Développement web

PHP

Guillaume Piolle

CentraleSupélec – mineure développement web

25 janvier 2017

Présentation de PHP

Historique

- **1994 :** Rasmus Lerdorf (DK) écrit un ensemble de scripts Perl, *Personal Home Page tools* destinés à la maintenance de sa page personnelle (mise en page de CV et statistiques de visites) ;
- Lerdorf réécrit ses scripts comme des scipts CGI en C et ajoute des fonctionnalités de gestion de formulaire : *Personal Home Page / Form Interpreter* (PHP/FI) ;
- **1995 :** Publication des scripts via Usenet (PHP/FI 1.0). Une petite communauté se crée ;
- **1997 :** PHP/FI 2 ;
- **1997 :** Andi Gutmans et Zeev Suraski fondent *Zend technologies* et réécrivent l'interpréteur, qui devient la base de PHP3. PHP devient l'acronyme de *PHP : Hypertext Preprocessor* :
- **1998 :** PHP 3. Gutmans et Suraski commencent à réécrire le cœur de PHP ;
- **2000 :** PHP 4, interprété par le *Zend Engine 1.0*. Premiers aspects objet ;

Historique

- **2004 :** PHP 5, interprété par le *Zend Engine II*. POO plus aboutie ;
- **2005 :** Projets pour PHP 6, avec support Unicode natif ;
- **2010 :** Abandon de PHP 6, rétroportage de fonctionnalités dans 5.3 et suivants (mais pas Unicode...). Instauration d'une procédure de *release* à base de RFC ;
- **2011 :** Facebook développe *HipHop Virtual Machine* (HHVM), concurrent JIT du Zend Engine ;
- **2014 :** Début des travaux pour une spécification formelle de PHP. Création par Facebook de Hack, dialecte à typage graduel de PHP pour HHVM ;
- **2015 :** PHP 7. *Zend Engine 3*, grosse amélioration des performances, travail sur le typage et le transtypage, beaucoup d'améliorations, rétro-compatibilité presque totale. Zend racheté par Rogue Wave Software.

Actuellement : 5.5 non supportée, 5.6 en *stable release* avec mises à jour de sécurité seulement, 7.0 et 7.1 en *stable release*.

Principes

Mise en œuvre classique pour le web

- Code PHP à l'intérieur de balises spécifiques intégrées au HTML (ou pas) : `<?php ... ?>` ;
- Interprété suite à une requête avant envoi de la réponse au client (par exemple par un module Apache) ;
- Possible modification des en-têtes HTTP ;
- Possible insertion de texte généré à l'endroit du code PHP ;
- Possibles effets de bord côté serveur (système de fichiers, processus, bases de données, communications réseau...).

NB : PHP peut aussi être utilisé comme langage de script, ou interprété en ligne de commande (à la mode Python).

Exemple basique d'intégration PHP/HTML

```
<html>
  <head>
   <title><?php
if  ( isset ($_GET['titre'])) {
  echo $_GET['titre'];
}
else  {
  echo "Titre par défaut";
}
    ?></title>
  </head>
  <body>
  Bonjour, vous venez de <?php echo $_SERVER['HTTP_HOST']; ?>
    <form method="GET">
      Nouveau titre  : <input type="text" name="titre" />
      <input type="submit" />
    </form>
  </body>
</html>
```

Éléments de base du langage

Particularités syntaxiques

La base de la base pour démarrer

- Inclusion dans le HTML à l'aide de `<?php ... ?>` ;
- Les noms de variables commencent par `$`, à la différence des constantes ;
- L'opérateur de concaténation de chaînes est `.` ;
- La syntaxe impérative est globalement « classique » (type C) ;
- Comparaison avec (== / !=) ou sans (=== / !==) conversion de type.

- Typage faible ;
- Typage dynamique.

Fonctions

Déclaration et utilisation de fonctions

```
function maFonction($nom) {
  return "Je m'appelle " . $nom;
}

echo maFonction("Gaspard");
```

- Déclaration conditionnelle possible (déclaration à l'intérieur d'un bloc de code) ;
- Pointeurs de fonctions possible (simple chaîne).

Fonctions

À partir de 5.3 : fonctions anonymes

```
function construireFonction($nom) {
  return function($nom) use ($titre) {
    return "Je suis " . $titre . " " . $nom;
  };
}

$introduction = construireFonction("Balthazar");
echo $introduction("Professeur");
```

Fonction `isset` : test d'existence d'une variable.
Fonction `unset` : désaffectation d'une variable.
Fonction `var_dump` : affichage "debug" d'une variable.

Fonctions

À partir de 5.4 : *type hinting*

Utilisable pour les paramètres formels de type : objets, interfaces, tableaux, *callable*. `NULL` est utilisable pour n'importe lequel de ces types.

```
function maFonction1(MaClasse $objet) {
  //...
}

function maFonction2(array $tableau) {
  //...
}
```

En cas de type non compatible

Émission d'une erreur fatale, `catchable`.

Fonctions

PHP 7 : extension du *type hinting*

Utilisable pour les paramètres formels de type scalaire (`int`, `float`...) et pour les types de retour.

```
function add(float $a, float $b) : float {
  return $a + $b;
}
```

7.1 : introduction du type de retour `void`.

Par défaut, la vérification de type est faible (toutes conversions autorisées). On peut imposer une vérification presque stricte (avec exceptions pour certaines conversions sures) avec `declare(strict_types=1);`

Fonctions et références

Passage par référence

Par défaut le passage se fait par valeur, sauf utilisation de &.

```
function foo(&$arg) {
  $arg++;
}
$a = 2;
foo($a); // $a == 3 en sortie
```

Attention : les objets sont passés par référence, les tableaux par valeur.

Reference mismatch

Si l'on se trompe entre référence et valeur lors de l'appel, il peut y avoir copie. Attention à l'empreinte mémoire en utilisant une API tierce ! Notamment, n'essayez pas de passer manuellement des objets par référence, et n'utilisez pas de références dans un but d'optimisation.

Fonctions et références

Retour par référence

Par défaut le retour se fait par valeur, sauf utilisation de &.

```
<?php
class foo {
    public $value = 42;
    public function &getValue() {
        return $this->value;
    }
}
$obj = new foo;
$myValue = &$obj->getValue(); // $myValue est une référence de $obj->value, qui vaut 42.
$obj->value = 2;
echo $myValue;                // affiche la nouvelle valeur de $obj->value, i.e. 2.
?>
```

N'utilisez pas le retour par référence dans un but d'optimisation, mais uniquement lorsque vous avez foncièrement besoin de manipuler une référence (et posez-vous alors la question : y a-t-il un autre moyen d'atteindre mon objectif ?).

Chaînes de caractères

- Séparées par " : variables interprétées (éventuellement avec utilisation d'accolades pour les noms de variables complexes) ;
- Séparées par ' : variables non interprétées (plus rapide).

Nécessité d'échapper " ou ', suivant le cas, avec \" ou \'.

- Concaténation par . ;
- Accès à un caractère donné : `$chaine[5]` ;
- Conversion chaîne → nombre automatique ;
- Fonctions `substr`, `strpos`, `str_replace`, `strval`, `str_split`, `strlen`, `trim`, `addslashes` / `stripslashes`...

Tableaux

Un tableau PHP peut être vu comme une liste chaînée d'associations clé-valeur. Les clefs sont des entiers (par défaut) ou des chaînes.

```
$tab1 = array(1, 2, 3); //Clés associées : 0, 1, 2
$tab2 = array("foo" => "bar", 7, $tab1);

echo $tab1[1];      // 2
echo $tab2["foo"]; // "bar"
echo $tab2[1];      // $tab1

$tab2[] = 42;        // ajout en fin de tableau

foreach($tab1 as $valeur) { // ou &$valeur au besoin
  // Attention, la liste de valeurs est générée avant la boucle
  echo $valeur;
}

foreach($tab2 as $cle => $valeur) {
  echo $cle . " : " . $valeur;
}
```

Tableaux

Opérateurs

- `$tab1 + $tab2` : union de tableaux (sur égalité clé/valeur) ;
- `$tab1 == $tab2` : les tableaux contiennent les mêmes paires clés/valeurs ;
- `$tab1 === $tab2` : les tableaux contiennent les mêmes paires clés/valeurs, dans le même ordre et du même type.

Fonctions utiles

`array_fill`, `array_map`, `print_r` (affichage récursif), `array_keys`, `array_key_exists`, `array_merge`, `sort` (tri des valeurs par attribution de nouvelles clés), `asort` (tri des paires clé-valeur)...

PHP et l'égalité

- == : comparaison **avec transtypage automatique** ;
- === : comparaison **sans transtypage automatique**.

Le problème de la comparaison des hachés

'0e123456' == '0e654321'

Pourquoi ? Parce que toute chaîne de caractère commençant par '0e' est convertie automatiquement en un nombre en notation scientifique, en l'occurence zéro !

Ce n'est qu'une raison parmi d'autres pour **toujours utiliser === (ou !==) et jamais == (ni !=)**.

PHP et l'égalité

== vs === : c'est pas fini...

Non-transitivité de ==

```
$a = 0;
$b = 'x';
 false  == $a; // true
$a == $b;    // true  ????
$b == true;  // true
```

PHP et l'égalité

Et c'est pas tout ! Il ne suffit pas de s'interdire == et !=, parce que le transtypage automatique de == est intégré par défaut à de nombreuses fonctions...

Il faut toujours s'assurer d'utiliser les versions « strictes » des fonctions lorsqu'elles sont disponibles.

Recherche dans un tableau

```
in_array('abc', array(0));          // true
in_array('abc', array(0), true);    // false

$a = array('3.1');
in_array('3.10', $a);               // true
in_array('7.10', $a, true);         // false
```

PHP et la précision des flottants

PHP n'est juste pas le bon langage pour faire du calcul (sauf à utiliser une bibliothèque logicielle spécialisée), mais c'est le cas de la plupart des langages de programmation !

```
$a = 0.1*0.1;
var_dump($a); // double(0.01)
var_dump($a == 0.01); // false

$a = 0.1+0.2-0.3;
echo $a; // 5.5511151231258E-17
```

Explication : de nombreux décimaux, comme 0.1 et 0.01 ne sont pas représentables de manière exacte en binaire.

Tous les langages de programmation souffrent de ces problèmes, mais ils le contournent de manière plus ou moins satisfaisante.

Gestion des erreurs

Le niveau d'erreur (et le fait qu'elles soient affichées) est fixé par la configuration mais peut être modifié pour un script donné par `error_reporting`.

Indépendamment de l'affichage, les erreurs peuvent également être journalisées (journaux système ou serveur de journalisation).

Suivant la configuration, `$php_errormsg` peut contenir le dernier message d'erreur.

`...or die($message)` affiche la chaîne `$message` et termine le script si l'expression de gauche est équivalente à faux.

Gestion des erreurs

Principaux niveaux d'erreurs

1	0001	`E_ERROR`	**Fatal error** (met un terme au déroulement du script)
2	0002	`E_WARNING`	**Warning**
4	0004	`E_PARSE`	**Parse error** (empêche le lancement du script)
8	0008	`E_NOTICE`	**Notice** (fait souvent référence à la discipline de codage)
...			
256	0100	`E_USER_ERROR`	Erreur déclenchée par le développeur (`trigger_error`)
512	0200	`E_USER_WARNING`	Avertissement déclenché par le développeur
1024	0400	`E_USER_NOTICE`	Notification déclenchée par le développeur
2048	0800	`E_STRICT`	Notification relative aux recommandations officielles de PHP
4096	1000	`E_RECOVERABLE_ERROR`	Une erreur fatale pas si fatale que ça :-\
8192	2000	`E_DEPRECATED`	Alerte d'interopérabilité avec les versions futures
16384	4000	`E_USER_DEPRECATED`	Alerte d'interopérabilité déclenchée par le développeur
32767	7FFF	`E_ALL`	Tous les niveaux (à l'exclusion de `E_STRICT` pour les versions $< 5.4.0$)

Inclusion

- `include` : tente d'inclure un fichier PHP (typiquement, des déclarations) à l'endroit où la directive est placée. En cas d'erreur, un `E_WARNING` est lancé ;
- `include_once` : idem, mais n'inclut chaque fichier qu'une seule fois (protection contre l'inclusion multiple de bibliothèques) ;
- `require` : similaire à `include`, mais le fichier est considéré nécessaire pour l'application : une erreur fatale interne de niveau `E_COMPILE_ERROR` est lancée en cas d'échec ;
- `require_once` : idem, mais n'inclut chaque fichier qu'une seule fois.

Le modèle objet de PHP

Classes et objets

Déclaration de classe

```php
class MaClasse {
  const n = 3; // constante de classe
  private $prop; // également public (par défaut), protected
  public static $stat = 0; // propriété de classe, depuis 5.3

  public function __construct($p) {
    echo get_class( $this );
    $this->prop = $p;
  }
  public function getN() {
    return self::n;
  }
  public function __destruct() {...}
}
```

```php
$obj = new MaClasse(5);

echo MaClasse::n;

echo $obj::$stat;
echo MaClasse::$stat;

$obj->oups = "nouveau";
```

- Par défaut, attributs et méthodes sont `public` ;
- `protected` n'a pas la même spécification qu'en Java : la variable est visible uniquement dans la classe et dans les classes dérivées ;
- Descripteurs de visibilité pour les constantes de classe : seulement à partir de 7.1.

Classes et objets

Instanciation, affectation, références et clonage

```
$instance = new MaClasse(5);

$sameRef = & $instance; // $sameRef et $instance sont la même référence
$newRef = $instance; // $newRef et $instance sont deux références
                     // distinctes pointant sur la même instance

$copie = clone $instance; // fait appel à une méthode __clone()
$instance = null; // $instance et $sameRef pointent sur null,
                  // $newRef pointe toujours sur l'instance
```

Espaces de nommage

Introduits à partir de PHP 5.3. Script de déclarations :

```
namespace Vendor\Product; // déclaration/définition

class MaClasse{}
```

Script d'utilisation :

```
use Vendor\Product\MaClasse as C; // utilisation et déclaration d'alias

$obj = new C();
$obj = new MaClasse();
```

L'imbrication des espaces de noms doit correspondre à des répertoires pour permettre l'autochargement des classes.

Héritage

```
class Parent {
  function __construct() {...}
  function blah() {...}
}

class Enfant extends Parent {
  function __construct() {
    parent::__construct(); // appel explicite
}
  function blah() {
    echo "Redéfinition de blah :";
    parent::blah();
  }
}
```

Un objet `$object` est-il de la classe `MaClasse` (ou d'une classe fille) ?

```
if ($object instanceof MaClasse) // Sans guillemets !
if ($object instanceof $nomClasse)
```

Héritage

- `self` : portée de la classe courante ;
- `parent` : portée de la classe parente ;
- `static` : portée des attributs et méthodes statiques ;
- `$this` : référence sur l'objet courant.

PHP propose des classes abstraites et des interfaces avec une syntaxe à peu près similaire à celle de Java.

Attention à `$this` : non définie si la méthode est appelée statiquement, peut pointer sur un objet d'une autre classe si c'est une méthode de cette autre classe qui fait l'appel statique ! Ce type d'appel lève un `E_DEPRECATED` depuis 5.6, `$this` devient indéfinie depuis 7.0.

Méthodes magiques

Ensemble de méthodes de rappel (*callback*) relatives aux objets :

- `__construct`, `__destruct`, `__clone`, `__toString` ;
- `__set`, `__get`, `__isset`, `__unset` : appelées lors de manipulation de propriétés normalement inaccessibles (non déclarées ou non visibles) ;
- `__call`, `__callstatic` : appel de méthodes normalement inaccessibles
- `__sleep`, `__wakeup` : appels à `serialize` et `unserialize` ;
- et quelques autres.

Gestion des exceptions

Gestion à la mode Java

```
try {
  throw new Exception('message');
  // On peut bien sûr créer des classes dérivées
}
catch (Exception $e) {
  echo $e->getMessage();
}
```

Pas de `throws` à déclarer dans les méthodes !

Méthodes de la classe `Exception`

`getMessage()`, `getCode()`, `getFile()`, `getLine()`, `getTrace()` (renvoie un tableau), `getTraceAsString()`.

PHP7 : Les erreurs `E_ERROR` et `E_RECOVERABLE_ERROR` donnent lieu au lancement d'objets `Throwable` (interface réalisée par `Exception`).

Environnement

Le fichier php.ini

Configuration du moteur PHP, ensemble de directives clés-valeurs.
La fonction `ini_set` (resp. un fichier `.htaccess`) permet de redéfinir localement certains réglages pour un script donné (resp. pour un répertoire).

Aspects de la configuration dans `php.ini` :

- Gestion des variables globales, options syntaxiques ;
- Gestion des erreurs, de l'encodage, des uploads ;
- Limites de ressources, options de compatibilité ascendante ;
- Configuration par hôte ou par répertoire ;
- ...

Affichage de la configuration avec `phpinfo()`.

Variables superglobales

Tableaux associatifs accessibles au développeur :

- `$GLOBALS` : Toutes les variables globales du script ;
- `$_SERVER` : Environnement fourni par le serveur (clés utiles : `SERVER_ADDR`, `REQUEST_METHOD`, `HTTP_HOST`, `HTTP_USER_AGENT`, `REMOTE_USER`...
- `$_GET`, `$_POST` : Variables provenant de la requête HTTP correspondante ;
- `$_FILES` : Fichiers uploadés par POST ;
- `$_COOKIE` : Variables de cookies transmises dans la requête HTTP ;
- `$_SESSION` : Environnement de la session ;
- `$_REQUEST` : Union de `$_GET`, `$_POST` et `$_COOKIE` (usage déconseillé !) ;
- `$_ENV` : Variables d'environnement passées à PHP (cf. CLI).

Constantes magiques

Syntaxiquement des constantes PHP, leur contenu varie suivant le contexte :

- `__LINE__` : Numéro de ligne dans le fichier ;
- `__FILE__` : Nom et chemin complet du fichier ;
- `__DIR__` : Répertoire du fichier ;
- `__FUNCTION__`, `__CLASS__`, `__METHOD__`, `__NAMESPACE__` : Nom de la fonction, classe, méthode ou namespace dans lequel on se trouve.

Cookies et sessions

Gestion des en-têtes HTTP

Comportement par défaut : dès que le script imprime un caractère (y compris un retour chariot ou une espace), les en-têtes HTTP sont envoyés (ou en tout cas fixés).
Les modifications des en-têtes doivent donc se faire avant tout affichage. Cela concerne :

- La gestion des cookies ;
- La gestion des sessions (par conséquent) ;
- Les redirections et erreurs HTTP ;
- Les options HTTP : type MIME, encodage, langue...

Possible dépassement de cette limitation avec les fonctions d'*output buffering* (fonctions `ob_*` et `flush`).

Gestion des en-têtes HTTP

La fonction `header` permet d'envoyer directement une en-tête forgée (ou des portions) au client.

```
header('Location: http://www.example.com/'); // code 302 automatique
header("HTTP/1.0 404 Not Found");
header('Content-type: application/pdf');
```

Exemple : envoi d'un PDF

```
header('Content-type: application/pdf');
header('Content-Disposition: attachment; filename="downloaded.pdf"');
readfile ('original.pdf');
```

Gestion des cookies

Envoi des infos

Envoi manuel de cookie via la fonction `setcookie` :

```
setcookie("TestCookie", $value, time()+3600, "/~rasmus/",
"example.com", false, true);
```

Paramètres : nom du cookie, valeur, date d'expiration, *path*, domaine, *secure*, *httponly*.

Récupération des infos

Transparente via `$_COOKIE`.

Gestion des sessions

Techniques de propagation de l'état de session

- Cookie de session (le plus courant) : géré automatiquement ;
- Encodage dans les paramètres de requêtes (solution de repli en cas de refus des cookies ou de choix dans le paramétrage).

Utilisation basique

- `session_start()` en tête de script ;
- Écritures et lectures dans `$_SESSION`, sans oublier les tests `isset` ;
- Suppression de variables de session : `unset($_SESSION['nomvar']);`
- Fermeture par `session_unset()` / `session_destroy()`.

Bibliothèques logicielles

Accès aux bases de données

PDO (*PHP Data Objects*)

Couche d'abstraction pour l'accès à un SGBD compatible (nécessite l'existence d'un pilote PDO pour le SGBD en question).

```
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
    foreach($dbh->query('SELECT * from FOO') as $row) {
        print_r ($row);
    }
    $dbh = null;
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die ();
}
```

\+ gestion de transactions, requêtes stockées

Gestion du XML

Gestion simpliste : SimpleXML

Simple. . . tant que le traitement à réaliser reste simple (parcours linéaire, requête XPath). Sinon, passer à DOM.
Fonctions `simplexml_*`, classes `SimpleXMLElement` et `SimpleXMLIterator`.

Gestion événementielle : SAX

XML Parser, fonctions `xml_*`

Gestion objet riche : DOM

Classes `DomDocument`, `DOMElement`, `DOMNode`, `DOMNodeList`, `DOMXPath`, `DOMException`. . .

Autres extensions

PECL (en C, `pecl.php.net`), Composer (en PHP, `getcomposer.org`)

- Expressions régulières PCRE (fonctions `preg_*`) ;
- Authentification (Kerberos, PAM, radius. . .), chiffrement, SELinux ;
- Gestion des dates et du temps ;
- Manipulation de HTML, web sémantique ;
- Images et multimédia (cairo, FreeImage. . .) ;
- Interpréteurs embarqués (Lua, Perl, Python, JS) ;
- Bibliothèques mathématiques ;
- Bibliothèques réseau, caching, mail, HTTP ;
- Paiement en ligne ;
- Internationalisation ;
- . . .

Frameworks MVC pour PHP

Zend framework

Développé par Zend (`http://framework.zend.com`)
Gratuit / BSD-like, fonctionnement facilité avec Zend Server et Zend Studio (mais peut fonctionner sur un autre serveur PHP5).
AJAX/JSON facilité, moteur de recherche intégré, moteur de syndication, gestion de l'authentification, des autorisations et des sessions, interface avec SGBD, mail, système de cache...

Symfony

Développé par SensioLabs (`https://sensiolabs.com/fr`)
FOSS, licence MIT, développé à l'origine par une entreprise française (SensioLabs). De nombreuses applications, y compris d'autres frameworks de développement (comme Laravel), s'appuient sur Symfony.
Configuration en YAML, *scaffolding* (échafaudage, génération de code pour l'interaction SGBD), internationalisation, AJAX, mapping objet-relationnel, système de plugins...

Références

- PHP, site officiel : `http://www.php.net/` ;
- Eric Daspet & Cyril Pierre de Geyer, *PHP 5 avancé*, Eyrolles, 2004 ;
- Josh Lockhart, *Modern PHP*, O'Reilly, 2015 ;
- Anthony Ferrara, *PHP RFC : Scalar Type Declarations*, The PHP.net wiki, 2015 (`https://wiki.php.net/rfc/scalar_type_hints_v5`) ;
- Julien Pauli, *Reference mismatch in PHP function calls*, Julien Pauli PHP's life, 2014 (`http://jpauli.github.io/2014/06/27/references-mismatch.html`) ;
- Aaron Piotrowski, *Throwable Exceptions and Errors in PHP 7*, trowski.com, 2015 (`https://trowski.com/2015/06/24/throwable-exceptions-and-errors-in-php7/`) ;
- Guillaume Rossolini, *Cours de PHP 5*, Developpez.com, 2008-2010 (`http://g-rossolini.developpez.com/tutoriels/php/cours/`).