvé.

applications/elements/MyClass.php

```
 1 <?php
 2 namespace application\Elements;
 3
 4 include SITE_PATH . 'includes/Db.php';
 5
 6 class MyClass {
 7
 8     public function MyMethod()
 9     {
10         $db = Db::connect(); // !!! Erreur : Tente de localiser depuis la
                                //             classe depuis
                                //             application\Elements\Db
..         [...]
88     }
89 }
```

Il faudra par conséquent importer la classe à l'aide du mot-clé « use » et le déclarer en début de fichier.

applications/elements/MyClass.php

```
 1 <?php
 2 namespace application\Elements;
 3
 4 include SITE_PATH . 'includes/Db.php';
 5
 6 use Db; // Appel à la classe à utiliser
 7
 8 class MyClass {
 9
10     public function MyMethod()
11     {
12         $db = Db::connect(); // OK
..         [...]
88     }
89 }
```

Dans le cas où l'élément importé dispose d'un espace de nom également, « use » devra informer du nom complet avec le nom de la classe.

include/Bd.php

```
1  <?php
2  namespace includes\Db;
3
4  class Db {
5
..      [...]
18
19 }
```

applications/elements/MyClass.php

```
1  <?php
2  namespace application\Elements;
3
4  include SITE_PATH . 'includes/Db.php';
5
6  use includes\Db\Db; // Appel à la classe à utiliser
7
8  class MyClass {
9
10     public function MyMethod()
11     {
12         $db = Db::connect(); // OK
..         [...]
88     }
89 }
```