



Formation PHP Perfectionnement

Romain Bohdanowicz

Twitter : @bioub - <https://github.com/bioub>

<http://formation.tech/>



Introduction

Introduction - Formateur



- Romain Bohdanowicz
Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle



- Expérience
Formateur/Développeur Freelance depuis 2006
Près de 2000 jours de formation animées

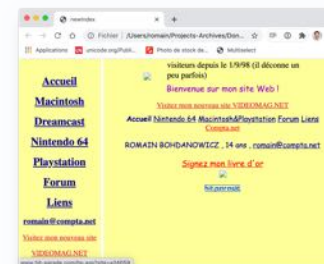
- Langages
Expert : HTML / CSS / JavaScript / TypeScript / PHP / Java
Notions : C / C++ / Objective-C / C# / Python / Bash / Batch



- Certifications
PHP / Zend Framework / Node.js



- A propos
Premier site web à 12 ans (HTML/JS/PHP)
Triathlète du dimanche



Introduction - Horaires



- Matin
 - 9h - 10h
 - 10h15 - 11h15
 - 11h30 - 12h30
- Après-midi
 - 13h45 - 14h45
 - 15h - 16h
 - 16h15 - 17h15
- Questionnaire de satisfaction à remplir en fin de formation :
<https://stagiaire.formation.tech/>



- Organisme de formation depuis 2016
- Référencé DataDock
- Certifié Qualiopi
- 15 formations au catalogue
- Une dizaine de formateurs indépendants
- Formations en français ou anglais
- <https://formation.tech/>



Introduction - Et vous ?



- Pré-requis ?
- Rôle dans votre société ?
- Intérêt / objectif de cette formation ?



Rappel sur PHP



- PHP 1 (1994)
Rasmus Lerdorf créer un ensemble d'outils CGI pour son site sur lequel il publie son CV, PHP veut alors dire Personal Home Page
- PHP 2 (1995)
PHP devient un langage de programmation à part entière, la syntaxe est inspirée du BASIC et les fonctions utilisateurs sont désormais possibles
- PHP 3 (1997)
Andi Gutmans et Zeev Suraski alors étudiant à Tel Aviv réécrivent entièrement le coeur de PHP, la syntaxe s'inspire du C. PHP devient un acronyme récursif - PHP: Hypertext Preprocessor.
- PHP 4 (1999)
Andi Gutmans et Zeev Suraski réécrivent le coeur de PHP qui se nomme désormais Zend Engine (puis la société Zend), introduction de la programmation orientée objet
- PHP 5 (2004)
Zend Engine 2, importante réécriture de la partie objet qui s'inspire désormais de Java : Classes Abstraites, Interfaces.
- PHP 7 (2015)
Pas de PHP 6 car une version a été en cours d'écriture puis abandonnée et des livres avaient été écrits. Principale nouveautés, le typage statique devient possible sur tous les types.
- PHP 8 (2020)
Compilation JIT, paramètres nommés, attributs, type unions...

PHP - A quoi ça sert ?

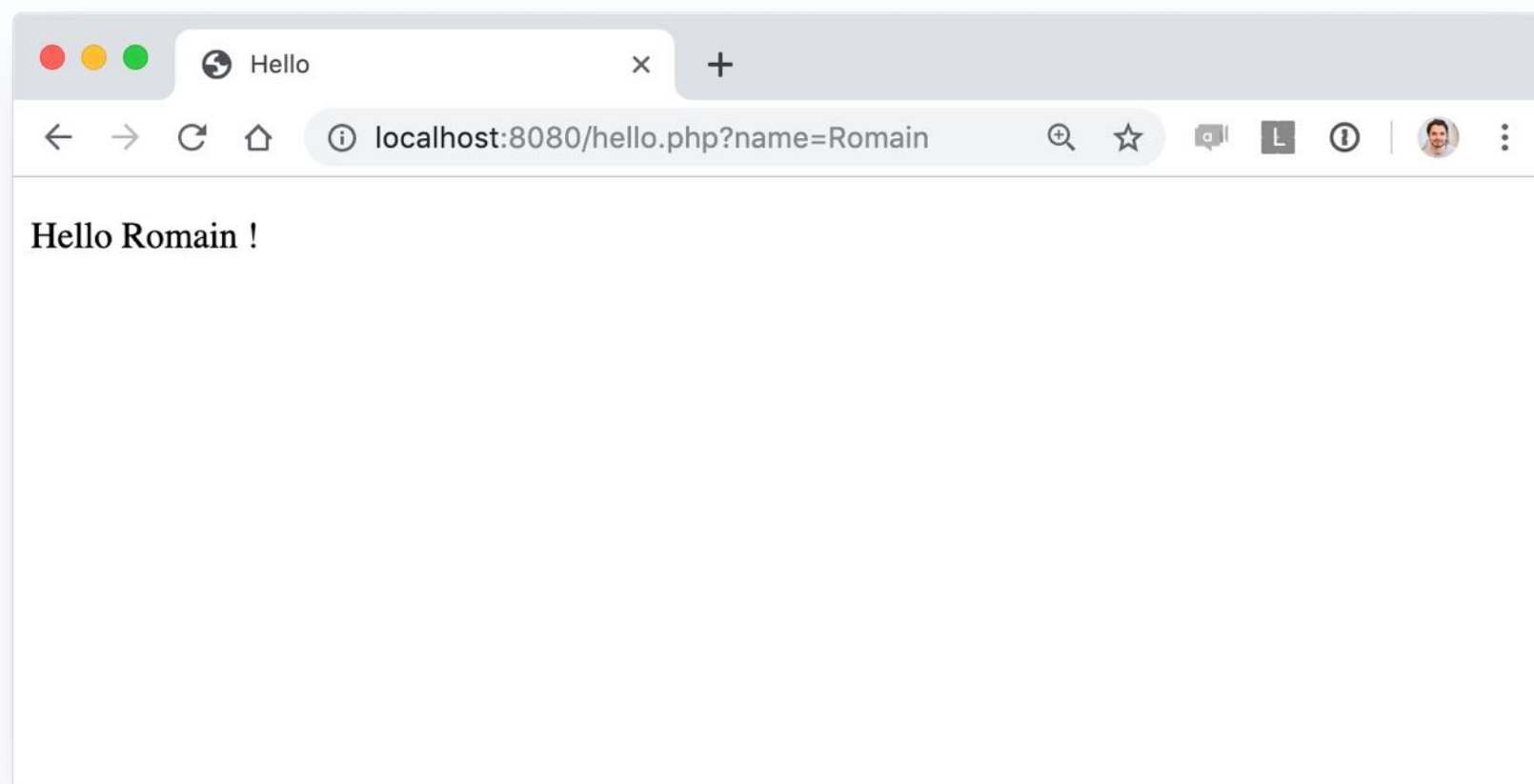


- Aujourd'hui on peut écrire 3 types de programmes en PHP
 - des scripts serveurs (la plupart du temps)
 - des scripts en ligne de commande (rarement)
 - des applications graphiques (quasiment jamais)

PHP - Exemple de script web



```
<?php
$name = $_GET['name'] ?? '';
?>
<p>Hello <?=$name?> !</p>
```



PHP - Exemple de script en ligne de commande (CLI)



```
<?php
$name = $argc > 1 ? $argv[1] : '';
echo "Hello $name\n";
```

A terminal window with a title bar that reads "Slides — -bash — 53x10". The terminal shows the command "\$ php hello.php Romain" being entered, followed by the output "Hello Romain" and a new prompt "\$".

```
Slides — -bash — 53x10
[$ php hello.php Romain
Hello Romain
$
```

PHP - Application Graphiques



- PHP permet l'écriture d'application graphiques avec PHP-GTK
- La documentation est inexistante
- Préférez d'autres langages si le besoin de fait ressentir





- On écrit le code dans des balises PHP

```
<?php
function hello($name)
{
    return "Hello $name\n";
}

$names = ['Romain', 'Edouard'];

foreach ($names as $name) {
    echo hello($name);
}
```

- Il n'y a pas besoin d'utiliser la balise fermante en fin de fichier ?> (et c'est même recommandé)
- Pour afficher quelque chose, on appelle utilise la construction de langage *echo*



- Lorsque contenu de la balise PHP ne fait qu'un echo d'une valeur, on peut utiliser la syntaxe courte
- Ainsi

```
<p><?php echo 'Bonjour' ?></p>
```

- Est l'équivalent de

```
<p><?= 'Bonjour' ?></p>
```

PHP - Échappement depuis du HTML



- On peut utiliser PHP comme un moteur de template
- C'est à dire que dès qu'on écrit en dehors d'une balise PHP le contenu sera écrit sur la page

```
<?php
function hello($name)
{
    return "Hello $name\n";
}

$names = ['Romain', 'Edouard'];
?>
<ul>
    <?php foreach ($names as $name) : ?>
        <li><?= hello($name) ?></li>
    <?php endforeach; ?>
</ul>
```



- Une variable permet de mettre de côté une valeur pour plus tard
- On déclare une valeur en commençant par \$

```
<?php  
$prenom = 'Romain';  
$age = 33;  
$estFormateur = true;  
  
echo $prenom; // Romain
```




- En PHP une valeur peut être de type :
string, int, double, boolean, array, object (et tous ces dérivés), resource, null, callable
- Une expression peut être assimilée à "tout ce qui a une valeur" (valeur, résultat d'un calcul, appel de fonction avec un retour...)

```
<?php
var_dump('texte'); // string(5) "texte"
var_dump(123); // int(123)
var_dump(1.2); // double(1.2)
var_dump(true); // bool(true)
var_dump([]); // array(0) {}
var_dump(new stdClass()); // class stdClass#1 (0) {}
var_dump(fopen('/dev/null', 'r')); // resource(5) of type (stream)
var_dump(null); // NULL
var_dump(function () {}); // callable
```



- Les chaînes de caractères (string) représentent du texte
- L'opérateur `.` permet de concaténer deux chaînes entre elles (les assembler)
- L'opérateur `[]` permet d'accéder au caractère présent à un certain indice
- Les autres opérations se feront via des fonctions
<https://www.php.net/manual/fr/ref.strings.php>
- C'est le type le plus répandu, faites le tour des fonctions savoir ce qu'il est possible de faire !

```
echo 'Romain'; // Romain
echo 'Romain' . ' ' . 'Bohdanowicz'; // Romain Bohdanowicz
echo strtoupper('Romain'); // ROMAIN
echo 'Romain'[0]; // R
```



- Echappement

Si la chaîne de caractère contient son délimiteur il faudra l'échapper en le préfixant avec un antislash \

```
echo 'Je m\'appelle Romain';  
echo "<a href=\"https://formation.tech/\">formation.tech</a>";
```

- Pour éviter d'échapper on peut pour des cas complexes utiliser les syntaxes Nowdoc et Heredoc qui permettent de remplacer les délimiteurs (ici par STR)
- Attention, il ne faut pas de caractères après STR sur la partie ouvrante et avant ou après STR sur la fermante

```
// Nowdoc  
echo <<<'STR'  
<a href="https://formation.tech/">C'est mon site !</a>  
STR;  
// Heredoc  
echo <<<STR  
<a href="https://formation.tech/">C'est mon site !</a>  
STR;
```



- Les chaînes de caractères déclarées avec les guillemets et Heredoc sont interprétées
 - Les variables sont remplacées par leur contenu
 - Les caractères tabulation, retour à la ligne sont pris en compte

```
$prenom = 'Romain';  
echo 'Bonjour $prenom\n'; // Bonjour $prenom\n  
echo "Bonjour $prenom\n"; // Bonjour Romain
```

```
$url = 'https://formation.tech/';  
$site = "C'est mon site !";  
// heredoc  
echo <<<STR  
<a href="$url">$site</a>  
STR;
```



- Le type booléen peut contenir 2 valeurs
 - `true` (vrai)
 - `false` (faux)
- C'est le résultat d'un test
- On l'utilise pour activer ou non une partie du code

```
<?php
if (true) {
    echo 'sera affiché';
}

if (false) {
    echo 'ne sera pas affiché';
}
```



- Opérateurs de comparaison
- Leur résultat est de type booléen

```
$age = 33;  
var_dump($age > 33); // false  
var_dump($age >= 33); // true  
var_dump($age < 33); // false  
var_dump($age <= 33); // true  
var_dump($age == 33); // true  
var_dump($age === 33); // true  
var_dump($age != 33); // false  
var_dump($age !== 33); // false
```

- Opérateur == vs ===

```
$age = 33;  
var_dump($age == '33'); // true  
var_dump($age === '33'); // false
```



- Il existe 3 opérateurs permettant de combiner des booléens (opérateurs logiques)
- Par exemple on ne peut pas écrire

```
$age = 33;  
var_dump(0 < $age < 120); // PHP Parse error: syntax error, unexpected '<'
```

- Car l'expression `0 < $age` est de type booléen et ne pourra pas être comparée avec l'opérateur `<`
- A la place il faut séparer 2 comparaisons `0 < $age` et `$age < 120` et les regrouper avec un opérateur logique

```
$age = 33;  
var_dump(0 < $age && $age < 120); // true
```



- ET Logique - `&&`
Toutes les expressions doivent être vraies pour que l'ensemble soit vrai
- OU Logique - `||`
Une seule expression doit être vraie pour que l'ensemble soit vrai
- NON Logique - `!`
Inverse le booléen

```
var_dump(true && true); // true
var_dump(true && false); // false
var_dump(false && true); // false
var_dump(false && false); // false

var_dump(true || true); // true
var_dump(true || false); // true
var_dump(false || true); // true
var_dump(false || false); // false

var_dump(!true); // false
var_dump(!false); // true
```




- Les structures conditionnelles permet d'exécuter ou non certaines instructions, selon une condition (expression booléenne)

```
<?php
if (/*condition*/) {
    /* bloc d'instructions; */
}
```

- Exemple

```
if (true) {
    echo 'sera affiché';
}

if (false) {
    echo 'ne sera pas affiché';
}
```

- Lorsqu'il n'y a qu'une seule instruction, les parenthèses ne sont pas obligatoires (mais recommandées)

```
if (true) echo 'sera affiché';
```



- Un bloc `if` peut être lié à un bloc `else` qui sera évalué si l'expression est fausse

```
if (true) {  
    echo 'sera affiché';  
} else {  
    echo 'ne sera pas affiché';  
}  
  
if (false) {  
    echo 'ne sera pas affiché';  
} else {  
    echo 'sera affiché';  
}
```



- Si un bloc `else` contient à nouveau un `if`

```
if ($nb < 10) {  
    echo "$nb est inférieur à 10";  
} else {  
    if ($nb < 20) {  
        echo "$nb est compris entre 10 et 20";  
    } else {  
        echo "$nb est supérieur à 20";  
    }  
}
```

- Il sera plus léger de ne pas mettre les accolades du `else` :

```
if ($nb < 10) {  
    echo "$nb est inférieur à 10";  
} else if ($nb < 20) {  
    echo "$nb est compris entre 10 et 20";  
} else {  
    echo "$nb est supérieur à 20";  
}
```



- Un tableau permet de regrouper plusieurs valeurs
- Un tableau numérique permet de retrouver sa valeur par rapport à la position dans le tableau
- Un tableau associatif permet de retrouver sa valeur par rapport à une autre (une clé)
- Il peuvent être hybrides (numériques / associatifs), mais ce serait une mauvaise pratique

```
<?php
// tableau numérique
$nbs = [2, 3, 4];

// tableau associatif
$capitales = [
    'France' => 'Paris',
    'Allemagne' => 'Berlin',
    'Espagne' => 'Madrid',
];
```



- Pour accéder aux valeurs d'un tableau on utilise les crochets []

```
<?php
// tableau numérique
$nbs = [2, 3, 4];
echo $nbs[0]; // 2

// tableau associatif
$capitales = [
    'France' => 'Paris',
    'Allemagne' => 'Berlin',
    'Espagne' => 'Madrid',
];
echo $capitales['France']; // Paris
```

- Les fonctions sur les tableaux :
<https://www.php.net/manual/fr/ref.array.php>



- Les boucles permettent de répéter plusieurs fois un bloc d'instructions
- while (nombre de passages inconnus à l'avance, entre 0 et n passages)

```
$nb = mt_rand(0, 100);  
  
while ($nb) {  
    echo $nb;  
    $nb = (int) ($nb / 2);  
}
```

- do ... while (nombre de passages inconnus à l'avance, entre 1 et n passages)

```
do {  
    $nb = mt_rand(0, 100);  
} while ($nb % 2 === 1);
```

- for (nombre de passages connus à l'avance)

```
$nbs = [2, 3, 4];  
  
for ($i=0; $i<count($nbs); $i++) {  
    echo $nbs[$i];  
}
```



- foreach (boucle sur tous les éléments d'un tableau)

```
$capitales = [  
    'France' => 'Paris',  
    'Allemagne' => 'Berlin',  
    'Espagne' => 'Madrid',  
];  
  
foreach ($capitales as $capitale) {  
    echo $capitale;  
}  
  
foreach ($capitales as $pays => $capitale) {  
    echo $pays;  
    echo $capitale;  
}
```



- Les fonctions permettent d'exécuter plusieurs fois le mêmes bloc, mais pas forcément à la suite
- Elles permettent de factoriser des traitements (évitant de dupliquer du code)

```
function hello($name) {  
    return "Bonjour $name";  
}  
  
echo hello('Romain'); // Bonjour Romain  
echo hello('Edouard'); // Bonjour Edouard
```

```
function addition($a, $b) {  
    return $a + $b;  
}  
  
echo addition(1, 2); // 3
```




- Recherche dans un tableau

```
$contacts = [
    ['prenom' => 'Steve', 'nom' => 'Jobs'],
    ['prenom' => 'Bill', 'nom' => 'Gates'],
];

function findByPrenom($array, $prenom) {
    foreach ($array as $elt) {
        if ($elt['prenom'] === $prenom) {
            return $elt;
        }
    }

    return null;
}

echo findByPrenom($contacts, 'Steve')['nom']; // Jobs
```



- Créer une fonction qui affiche le plus grand des 2 nombres en paramètres
- Créer une fonction qui affiche le plus grand des nombres du tableau reçu en paramètre
- Créer une fonction qui retourne les éléments d'un tableau dans l'ordre inverse
- Créer une fonction qui inverse les clés et les valeurs d'un tableau associatif



PHP 7.0



- PHP 7.0 est une nouvelle version majeure de PHP qui introduit certains changements rétro-incompatibles
- Il n'y a pas eu de PHP 6 car il était prévu une version 6 au début des années 2000 et des livres ont été écrit sur le sujet alors qu'il n'est jamais sorti (remplacé par PHP 5.3)
- Une réécriture du Zend Engine améliore significativement les performances et la consommation mémoire
- On retrouve de nouvelles syntaxes, fonctions, classes et constantes
- Pour connaître toutes les nouveautés :
<https://www.php.net/manual/en/migration70.php>

PHP 7.0 - Scalar type hint



- Depuis PHP 5 il est possible de typer statiquement les paramètres de fonctions avec des classes, des interfaces, des traits, des tableaux et des fonctions :

```
class User {  
    protected $name = 'Romain';  
    public function hello() {  
        return "Hello $this->name";  
    }  
}
```

```
function showUser(User $user) {  
    echo $user->hello(); // Romain  
}
```

```
$romain = new User();  
showUser($romain);
```

PHP 7.0 - Scalar type hint



- Depuis PHP 5 il est possible de typer statiquement les paramètres de fonctions avec des classes, des interfaces, des traits, des tableaux et des fonctions :

```
function showArray(array $list) {  
    foreach ($list as $elt) {  
        echo $elt;  
    }  
}
```

```
showArray([ 'a', 'b', 'c' ]);
```

PHP 7.0 - Scalar type hint



- Depuis PHP 5 il est possible de typer statiquement les paramètres de fonctions avec des classes, des interfaces, des traits, des tableaux et des fonctions :

```
function callFunction(callable $fct) {  
    $fct();  
}
```

```
function hello() {  
    echo 'Hello';  
}
```

```
callFunction('hello');
```

PHP 7.0 - Scalar type hint



- Avec PHP 7.0 il devient possible d'utiliser les types scalaires en PHP (bool, int, float, string)

```
function bonjour(string $name) {  
    echo "Hello $name";  
}
```

```
bonjour('hello');
```


PHP 7.0 - Return type



- Il est également possible de spécifier statiquement les types de retour

```
function bonjour(string $name): string {  
    return "Hello $name";  
}
```

```
echo bonjour('hello');
```

PHP 7.0 - Nullish Coalescing Operator



- Pour vérifier le contenu d'un tableau associatif en PHP 5

```
$firstName = '';  
  
if (isset($_POST['first_name'])) {  
    $firstName = $_POST['first_name'];  
}
```

- Ou en une ligne avec l'opérateur ternaire :

```
$firstName = isset($_POST['first_name']) ? $_POST['first_name'] : '';
```

- Avec PHP 7.0 on peut utiliser un nouvel opérateur ??

```
$firstName = $_POST['first_name'] ?? '';
```

PHP 7.0 - Spaceship operator



- Trier un tableau avec une fonction de comparaison

```
function compareContacts($a, $b)
{
    if ($a['age'] == $b['age']) {
        return 0;
    }
    return ($a['age'] < $b['age']) ? -1 : 1;
}
```

```
usort($contacts, 'compareContacts');
```

```
print_r($contacts);
```

```
//[
//    ['name' => 'Jean', 'age' => 18],
//    ['name' => 'Eric', 'age' => 20],
//];
```

- La fonction de comparaison doit retourner -1, 0 ou 1

PHP 7.0 - Spaceship operator



- Le spaceship operator remplace retourne déjà -1, 0 ou 1 :

```
function compareContacts($a, $b) {  
    return $a['age'] <=> $b['age'];  
}  
  
usort($contacts, 'compareContacts');  
  
print_r($contacts);  
//[  
//    ['name' => 'Jean', 'age' => 18],  
//    ['name' => 'Eric', 'age' => 20],  
//];
```

PHP 7.0 - Anonymous class



- PHP 7.0 permet l'utilisation de classes anonymes

```
<?php
interface Logger {
    public function log(string $msg);
}

function useLogger(Logger $logger) {
    $logger->log( 'Message' );
}

useLogger(new class implements Logger {
    public function log(string $msg) {
        echo $msg;
    }
});
// Message
```

PHP 7.0 - Group use statement



- Des use multiples

```
use Some\Framework\ClassA;  
use Some\Framework\ClassB;  
use Some\Framework\ClassC as C;
```

- Peut désormais s'écrire

```
use Some\Framework\{ClassA, ClassB, ClassC as C};
```



PHP 7.1

PHP 7.1 - Nullable types



- Il devient possible de préciser qu'un paramètre d'entrée ou de retour accepte la valeur null en utilisant le symbole ?

```
<?php

class Contact {
    /** @var Company|null */
    protected $company = null;

    public function getCompany(): ?Company
    {
        return $this->company;
    }

    public function setCompany(?Company $company): Contact
    {
        $this->company = $company;
        return $this;
    }
}
```




- Le type de retour void a été introduit pour les fonctions qui ne retournent rien :
 - pas de return
 - return sans valeur

```
<?php
```

```
function swap(&$left, &$right): void
{
    if ($left === $right) {
        return;
    }

    $tmp = $left;
    $left = $right;
    $right = $tmp;
}
```

PHP 7.1 - Array destructuring



- Pour créer des variables qui reçoivent directement les éléments d'un tableau sans passer par une variable intermédiaire on peut désormais utiliser la destructuration

```
<?php
```

```
// with temporary variable
```

```
$tmp = explode(' ', 'Romain Bohdanowicz');
```

```
$firstName = $tmp[0];
```

```
$lastName = $tmp[1];
```

```
var_dump($firstName, $lastName);
```

```
// with list
```

```
list($firstName, $lastName) = explode(' ', 'Romain Bohdanowicz');
```

```
var_dump($firstName, $lastName);
```

```
// with array destructuring
```

```
[$firstName, $lastName] = explode(' ', 'Romain Bohdanowicz');
```

```
var_dump($firstName, $lastName);
```

PHP 7.1 - Keys in list and destructuring



- On peut également déstructurer des tableaux associatifs :

```
class User {  
    public $username = 'romain';  
    public $isActive = true;  
}
```

```
$user = new User();  
list('username' => $username, 'isActive' => $isActive) = get_object_vars($user);  
var_dump($username, $isActive);
```

```
['username' => $username, 'isActive' => $isActive] = get_object_vars($user);  
var_dump($username, $isActive);
```

PHP 7.1 - class constant visibility



- Les constantes d'une classe peuvent désormais avoir une visibilité public, private ou protected

```
<?php  
  
class ConstDemo  
{  
    const PUBLIC_CONST_A = 1;  
    public const PUBLIC_CONST_B = 2;  
    protected const PROTECTED_CONST = 3;  
    private const PRIVATE_CONST = 4;  
}
```

PHP 7.1 - Multi catch exception handling



- Il devient possible d'intercepter plusieurs types d'exception dans un même bloc avec le symbole |

```
<?php
try {
    // some code
} catch (FirstException | SecondException $e) {
    // handle first and second exceptions
}
```

PHP 7.1 - Negative string offset



- On peut utiliser des indices négatifs pour indiquer que l'on souhaite partir de la fin d'une chaîne de caractères :

```
<?php  
var_dump( "Romain" [-2] ); // i
```



PHP 7.2

PHP 7.2 - object type



- Un nouveau type object est introduit

```
<?php

function test(object $obj) : object
{
    return new SplQueue();
}

test(new StdClass());
```


PHP 7.2 - Password hashing with Argon2



- Les mots de passe peuvent désormais être encodé avec l'algorithme Argon2 (gagnant de la Password Hashing Competition en juillet 2015)

```
<?php
```

```
$hash = password_hash('somepassword', PASSWORD_ARGON2I, [  
    'memory_cost' => 2048,  
    'time_cost'   => 4,  
    'threads'     => 3  
]);
```

PHP 7.2 - Trailing comma for grouped namespace



- Les namespaces groupés peuvent se terminer par une virgule (améliore les diffs et copier/coller)

```
<?php
```

```
use Foo\Bar\  
    Foo,  
    Bar,  
    Baz,  
};
```



PHP 7.3

PHP 7.3 - More Flexible Heredoc / Nowdoc Syntax



- Les syntaxes Heredoc et Nowdoc deviennent moins strictes :
 - le marqueur de fin (ici SQL) peut désormais être identé
 - le marqueur de fin (ici SQL) n'a plus à être suivi d'un retour à la ligne (ici le commentaire est possible)

```
<?php

class SomeClass {
    public function containingSomeSQL() {
        return <<<SQL
            SELECT firstName, lastName, email, phone
            FROM users
            WHERE id = :id
        SQL; // commentaire
    }
}
```

PHP 7.3 - Password hashing with Argon2



- Les mots de passe peuvent utiliser la variante Argon2id

```
<?php

$hash = password_hash('somepassword', PASSWORD_ARGON2ID, [
    'memory_cost' => 2048,
    'time_cost'   => 4,
    'threads'     => 3
]);
```

PHP 7.3 - Trailing comma in function call



- Les appels de fonction peuvent se terminer par une virgule

```
<?php
```

```
var_dump(  
    "Lorem ipsum dolor sit amet consectetur adipisicing",  
    "Vel consequuntur ullam debitis nam iusto deserunt",  
    "Provident, hic, temporibus libero dolorem voluptas",  
);
```



PHP 7.4

PHP 7.4 - Typed properties



- Les propriétés peuvent désormais être typées

```
<?php  
  
class User {  
    public int $id;  
    public string $name;  
}
```




- Les fonctions fléchées sont une syntaxe courte pour déclarer des fonctions, Exemple d'une fonction somme :
`fn($a, $b) => $a + $b`
- Elles capturent automatiquement les variables de la portée où elle sont définies (comme le use des closures)

```
<?php
```

```
$factor = 10;
```

```
$nums = array_map(fn($n) => $n * $factor, [1, 2, 3, 4]);
```

PHP 7.4 - Unpacking inside array



- L'utilisation de l'opérateur `...` devant un tableau le transforme en son contenu (ici `'apple', 'pear'`)

```
<?php
$parts = ['apple', 'pear'];
$fruits = ['banana', 'orange', ...$parts, 'watermelon'];
// ['banana', 'orange', 'apple', 'pear', 'watermelon'];
```

PHP 7.4 - Numeric literal separator



- Les nombres peuvent être séparé en utilisant des underscores

```
<?php  
$million = 1_000_000;
```

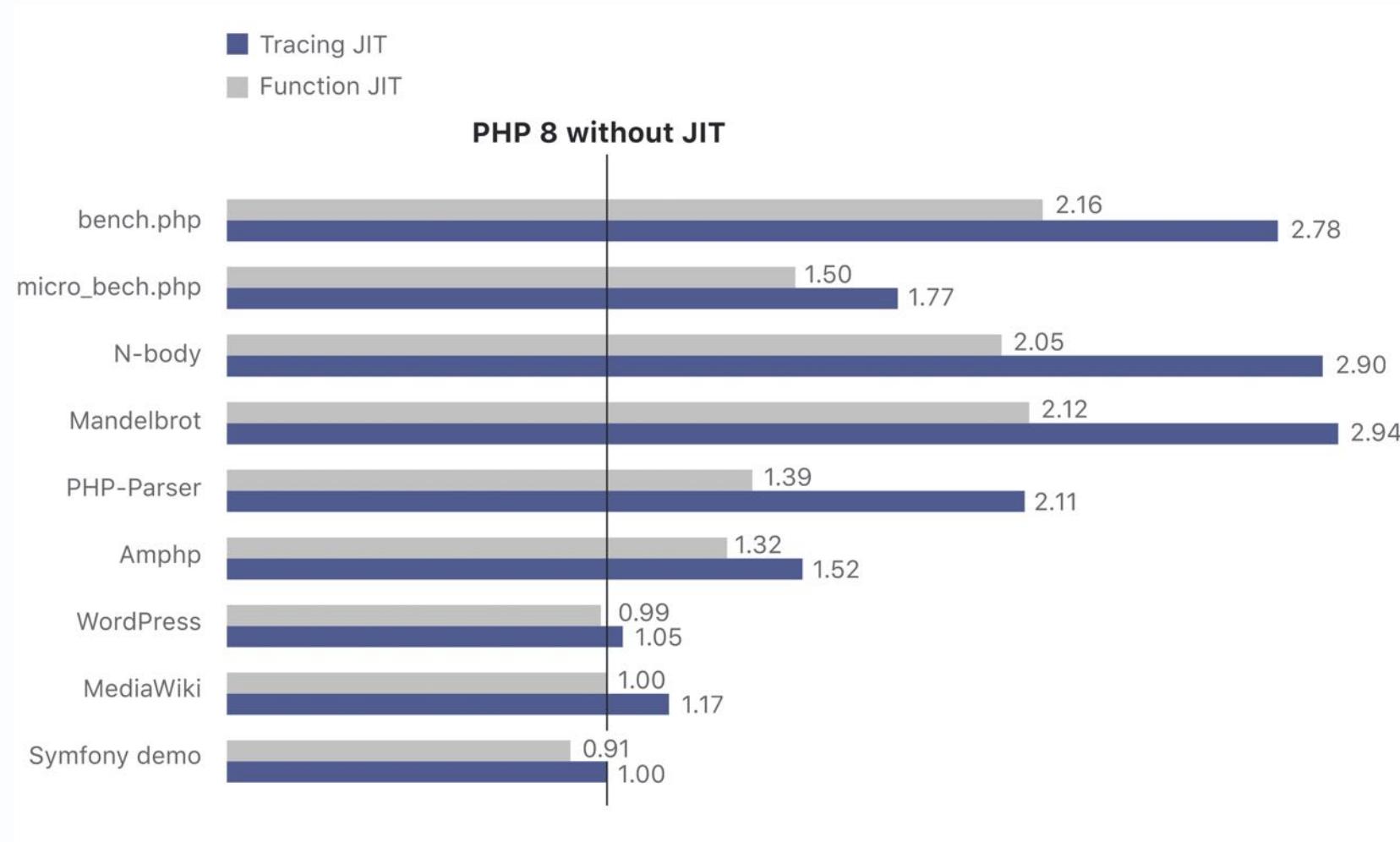


PHP 8.0

PHP 8.0 - Compilation JIT



- PHP 8 introduit la compilation Just In Time, optimisation apportée au niveau du Zend Engine qui compile en code natif lorsque le code est réutilisé (fonctions, boucles, ...)



PHP 8.0 - Named arguments



- Il est désormais possible de passer des paramètres à une fonction en utilisant leur nom plutôt que leur position

```
var_dump(str_replace('jour', 'soir', 'Bonjour')); // Bonsoir
```



```
var_dump(str_replace(subject: 'Bonjour', search: 'jour', replace: 'soir')); // Bonsoir
```



- Les bibliothèques Doctrine et phpDocumentator avaient introduit les annotations en PHP
- Les annotations permettent d'exécuter du code de configuration de manière déclarative
- Les annotations étaient écrites sous forme de commentaires qui étaient parsés par introspection ce qui n'était pas très performant
- PHP 8.0 introduit les attributs qui est une syntaxe native pour remplacer ces annotations

```
class PostsController
{
    /**
     * @Route("/api/posts/{id}", methods={"GET"})
     */
    public function get($id) { /* ... */ }
}
```



```
class PostsController
{
    #[Route("/api/posts/{id}", methods: ["GET"])]
    public function get($id) { /* ... */ }
}
```

PHP 8.0 - Constructor property promotion



- En utilisant une visibilité (public, private, protected) au niveau du constructeur on peut déclarer et affecter des paramètres à des propriétés de la classe

```
class Logger {  
    protected Writer $writer;  
  
    public function __construct(Writer $writer)  
    {  
        $this->writer = $writer;  
    }  
}
```



```
class Logger  
{  
    public function __construct(protected Writer $writer)  
    {  
    }  
}
```


PHP 8.0 - Union type



- Il est désormais possible de définir plusieurs types possible par arguments ou propriété

```
function square(int|float $nb) {  
    return $nb * $nb;  
}
```

PHP 8.0 - Nullsafe operator



- L'opérateur nullsafe permet de ne pas exécuter la suite d'une expression si la valeur qui le précède vaut null

```
$country = null;

if ($session !== null) {
    $user = $session->user;

    if ($user !== null) {
        $address = $user->getAddress();

        if ($address !== null) {
            $country = $address->country;
        }
    }
}
```



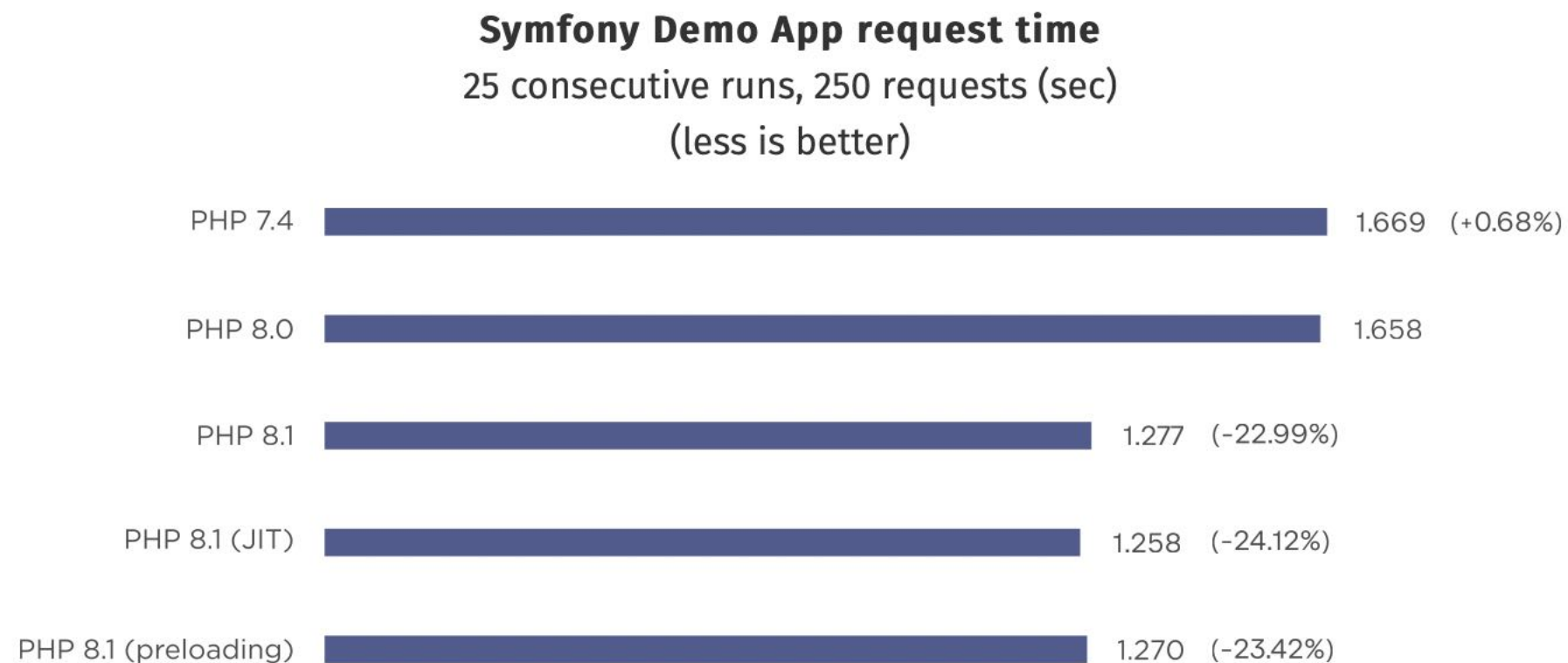
```
$country = $session?->user?->getAddress()?->country;
```



PHP 8.1



- Comme régulièrement avec les mises à jour de PHP, les performances sont améliorées





- Les enum sont introduits en PHP pour remplacer les classes ne contenant que des constantes

```
<?php
enum Status
{
    case Draft;
    case Published;
    case Archived;
}
function acceptStatus(Status $status) {
    if (Status::Published) {
        // ...
    }
}
```

PHP 8.1 - Read only properties



- Les propriétés peuvent être en lecture seule avec le mot clé readonly
- Elles ne pourront recevoir qu'une valeur initiale

```
<?php
class BlogData
{
    public readonly Status $status;

    public function __construct(Status $status)
    {
        $this->status = $status;
    }
}
```



PHP CLI



- Ce n'est pas l'utilisation la plus répandue mais PHP permet de créer des programmes en ligne de commande
- Des programmes comme composer, phpcs, symfony/cli sont écrits en PHP
- Pour démarrer un programme en ligne de commande :
`php [CHEMIN-VERS-LE-FICHIER.php]`
- On peut également directement exécuter du code PHP avec l'option -r
`php -r 'print_r(get_defined_constants());'`

PHP CLI - echo



- echo écrit dans le Terminal

```
<?php  
echo 'Hello\n';
```

A screenshot of a macOS terminal window titled "PHP-CLI — romain@MacBook-Pr...". The terminal shows a command prompt with a green arrow icon, followed by the command "PHP-CLI php hello.php". The output "Hello" is displayed on the next line. Below the output, the prompt "PHP-CLI" is shown with a grey cursor bar.

```
PHP-CLI — romain@MacBook-Pr...  
[→ PHP-CLI php hello.php]  
Hello  
→ PHP-CLI █
```

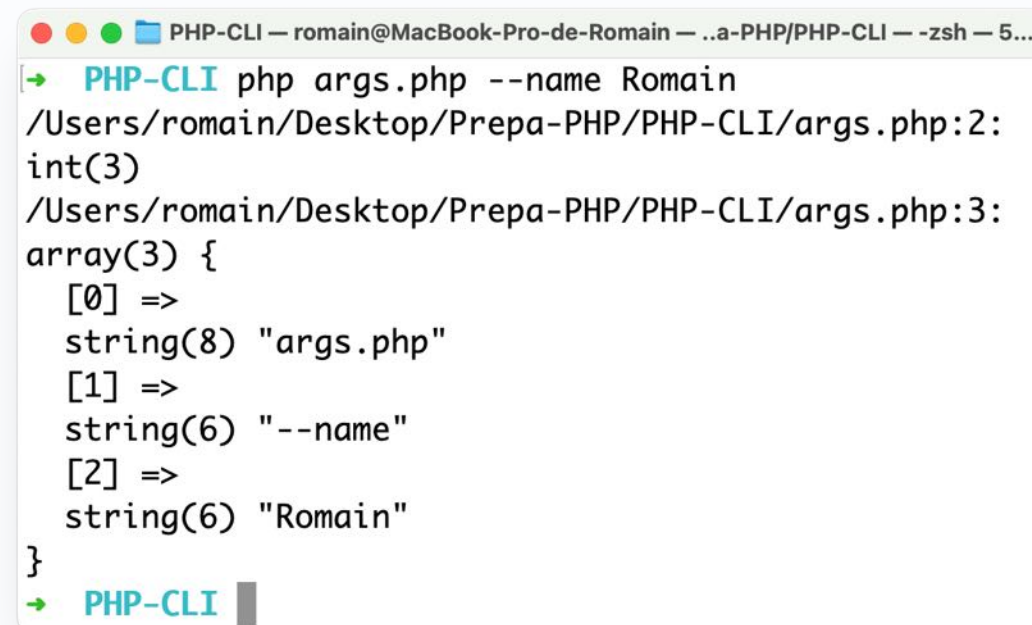
- Pour revenir à la ligne on utilise \n

PHP CLI - arguments



- Lorsqu'on appelle un programme PHP CLI on peut lui passer des arguments en ligne de commande
- Ex :
`php args.php --name Romain`
- Pour récupérer le nombre d'arguments on utilise la variable globale `$argc`
- Pour récupérer les arguments on utilise la variable `$argv`

```
<?php  
var_dump($argc);  
var_dump($argv);
```

A screenshot of a macOS terminal window titled "PHP-CLI — romain@MacBook-Pro-de-Romain — ..a-PHP/PHP-CLI — -zsh — 5...". The terminal shows the command `PHP-CLI php args.php --name Romain` being executed. The output is as follows:
/Users/romain/Desktop/Prepa-PHP/PHP-CLI/args.php:2:
int(3)
/Users/romain/Desktop/Prepa-PHP/PHP-CLI/args.php:3:
array(3) {
 [0] =>
 string(8) "args.php"
 [1] =>
 string(6) "--name"
 [2] =>
 string(6) "Romain"
}
The prompt `PHP-CLI` is visible at the bottom.

PHP CLI - Shebang



- PHP supporte le Shebang, un entête de fichier qui commence par `#!` et qui rend le programme autoexécutable
- Il faudra donner les droits d'exécution

```
#!/usr/bin/env php  
<?php  
echo "Hello\n";
```

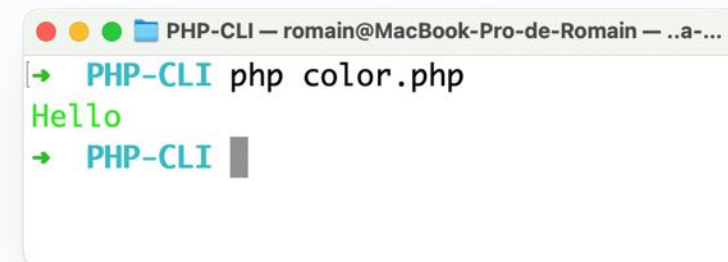
A screenshot of a macOS terminal window titled "PHP-CLI — romain@MacBook-Pro-de-Romain — ..a-...". The terminal shows the following commands and output:
→ PHP-CLI chmod +x shebang.php
→ PHP-CLI ./shebang.php
Hello
→ PHP-CLI █



- Certains Terminaux supporte la couleur, il suffit d'utiliser les codes correspondants :
<https://www.codeproject.com/Articles/5329247/How-to-change-text-color-in-a-Linux-terminal>

```
<?php
function green(string $msg) {
    return "\033[92m$msg\033[0m";
}

echo green("Hello\n");
```



- Pour simplifier l'utilisation de la mise en forme :
<https://climate.thephpleague.com/>
<https://github.com/php-parallel-lint/PHP-Console-Color>



- Symfony Console :

<https://symfony.com/doc/current/components/console.html>

```
<?php
use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;

class HelloCommand extends Command
{
    public function configure()
    {
        $this->setName('hello');
    }

    public function execute(InputInterface $input, OutputInterface $output)
    {
        $output->writeln('Hello');
        return Command::SUCCESS;
    }
}
```

PHP CLI - Bibliothèques



- Symfony Console :

<https://symfony.com/doc/current/components/console.html>

```
#!/usr/bin/env php
<?php
require __DIR__.'/vendor/autoload.php';

use Symfony\Component\Console\Application;

$application = new Application();

$application->add(new HelloCommand());

$application->run();
```

A screenshot of a macOS terminal window titled 'PHP-CLI — romain@MacBook-Pro-de-Romain — ..a-...'. The terminal shows a command prompt with a green arrow icon, followed by the command 'PHP-CLI ./my-app hello'. The output 'Hello' is displayed on the next line. A second prompt with a green arrow icon and 'PHP-CLI' is shown on the following line, with a grey cursor block indicating the next input.

```
→ PHP-CLI ./my-app hello
Hello
→ PHP-CLI
```



Composer

Composer - Introduction



- Utilité de composer

Composer est un système de gestion de dépendances dans un projet PHP.

Il permet de télécharger et de mettre à jour automatiquement les bibliothèques utilisés dans nos projets ainsi que les bibliothèques dont elles dépendent en cascade.

- Equivalents dans d'autres langages :

Ruby : gem, bundler

Python : pip

JavaScript : npm / Yarn

Objective-C : CocoaPods

- Prérequis :

PHP 5.3.2+

Pour certains packages : git, svn ou mercurial

Composer - Installation



- Avec curl
`curl -sS https://getcomposer.org/installer | php`
- Avec PHP
`php -r "readfile('https://getcomposer.org/installer');" | php`
- Téléchargement Manuel
Télécharger le « latest snapshot » sur <https://getcomposer.org/download/>

Composer peut s'installer localement (dans un projet) ou globalement (utile pour ne pas avoir à reconfigurer notre IDE à chaque projet).

- Utilisation d'un proxy
 - Définir une variable d'environnement `http_proxy`
`setenv http_proxy http://login:pass@hote_du_proxy`
 - Eventuellement configurer les variables d'environnement suivantes à `false`
`setenv HTTP_PROXY_REQUEST_FULLURI false`
`setenv HTTPS_PROXY_REQUEST_FULLURI false`



- Pour obtenir de l'aide :
 - Utiliser l'option --help
`composer [votre commande] --help`
 - Documentation officielle
<https://getcomposer.org/doc/>
 - Cheatsheet
<http://composer.json.jolicode.com>
 - Awesome Composer
<https://github.com/jakoch/awesome-composer>

Composer - Nouveau projet



▸ Pour créer un nouveau projet avec composer on peut :

- créer un fichier composer.json manuellement

```
{}
```

- `composer init` (puis répondre aux questions)

A screenshot of a terminal window titled 'hello-composer — composer init — composer — php • php /usr/local/bin/composer init — 80x20'. The terminal shows the execution of 'mkdir hello-composer' and 'composer init'. A blue banner displays 'Welcome to the Composer config generator'. The terminal then prompts for package name, description, author, minimum stability, package type, and license, with the following values entered: 'romain/hello-composer', an empty description, 'Romain Bohdanowicz <romain.bohdanowicz@gmail.com>', an empty minimum stability, 'project', and 'MIT'. It then asks to define dependencies and if they should be defined interactively, with 'yes' entered.

```
hello-composer — composer init — composer — php • php /usr/local/bin/composer init — 80x20
→ Desktop mkdir hello-composer && $_
→ hello-composer composer init

Welcome to the Composer config generator

This command will guide you through creating your composer.json config.

Package name (<vendor>/<name>) [romain/hello-composer]:
Description []:
Author [Romain Bohdanowicz <romain.bohdanowicz@gmail.com>, n to skip]:
Minimum Stability []:
Package Type (e.g. library, project, metapackage, composer-plugin) []: project
License []: MIT

Define your dependencies.

Would you like to define your dependencies (require) interactively [yes]? █
```

Composer - Nouveau projet



- On peut également créer un projet à partir d'un squelette d'application avec la commande `create-project`
- Exemple :
`composer create-project symfony/skeleton`

Composer - Ajouter des dépendances

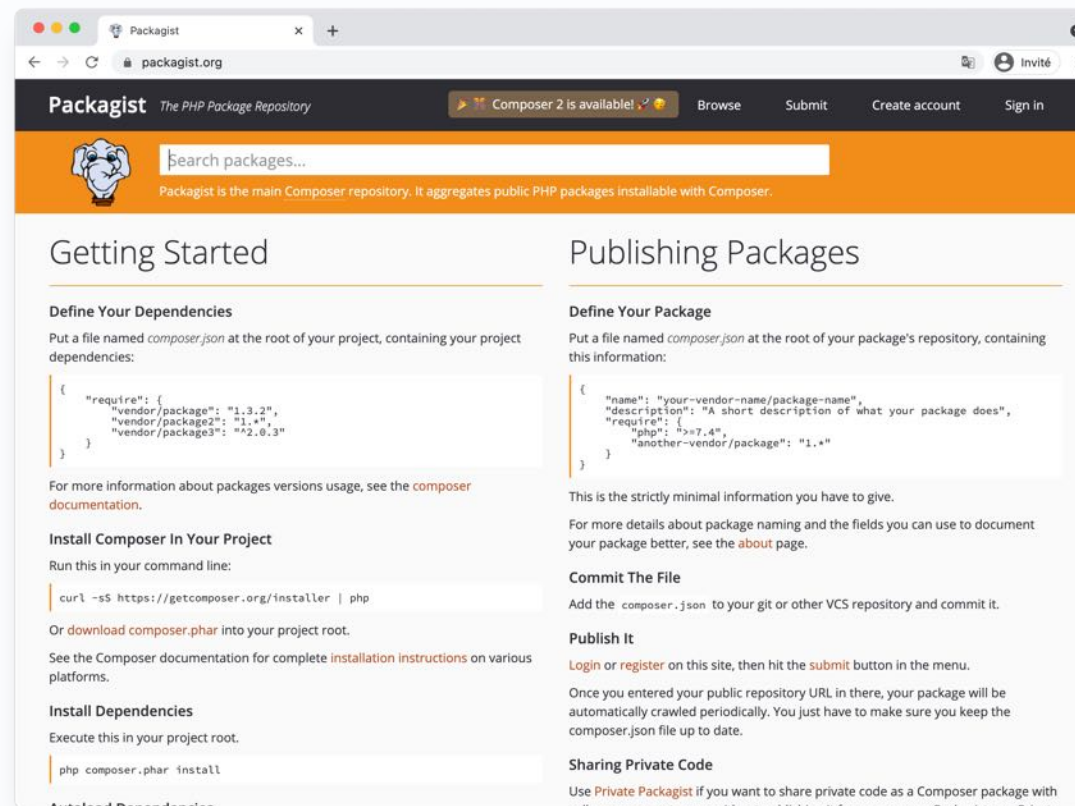


- Pour ajouter de nouvelles dépendances on utilise la commande
`composer require [nom-du-paquet]`
- Ou bien pour des dépendances de dev à ne pas installer en production
`composer require --dev [nom-du-paquet]`
- Pour simuler une installation
`composer require --dry-run [nom-du-paquet]`

Composer - Où trouver des dépendances



- La principale source de paquet est Packagist
<https://packagist.org>



- D'autres repositories existent :
<https://packages.drupal.org/>
<https://wpackagist.org>

...

Composer - Mettre à jour



- `composer self-update`
Met à jour automatiquement composer lui-même
- `composer update`
Met à jour automatiquement les bibliothèques en tenant compte des versions souhaitées par le fichier *composer.json*

Composer - Reprendre un projet



- `composer install (--no-dev)`
Installe les dépendances présentent dans le fichier *composer.lock* ou si inexistant dans le fichier *composer.json*
- `composer dump-autoload -o`
Optimise l'autochargeur de classes en éditant le fichier *vendor/composer/autoload_classmap.php*

Composer - Le fichier composer.json



- "require": {}, dépendances générales du projet
"require-dev": {}, dépendances dans un environnement de développement
- Packagist : annuaire contenant les noms et versions des bibliothèques à utiliser pour composer :
<https://packagist.org>
- Plus de détails et autres options du le fichier composer.json :
<https://getcomposer.org/doc/04-schema.md>
<http://composer.json.jolicode.com>

```
{
  "name": "AddressBookZF2",
  "description": "Un carnet d'adresse utilisant Zend Framework 2",
  "license": "BSD-3-Clause",
  "authors": [{
    "name": "Romain Bohdanowicz",
    "email": "romain.bohdanowicz@gmail.com"
  }],
  "require": {
    "php": ">=5.3.3",
    "zendframework/zendframework": "2.3.*",
    "doctrine/doctrine-orm-module": "0.*"
  },
  "require-dev": {
    "zendframework/zftool": "dev-master",
    "zendframework/zend-developer-tools": "dev-master"
  }
}
```



PHP Objet

PHP Objet - Introduction



- Programmation orientée objet apparue dans les années 70 avec le langage Smalltalk
- Application de principes de tous les jours à la programmation
- Paradigme de programmation (style de programmation)
- Architecture l'application autour des données plutôt que des traitements (programmation procédurale)
- Facilite les tests, la réutilisation de code, le travail à plusieurs

PHP Objet - Classe vs Objet



- Classe
 - Un concept (Ordinateur)
 - Un type
 - Permet la création d'objets (un moule)
- Objet
 - Représentation en mémoire d'une classe (instance de classe)
 - Plusieurs objets possibles pour une classe

PHP Objet - Classe et Type



- Une classe permet la déclaration d'un nouveau type complexe
- Peut contenir plusieurs valeurs stockées dans des (nom au choix) :
 - propriétés
 - attributs
 - champs
- Pour accéder au contenu d'un objet, on utilise l'opérateur -> (un tiret - suivi du symbole supérieur >)

```
<?php

class Computer
{
    public $brand;
    public $model;
}

$macbook = new Computer;
$macbook->brand = 'Apple';

echo "This MacBook was built by $macbook->brand";
```



- Une classe peut également contenir des fonctions appelées « méthodes »
- La pseudo-variable *\$this* est une référence interne à l'objet (permet d'accéder aux autres propriétés et méthodes de l'objet sur lequel la méthode d'origine a été appelée)

```
<?php

class Computer
{
    public $brand;
    public $model;

    public function showInfos() {
        echo "This $this->model was built by $this->brand";
    }
}

$macbook = new Computer;
$macbook->brand = 'Apple';
$macbook->model = 'MacBook';
$macbook->showInfos();
```



- Chaque membre (propriété, méthode) peut se déclarer avec une visibilité
 - public : accessible dès lors que l'objet est accessible
 - protected : accessible depuis la classe, les classe ascendante ou descendante (voir le chapitre sur l'Héritage)
 - private : uniquement à l'intérieur de la classe

PHP Objet - Principe d'encapsulation



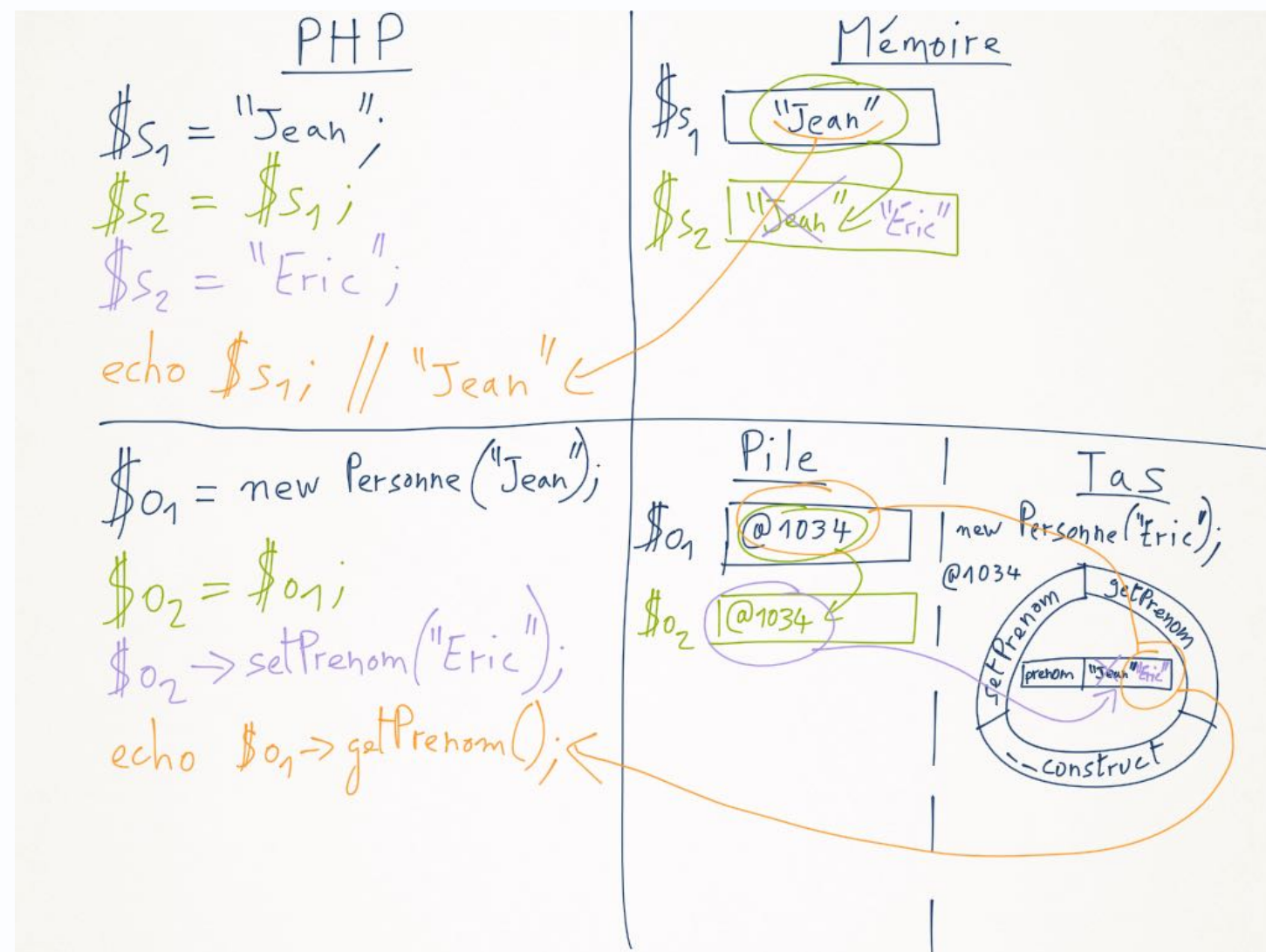
- Bonne pratique : ne jamais rendre les propriétés publiques
- Un objet peut représenter un concept complexe, il faut qu'à l'utilisation un autre développeur voit un API le plus simple possible
- Pour accéder aux propriétés on écrit (ou on génère avec le bon IDE) des méthodes appelées accepteurs ou getter/setter (get[Propriété] en lecture et set[Propriété] en écriture)
- Masquer les propriétés permet également de choisir si elles sont accessibles en lecture et/ou écriture

```
<?php  
  
class Computer  
{  
    protected $brand;  
  
    public function getBrand(): string  
    {  
        return $this->brand;  
    }  
  
    public function setBrand(string $brand): Computer  
    {  
        $this->brand = $brand;  
        return $this;  
    }  
}
```


PHP Objet - Référence



- la différence des types scalaires (*int*, *boolean*, *float*, *string*, *array*...) les objets se manipulent au travers de références
- une référence permet de retrouver l'objet en mémoire
- seule les opérateurs *new* et *clone* permettent de construire de nouveaux objets



PHP Objet - Namespace



- Pour éviter les conflits sur les noms de classe, PHP 5.3 introduit la notions de namespace
- Un namespace est un dossier virtuel qui vient préfixer le nom de la classe

```
<?php  
  
namespace FormationTech\Model;  
  
class Contact  
{  
  
}
```



- A l'utilisation il faut utiliser le Fully Qualified Class Name (FQCN ou FQN)

```
<?php  
  
require_once './Contact.php';  
  
$romain = new FormationTech\Model\Contact();
```

- On peut également raccourcir les lignes en utilisant au début de chaque fichier le mot clé use, suivi du FQCN

```
<?php  
  
use FormationTech\Model\Contact;  
  
require_once './Contact.php';  
  
$romain = new Contact();
```



- On peut également créer un Alias (nécessaire si 2 classes portent le même nom)

```
<?php  
  
use FormationTech\Model\Contact as ContactModel;  
  
require_once './Contact.php';  
  
$romain = new ContactModel();
```

- Les namespaces peuvent également être aliassés

```
<?php  
  
use FormationTech\Model as M;  
  
require_once './Contact.php';  
  
$romain = new M\Contact();
```



- Pour éviter d'inclure chaque classe avec un require, on peut créer un autoloader
- Il faut alors avoir une corrélation entre le nom la classe et le chemin sur le disque
- Historiquement chaque bibliothèque vient avec sa propre convention donc sont autoloader (problématique puisque certains s'exécuteront inutilement)
- Le PHP Framework Interop Group (PHP-FIG) a été créé pour que les développeurs de bibliothèques PHP s'accordent sur des normes
- PSR-0 (dépréciée) puis PSR-4 sont des normes d'autochargement
- Dans PSR-4 il faut associer un namespace préfixe à un répertoire sur le disque, à chaque namespace suivant doit correspondre une dossier, à chaque classe un fichier suffixé par .php
- PSR-4: Autoloader
<https://www.php-fig.org/psr/psr-4/>

PHP Objet - Autochargement



- Exemple d'autoloader PSR-4 (basique)

```
<?php

$prefixes = [
    'FormationTech\\' => __DIR__ . '/lib/',
];

spl_autoload_register(function ($fqcn) use ($prefixes) {

    $path = strtr($fqcn, $prefixes) . '.php';
    $path = strtr($path, '\\', DIRECTORY_SEPARATOR);

    @include_once $path;
});
```

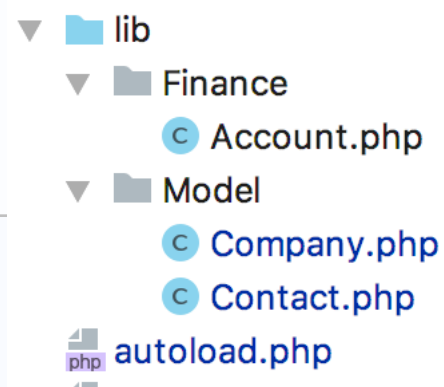
- A l'utilisation

```
<?php

use FormationTech\Model\Contact;

require_once 'autoload.php';

$romain = new Contact();
```





- Pour lier des objets entre eux on utilise des associations
- Il suffit en PHP de stocker dans un objet la référence vers un autre objet
- Si plusieurs objets sont liés, les mettre dans un tableau, ou une classe de Collection comme *\SplObjectStorage*
- Si l'association n'est possible que dans un sens (classe A → classe B) on parle d'association unidirectionnelle
- Si elle est possible dans les 2 sens (classe A ↔ classe B) on parle d'association bidirectionnelle (ne le faire que si nécessaire)

```
<?php
namespace FormationTech\Model;

class Contact
{
    /** @var string */
    protected $firstName;

    /** @var string */
    protected $lastName;

    /** @var Company */
    protected $company;
}
```

```
<?php
namespace FormationTech\Model;

class Company
{
    /** @var string */
    protected $name;
}
```



- Pour réaliser l'association vers une référence on utilise un setter

```
<?php

namespace FormationTech\Model;

class Contact
{
    /** @var Company */
    protected $company;

    public function getCompany(): Company
    {
        return $this->company;
    }

    public function setCompany(Company $company): Contact
    {
        $this->company = $company;
        return $this;
    }
}
```

- TODO exemple utilisation



▸ Héritage

Une classe réutilise les membres d'une autre classe.

```
<?php
namespace Application\Entity;

class Contact
{
    protected $id;
    protected $prenom;
    protected $nom;
    protected $email;
    protected $telephone;

    /**
     * @var Societe
     */
    protected $societe;
}
```

```
<?php
namespace Application\Entity;

class Salarie extends Contact
{
    protected $salaire;
}
```



▸ Interface

La définition d'une liste de méthodes, implémenter une interface oblige à implémenter ses méthodes.

Dans ZF2 : leur nom se termine toujours par Interface

```
<?php
namespace Application\Entity;
use Zend\Stdlib\ArraySerializableInterface;
class Contact implements ArraySerializableInterface
{
    // ...

    public function exchangeArray(array $array)
    {
        foreach ($array as $key => $value) {
            if (property_exists($this, $key)) {
                $this->$key = $value;
            }
        }
    }

    public function getArrayCopy()
    {
        return get_object_vars($this);
    }
}
```

```
<?php
namespace Zend\Stdlib;
interface ArraySerializableInterface
{
    public function exchangeArray(array $array);
    public function getArrayCopy();
}
```



▸ Classe abstraite

Une classe qui n'a pour vocation à être utilisée qu'au travers d'un héritage. Peut également imposer l'implémentation de certaines méthodes comme une interface. Comme pour une interface, c'est la garantie que certaines méthodes seront bien présentes (programmation par contrat).

Dans ZF2 leur nom commence toujours par Abstract.

```
<?php
namespace Zend\Mvc\Controller;

abstract class AbstractActionController extends AbstractController
{
    // ...
}
```

```
<?php
namespace AddressBook\Controller;

use Zend\Mvc\Controller\AbstractActionController;

class ContactController extends AbstractActionController
{
    // ...
}
```



- PHP FIG
PHP Framework Interop Group, groupe de travail regroupant les principaux créateurs de frameworks/bibliothèques créant des normes PHP.
<http://www.php-fig.org/>
- PSR-1
PSR-0 + Basic Coding Standard
- PSR-2
PSR-1 + Coding Style Guide
- PSR-3
Logger Interface
- PSR-4
Improved Autoloading
- PSR-7
HTTP Message Interface



▸ Composition

Une composition est un type d'association forte entre 2 objet. La destruction d'un objet entrainerait la destruction de l'objet associé.

Exemple : Un objet Tasse est composée de Café

```
<?php
namespace EspressoComposition;

class Cafe
{
    protected $variete;
    protected $provenance;

    public function __construct($provenance, $variete)
    {
        $this->provenance = $provenance;
        $this->variete = $variete;
    }
}
```

```
<?php
namespace EspressoComposition;

class Tasse
{
    protected $contenu;

    public function __construct() {
        $this->contenu = new Cafe("Arabica", "Mexique");
    }
}
```

```
<?php
require_once 'autoload.php';

$tasseDeCafe = new \EspressoComposition\Tasse();
```

▸ Mauvaise Pratique

La composition est désormais considéré comme une mauvaise pratique. Premièrement la classe Tasse n'est très réutilisable, elle ne peut contenir que du café. De plus il n'est pas possible d'écrire d'écrire un test unitaire de Tasse, puisqu'il faudrait en même temps tester Café.

Globalement il faut essayer de proscrire l'utilisation de new il l'intérieur d'une classe (à l'exception des Values Objects (DateTime, ArrayObject, etc...))



▸ Solution

La solution est simple, pour éviter le new dans cette classe nous allons injecter la dépendance.

```
<?php
namespace EspressoInjection;

interface Liquide {
}
```

```
<?php
namespace EspressoInjection;

class Cafe implements Liquide
{
    protected $variete;
    protected $provenance;

    public function __construct($provenance, $variete)
    {
        $this->provenance = $provenance;
        $this->variete = $variete;
    }
}
```

```
<?php
namespace EspressoInjection;

class Tasse
{
    protected $contenu;

    public function __construct(Liquide $contenu) {
        $this->contenu = $contenu;
    }
}
```

```
<?php
require_once 'autoload.php';

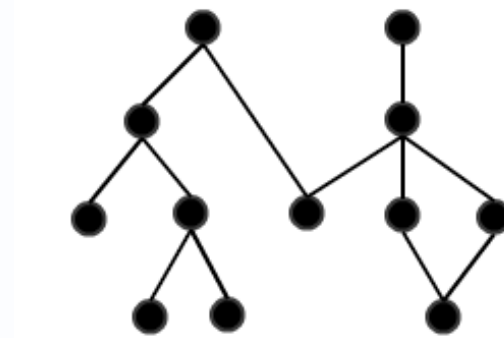
$cafe = new \EspressoInjection\Cafe();
$tasseDeCafe = new \EspressoInjection\Tasse($cafe);
```

- La classe Tasse peut désormais recevoir n'importe quel contenu qui implémente l'interface Liquide.



▸ Conteneur d'injection de dépendance (DIC)

Le problème lorsqu'on injecte les dépendances est qu'on peut parfois se retrouver avec des dépendances complexes :



▸ Dans ce cas il devient utile d'utiliser un conteneur d'injection de dépendance qu'on aura configuré au préalable. En PHP il existe quelques DIC connus :

- Pimple par Fabien Potencier
- Dice par Tom Butler
- PHP-DI par Matthieu Napoli
- Symfony\Container intégré Symfony2
- Zend\Di & Zend\ServiceManager intégrés ZF 2

▸ Zend\Di est très simple à configurer mais peu performant (utilise la Reflection), Zend\ServiceManager est le plus utilisé.

Documentation : <http://framework.zend.com/manual/current/en/modules/zend.service-manager.quick-start.html>



PDO



- PDO, PHP Data Object : inclut une compatibilité avec toutes les bases de données avec lesquelles PHP peut communiquer.

Extension PDO

Extension MySQL pour PDO

Extension PgSQL pour PDO

Extension Oracle pour PDO

Extension SQLite pour PDO

Grace à cette extension commune à tous les SGBD, j'utilise les mêmes fonctions, quelque soit le SGBD.

Ces extensions sont appelées des "drivers", il faut les installer pour pouvoir accéder aux SGBD voulus... le principe est le même que les codecs vidéo.



- PDO : trois pôles de fonctionnalités bien utiles

Fonctions d'accès : toutes les fonctions pour exécuter des requêtes et récupérer des données.

PDO

Mode transactionnel : assure l'intégrité des données et permet de revenir en arrière en cas de problème.

Requêtes préparées : augmente la vitesse d'exécution et sécurise les requêtes de modification, suppression, ajout.



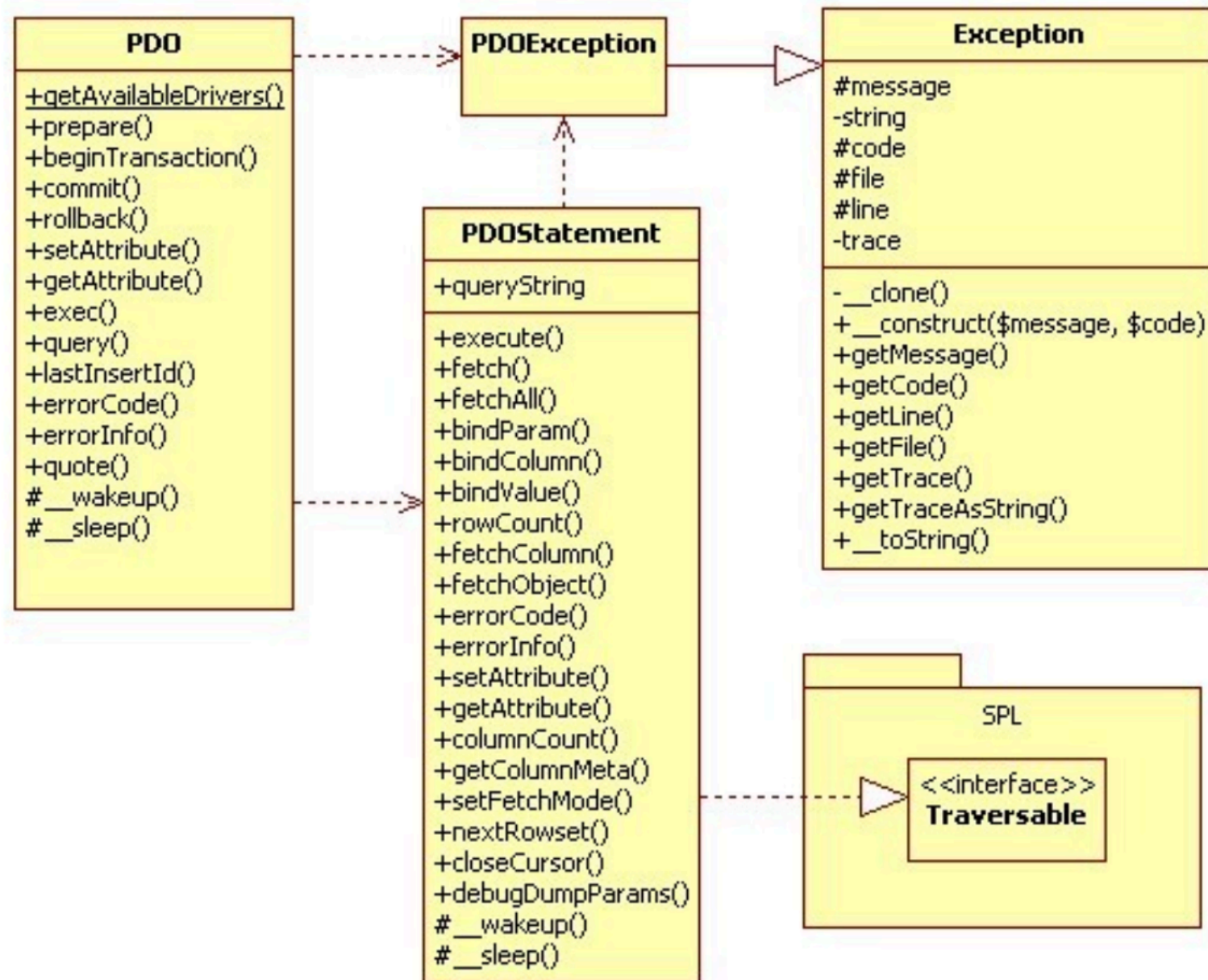
- PDO est fourni avec PHP depuis la version 5.1. C'est un système d'abstraction de l'accès à la base de données.
- Performances
 - Ecrit en C
 - Plus rapide qu'un système d'abstraction en PHP (ADODB, PEAR DB)
 - Aussi performant qu'une autre extension PHP
- Aptitudes
 - Requêtes, transactions, gestion des procédures stockées
 - Requêtes paramétrées, émulations de fonctionnalités Connexions à plusieurs SGBD via le même objet
 - MySQL, Oracle, MSSQL, PgSQL, Sqlite, Db2, Odbc, Sybase, FireBird

PDO - Utiliser votre base de données



- Cela se fait en 4 temps :
 - Création de l'instance PDO => Connexion
 - Envoi de la requête (ou préparation)
 - Récupération du résultat
 - Déconnexion éventuelle
- L'objet PDO n'est pas sérialisable

PDO - Structure des classes avec PDO



PDO - Exemple



```
<?php
try {
    $dsn = 'mysql:host=localhost;dbname=test';
    $user = 'root';
    $pass = 'qsdklkfd';
    $options = [];
    $dbh = new PDO($dsn, $user, $pass, $options);
    $result = $dbh->query('SELECT * from auteurs');
    while ($row = $result->fetch()) {
        print_r($row);
    }
} catch (PDOException $e) {
    echo $e->getMessage() . "<br/>";
}
```



- DSN (Data Source Name) pour MySQL
 - **mysql:** → Préfixe DSN.
 - **host** → l'hôte sur lequel le SGBD se situe
 - **port (optionnel)** → le numéro de port que le SGBD est en train d'écouter.
 - **dbname (optionnel)** → nom de la DB
 - **unix_socket (optionnel)** → le socket unix MySQL (ne devrait pas être utilisé avec host ou port)
- Exemple
`mysql:host=localhost;dbname=testdb`



- `mysql:host=name;dbname=dbname`
- `pgsql:host=name dbname=dbname`
- `odbc:odbc_dsn`
- `oci:dbname=dbname;charset=charset`
- `sqlite:/path/to/file`
- Dans le `php.ini` :
 - `pdo.dsn.mondsn="sqlite:/path/to/name.db"`
- Puis :
 - `$dbh = new PDO("mondsn");`



- Avec PDO, vous pouvez définir le déclencheur d'erreurs.
- Vous pouvez demander à PDO :
 - de ne pas afficher les erreurs (mode silencieux) (défaut) ;
 - d'utiliser le mode d'erreur classique (lance une erreur PHP de niveau E_WARNING) ;
 - d'utiliser les exceptions (recommandé) Par défaut, PDO utilise le mode silencieux
- Sauf le constructeur qui lui renvoie une Exception si problème
- Les erreurs sont stockées et consultables en faisant appel aux méthodes `errorCode()` et `errorInfo()`.



- On préférera profiter des exceptions, une exception PDOException est lancée.
- Ainsi vous pouvez décider de la meilleure façon de gérer l'erreur (ou de ne pas la gérer, ce qui arrêtera l'exécution).

```
<?php
try {
    $dsn = 'mysql:host=localhost;dbname=test';
    $user = 'root';
    $pass = 'qsdklkfd';
    $dbh = new PDO($dsn, $user, $pass);
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
catch (PDOException $e) {
    echo $e->getMessage() . "<br/>";
}
```

PDO - Fermer une connexion



- La connexion est active tant que l'objet PDO l'est.
- Pour clore la connexion, vous devez détruire toutes les références vers l'objet PDO.
- Vous pouvez faire cela en assignant NULL à la variable représentant l'objet ou en utilisant la fonction unset().
- Si vous ne le faites pas explicitement, PHP fermera automatiquement la connexion lorsque le script arrivera à la fin.

PDO - Effectuer une requête



- Pour envoyer une requête au serveur on peut utiliser deux méthodes de l'objet PDO : `exec()` et `query()`.
- La méthode `exec()` permet d'exécuter une requête mais ne renvoie que le nombre de lignes modifiées
 - On s'en servira généralement pour faire des insertions, des modifications ou des suppressions.
 - INSERT, UPDATE, DELETE, CREATE, ALTER
- La méthode `query()` permet de récupérer des données. Elle renvoie une instance de la classe `PDOStatement`.
 - SELECT, EXPLAIN, SHOW, DESC
- Dans le cas où la requête ne fonctionne pas les méthodes `query()` et `exec()` renvoient `FALSE`.

PDO - Lire enregistrements, PDOStatement



- Après un `query()`, on souhaite exploiter les résultats c'est l'objet de type `PDOStatement` qui permet ceci
- `fetch()`
 - Séquentielle => moins de charge serveur. Indispensable sur les requêtes renvoyant beaucoup d'enregistrements.
 - Pas d'accès direct au nombre d'enregistrements retournés par la requête.
 - Utilise le buffer du SGBD pour récupérer les résultats un à un.
- `fetchAll()`
 - Accès direct au nombre d'enregistrements retournés.
 - Le buffer du SGBD est vidé à la fin de l'exécution de la méthode.
 - Consomme beaucoup de mémoire pour des requêtes retournant beaucoup de résultats (retourne un tableau PHP)

PDO - Lire enregistrements, PDOStatement



- fetch()

```
while ($result = $stmt->fetch()) {  
    vprintf("num :%d - nom:%s\n", $result)  
}
```

- fetchAll()

```
$results = $stmt->fetchAll();  
foreach($results as $result) {  
    vprintf("num :%d - nom:%s\n", $result);  
}
```

PDO - Lire les enregistrements : fetch modes



▸ fetch() comme fetchAll() acceptent un paramètre

- PDO::FETCH_BOTH (défaut)
- PDO::FETCH_ASSOC
- PDO::FETCH_OBJ
- PDO::FETCH_CLASS

```
$result = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

▸ setFetchMode() fonctionne aussi sur PDOStatement

```
$stmt->setFetchMode(PDO::FETCH_ASSOC);  
$result = $stmt->fetchAll();
```

▸ Alias : fetchColumn() - fetchObject()

PDO - Gestion des transactions



- PDO fonctionne en mode « autocommit » par défaut
Pour utiliser une transaction, il convient de respecter le schéma suivant :
- Indiquer à PDO que l'on souhaite commencer une transaction avec la méthode
 - beginTransaction() (PDO sort alors du mode autocommit)
 - Valider la transaction (commiter) avec la méthode commit() ou l'annuler avec la méthode rollBack().

```
<?php
try {
    $pdo->beginTransaction();
    $sql = "INSERT INTO auteurs (nom, prenom) VALUES ('de Nerval', 'G rard)";
    $pdo->exec($sql);

    $idInsere = $pdo->lastInsertId();
    // violation volontaire de cl  primaire
    $sql = "INSERT INTO auteurs VALUES ($idInsere, 'EL DESDICHADO', 'Ramos)";
    $pdo->exec($sql);
    $pdo->commit();
} catch (PDOException $e) {
    $pdo->rollback();
    echo $e->getMessage() . "<br/>";
}
```


PDO - Gestion des transactions



- Le SGBD doit supporter les transactions, dans le cas contraire PDO ne renvoie aucune erreur
- Si une transaction n'est pas fermée en fin de script, rollback() est automatiquement appelée (et la transaction est donc annulée)



- Une chaîne contenant une apostrophe peut faire dérailler le SGBD.
- Pour utiliser réellement une apostrophe il faudra lui appliquer un échappement.
 - Le plus souvent il s'agit d'ajouter un antislash devant (\') ou de doubler l'apostrophe (").
- PDO vous propose la méthode `quote()` pour faire cette opération.
 - Oublier d'échapper des caractères spéciaux peut amener de graves problèmes de sécurité (on parle souvent de faille par injection SQL).
 - Vous avez la responsabilité de penser à échapper toutes les chaînes de caractères sans exception avant de les utiliser.
- Les requêtes préparées ne nécessitent pas l'échappement et sont recommandées



- Le principe :
 - créer un modèle de requête et l'enregistrer sur le SGBD le temps de l'exécution du script
 - par opposition aux procédures stockées qui sont, elles, enregistrées de manière permanente.
- Quand vous aurez besoin de faire une requête vous ferez appel à ce modèle.
- Le serveur exécutera alors votre requête en utilisant les données que vous lui aurez fournies en paramètres pour construire la requête SQL complète.



- Les requêtes préparées sont recommandées pour :
 - Les requêtes multiples
 - La requête n'est interprétée qu'une seule fois mais peut être exécutée plusieurs fois avec des paramètres identiques ou différents.
 - En utilisant des requêtes préparées vous éviterez de répéter le cycle d'analyse / compilation / optimisation.
 - Protéger vos requêtes
 - Les paramètres des requêtes préparées n'ont pas besoin d'être protégés, le pilote de votre base de données le fait tout seul.
 - Vous vous protégez donc des attaques classiques par injection SQL.
- Séparer la syntaxe SQL des données insérées : celles-ci ne peuvent plus modifier la sémantique de la requête



- Pour construire un modèle de requête il suffit de remplacer chaque paramètre par un point d'interrogation ou par paramètre nommé (:nom).
- Vous fournirez les paramètres à substituer quand vous exécuterez réellement la requête.
- Requête normale
`$sql = "INSERT INTO auteurs (nom_auteur) VALUES ('Toto')";`
- Modèle de requête avec des paramètres nommés (recommandé)
`$sql2 = 'INSERT INTO auteurs (nom_auteur) VALUES (:nom)';`
- Modèle de requête avec des points d'interrogation
`$sql3 = 'INSERT INTO auteurs (nom_auteur) VALUES (?)';`



- Il faut choisir
 - Il n'est pas possible d'utiliser à la fois des noms de paramètre (:nom) et des points d'interrogation. Il faut choisir. Vous ne pouvez pas non plus mettre le même nom de paramètre plusieurs fois.
 - Un des avantages notable de cette méthode est que vous êtes protégé des attaques classiques dites par injection SQL
- C'est le SGBD qui vérifiera que les données transmises sont correctes et qui fera les échappements nécessaires.
- Les données transmises ne peuvent plus changer la sémantique de la requête, le SGBD l'a déjà compilée
-

PDO - Préparer une requête



- Vous devrez fournir la requête à votre SGBD pour qu'il l'analyse grâce à la méthode `prepare()`
- Cette méthode prend en argument le modèle de requête SQL et renvoie une instance de la classe `PDOStatement` qu'il faudra utiliser pour les futures opérations de traitement.
- En cas d'échec la méthode renvoie `FALSE`.

```
<?php
$sql = "INSERT INTO auteurs (nom_auteur, prenom_auteur)
VALUES (:nom, :prenom)";
$stmt = $pdo->prepare($sql);
// $stmt est en attente de données
```

- Attention, pas de quotes autour des paramètres (':nom')

PDO - Préparer une requête



- Une fois votre modèle de requête créé, il vous faudra le remplir.
La façon la plus simple consiste à passer un tableau contenant les différentes
- valeurs à la méthode `execute()` de votre objet `PDOStatement`.

```
$sql = "INSERT INTO auteurs (nom_auteur, prenom_auteur)
VALUES (:nom, :prenom)";
$stmt = $pdo->prepare($sql);
$nom = 'Bohdanowicz';
$prenom = 'Romain';
$stmt ->execute(array(' :nom'=>$nom, ' :prenom'=>$prenom));
printf("%d lignes affectées", $stmt->rowCount());
```

- `RowCount()` renseignera sur les lignes affectées, si besoin

PDO - Préparer une requête - 2e approche



- L'autre approche, plus pointue, consiste à associer distinctement chaque paramètre de la requête SQL à une variable PHP ou à une valeur.
 - `bindParam (parametre, &variable [, type [, taille]])`
 - `bindValue (parametre, variable [, type [, taille]])`
- `bindParam()` lie une variable PHP à une variable passée en paramètre.
- `bindValue()` lie une valeur à un paramètre
- `execute()` exécute alors la requête en prenant en compte les liaisons effectuées précédemment au moyen de `bind*`()

PDO - Préparer une requête - 2e approche



```
$sql = "INSERT INTO auteurs (nom_auteur, prenom_auteur) VALUES (:nom, :prenom)";  
$stmt = $dbh->prepare($sql);  
$stmt->bindParam(':prenom', $prenom) ;  
$stmt->bindParam(':nom', $nom) ;  
$nom = 'Bohdanowicz';  
$prenom = 'Romain';  
$stmt->execute();  
$nom = 'Dupont';  
$stmt->execute();
```

- Pour exécuter une requête qui a été préparée précédemment on utilise la méthode `execute()`
- Lors de l'exécution, PHP lira le contenu des variables liées aux paramètres pour exécuter la requête.
 - Dans l'exemple précédent nous faisons appel au modèle plusieurs fois de suite. C'est l'un des intérêts des requêtes préparées.
 - Si vous faites dix insertions de cette façon, le traitement sera plus rapide que si vous faisiez dix insertions avec la méthode `exec()`



PHP HTTP Client

PHP HTTP Client - Introduction



- PHP dispose d'une intégration avec cURL via la bibliothèque libcurl
- cURL étant bas niveau de nombreuses bibliothèques proposent des APIs plus simple à utiliser :
 - Guzzle : <https://docs.guzzlephp.org/en/stable/>
 - Symfony HTTP Client : https://symfony.com/doc/current/http_client.html
 - Wordpress Requests : <https://github.com/WordPress/Requests>
 - ...



- Requête GET

```
<?php
$ch = curl_init();

curl_setopt($ch, CURLOPT_URL, "https://jsonplaceholder.typicode.com/users");
curl_setopt($ch, CURLOPT_HEADER, 0);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$body = curl_exec($ch);

$users = json_decode($body, true);

foreach ($users as $user) {
    echo $user['name'] . "\n";
}

curl_close($ch);
```



▸ Requête POST

```
<?php
$ch = curl_init();

curl_setopt($ch, CURLOPT_URL, "https://jsonplaceholder.typicode.com/users");
curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type:application/json'));
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_POST, true);

curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode([ 'name' => 'Romain' ]));

$body = curl_exec($ch);
$user = json_decode($body, true);

echo "id: {$user['id']}, name: {$user['name']}\n"; // id: 11, name: Romain

curl_close($ch);
```

PHP HTTP Client - Symfony HTTP Client



- Requête GET

```
<?php
require 'vendor/autoload.php';

use Symfony\Component\HttpClient\HttpClient;

$httpclient = HttpClient::create();
$response = $httpClient->request('GET', 'https://jsonplaceholder.typicode.com/users');

$users = $response->toArray();

foreach ($users as $user) {
    echo $user['name'] . "\n";
}
```

PHP HTTP Client - Symfony HTTP Client



▸ Requête POST

```
<?php
require 'vendor/autoload.php';

use Symfony\Component\HttpClient\HttpClient;

$httpclient = HttpClient::create();
$response = $httpClient->request('POST', 'https://jsonplaceholder.typicode.com/users', [
    "json" => ['name' => 'Romain'],
]);

$user = $response->toArray();

echo "id: {$user['id']}, name: {$user['name']}\n"; // id: 11, name: Romain
```




PHP XML



- Choisir un format de représentation de données normalisé
- Transporter des données
 - C'est la première utilisation d'XML : transport de données structurées entre applications
- XML : eXtensible Markup Language
 - Un langage universel permettant d'interfacer des systèmes ne parlant pas la même langue
 - Issu du SGML
 - Document structuré et validable
 - Document transformable



- XML permet un stockage structuré et hiérarchique de l'information. C'est très pratique pour maintenir les relations entre les données.
- XML est un format texte, Unicode, facile à éditer et à transmettre. Cela facilite l'interopérabilité et l'édition de données.
- XML permet de valider la structure des données
XML possède de nombreux outils de manipulation standardisés (DOM, SAX).
- XML est compris par la majorité des applications et technologies, c'est un standard du W3C
- XML représente un document sous forme d'arbre de noeuds



- Utilisation de services fournis par des tiers (services Web).
- Echanges basés sur un format standard et connu de tous.
- Agrégation de flux RSS.
- Génération de sorties multiples à partir d'un seul élément :
- XML →
 - PDF
 - HTML
 - OpenOffice
 - Word



- Description par balise
 - Toute balise doit être fermée
 - Une balise "racine" doit encapsuler toutes les autres
- Hiérarchie
- Entités et caractères spéciaux

```
<products>  
  <product id="3" title="Disque dur">  
    <description>Un super disque.</description>  
  </product>  
  <product id="4" title="Fleurs" />  
</products>
```



- Description par balise
 - Toute balise doit être fermée
 - Une balise "racine" doit encapsuler toutes les autres
- Hiérarchie
- Entités et caractères spéciaux

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="bouteille1.xsl"?>
<products>
  <!-- produits qui sont en ligne -->
  <online:product id="3" title="Disque dur">
    <online:description><![CDATA[Regardez <ça>]]></online:description>
  </online:product>
  <product id="4" title="Fleurs"/>
</products>
```



- **L'espace de noms** : Espace de rangement des données.
C'est un peu comme si nous voulions séparer les pommes et les bananes d'Afrique et les pommes et les bananes d'Amérique, l'espace de nom serait le pays.
L'espace de noms est défini à une adresse précisée dans l'attribut `xmlns:` de la balise racine <http://www.xml.com/pub/a/1999/01/namespaces.html>
- **L'encoding** est important pour la validité du fichier XML, il renseigne le parseur sur le jeu de caractères à utiliser et permet d'éviter des incohérences.
En règle générale et dans la majorité des cas, on utilisera UTF-8, dans tous les cas de l'Unicode
- **Une feuille de styles XSL** contient les informations de mise en forme d'un flux (en HTML, ou PDF). On peut indiquer dans le fichier XML une feuille de style XSL pour formater le contenu dans un navigateur web.
- **La balise spéciale CDATA** permet de renfermer une chaîne de caractères sujette à des incohérences, soit parce qu'elle contient des balises qui peuvent interférer avec le fichier XML (`< > & " '`), soit parce que l'encoding n'est pas bon.



- Afin de s'assurer que le XML que l'on veut analyser est correct, on demande la plupart du temps à le valider (sémantiquement)
- Différentes technologies de validation existent :
 - DTD (DocType)
 - Le plus courant mais assez verbeux
 - RelaxNG
 - Le plus simple
<http://www.oasis-open.org/committees/relax-ng/spec-20011203.html>
 - XML Schema
 - Fichiers XSD
<http://xmlfr.org/w3c/TR/xmlschema-0/>
- Ces technos ont pour but de décrire à quoi doit ressembler un fichier XML valide



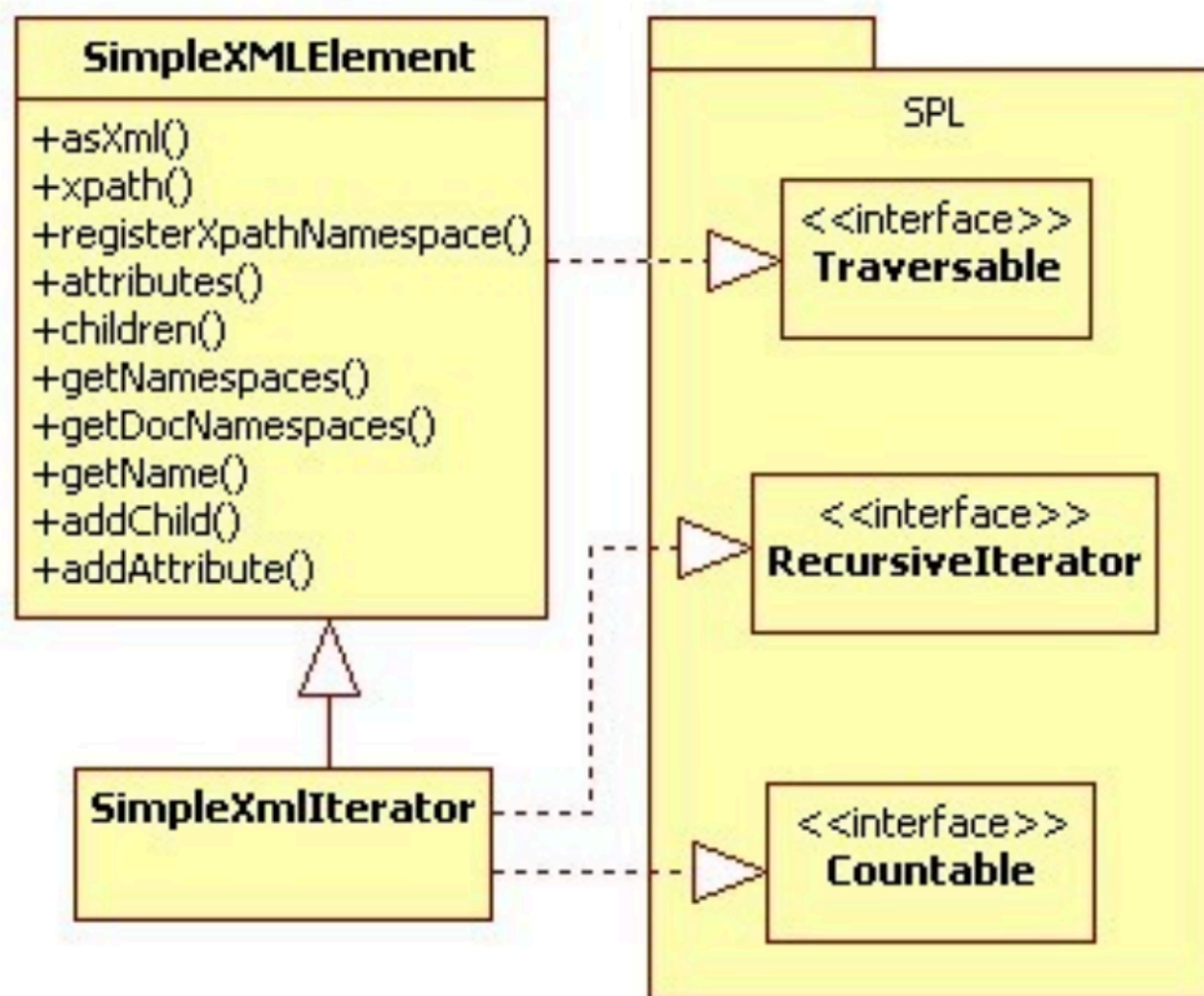
- SOAP (Simple Object Access Protocol)
 - Protocole permettant de mettre en place des services Web
- RSS (Really Simple Syndication)
 - Protocole d'échange de nouvelles au travers du Web
- WDDX (Web Distributed Data eXchange)
 - Outil de sérialisation pour échanger des données structurées par paquet
- XSLT (eXtensible Stylesheet Language Transformations)
 - Mécanisme qui analyse et fusionne deux documents XML, l'un contenant les données, l'autre un jeu de styles



- SAX (Simple API for XML)
 - Parseur séquentiel éprouvé. Il lit le document et fait appel à des gestionnaires au fur et à mesure de son parcours : à une balise est associé une action.
- DOM (Document Object Model)
 - Outil de manipulation d'arbre. La navigation se fait de noeuds en noeuds, tout le document est chargé en mémoire d'un coup
- SimpleXML
 - Outil simple de manipulation XML.
- LibXML2
 - Bibliothèque libre écrite en C utilisée par PHP pour parser (analyser) le XML. Certaines fonctions de PHP permettent de prendre la main sur la libXml et donc d'impacter son analyse du XML
Voir <http://xmlsoft.org/html/libxml-parser.html>



- SimpleXML répond de manière Simple à la gestion de documents XML de structure relativement simple aussi (flux rss)
- S'assure de la validité syntaxique XML de son entrée
 - Absorbe ainsi la balise "root" (racine)
- Utilise des objets composites SimpleXMLElements
- Utilise des tableaux
 - si 2 noeuds de même niveau possède la même valeur
 - pour les attributs XML
- SimpleXML ne gère pas les balises imbriquées avec du texte





- 2 fonctions PHP
 - `simplexml_load_file($fichier)`
 - `simplexml_load_string($chaine)`
- Ou le constructeur de `SimpleXmlElement`
 - `new SimpleXmlElement($fileOrString, $options, $url)`

```
<?php
$xml = simplexml_load_file('mon_fichier.xml');
$xml = simplexml_load_string('<foo></foo>');
$xml = new SimpleXmlElement('<foo></foo>');
$xml = new SimpleXmlElement('mon_fichier.xml', null, 1);
```

- Si le XML n'est pas bien encodé ou pas valide syntaxiquement, un warning sera généré



- Vous accédez en lecture en parcourant l'arbre xml (parents<->enfants) au moyen des attributs sur l'objet
- Utilisez des tableaux numériques lorsque plusieurs noeuds de même niveau existent (des "frères")
- Utilisez des tableaux associatifs pour accéder aux attributs des noeuds

```
<membres>
  <membre id="4" nom="Bob">
    <message num="21">foo bar</message>
    <message num="24">hello world !</message>
  </membre>
  <membre id="4" nom="Estelle">
    <message num="13">formation PHP</message>
  </membre>
</membres>
```

```
<?php
$xml = simplexml_load_file($xmlFile);
echo $xml->membre[0]->message[1]; // hello world!
echo $xml->membre[1]['nom']; // Estelle
```



- SimpleXML ne traite qu'en UTF-8 (en interne)
- Si le xml d'entrée précise un encodage, SimpleXML le convertit en UTF-8 automatiquement (c'est une caractéristique de XML en général)
 - `<?xml version=1.0" encoding="iso8859-1">`
- Si le xml d'entrée ne précise pas d'encodage, les caractères doivent être encodés en UTF-8 sinon SimpleXML ne peut les lire (les convertir) et générera une erreur
 - `<?xml version=1.0">`
- La sortie de SimpleXML est de l'UTF-8 quoiqu'il arrive, pensez à les décoder si vous affichez dans un autre charset, ou à changer le charset dans la réponse HTTP (en-tête Charset).
- SimpleXML gère les espaces de noms, en constructeur, ou via les méthodes `children()` et `attributes()`



- L'écriture est aussi simple que la lecture
 - Via attributs sur l'objet simplexmlelement
 - ou via la méthode addChild(\$nom, \$valeur) qui retourne alors le noeud créé
- Exemple d'ajout d'un nouveau membre et d'un message pour lui :

```
<?php
$membre = $xml->addChild( 'membre' );
$membre[ 'id' ] = 10;
$membre[ 'nom' ] = "exemple";
$membre->message[ 'num' ] = 99;
$membre->message = "nouveau message";
```




- La méthode `asXml()` déclenche une sortie XML du noeud sur lequel on l'applique
 - `$xml = $simpleXmlNode->asXml();`
- Si on lui passe un paramètre : elle écrit dans un fichier
 - `$simpleXmlNode->asXml('/path/to/file.xml');`
- La sortie est de l'UTF-8, on devra donc s'arranger d'une manière ou d'une autre pour la traiter
 - `header("Content-Type:text/plain; charset=utf-8");`
`iconv_set_encoding('internal_encoding','utf-8');`
`iconv_set_encoding("output_encoding", "iso-8859-1");`
`ob_start('ob_iconv_handler');`



- Document Object Model : Standardisation pour échange de documents XML
- PHP DOM respecte le niveau 3 : <http://www.w3.org/TR/DOM-Level-3-Core/>
- Basée sur la bibliothèque libXml2 (phpinfo pour vérifier la version)
 - Plus complexe que SimpleXML
 - Plus de possibilités



- DomDocument sait charger :
 - Du XML : `load($xmlFile)` ou `loadXML($xmlstring)`
 - Du HTML : `loadHtml($htmlString)` ou `loadHtmlFile($htmlFile)`
- DomDocument sait sauvegarder vers le XML ou le HTML (chaine et fichiers)
- En général cela fonctionne mal pour HTML qui est souvent "brouillon"

```
<?php
$dom = new DOMDocument();
$dom->load($xmlFile);
$title = $dom->getElementsByTagName('title');
printf("Chapitres de %s \n", $title->item(0)->nodeValue);
foreach ($dom->getElementsByTagName('chapter') as $chapter) {
    $titles = $chapter->getElementsByTagName('title');
    printf("\n- %s \n", $titles->item(0)->nodeValue);
    $paragraphs = $chapter->getElementsByTagName('paragraph');
    foreach ($paragraphs as $paragraph) {
        printf(" * %s \n", $paragraph->nodeValue);
    }
}
```



- Dans DOM, tout hérite de DomNode, quel que soit l'objet que vous possédez, vous pouvez :
 - `$element` est un DomElement
 - `$element->parentNode` : le noeud parent
 - `$element->previousSibling` : le noeud frère précédent
 - `$element->nextSibling` : le noeud frère suivant
 - `$element->firstChild` : Renvoie le premier enfant
 - `$element->lastChild` : Renvoie le dernier enfant
 - `$element->childNodes` : Renvoie la liste des noeuds enfants (une DomNodeList)
 - `$element->hasAttribute('attribut')` : Vérifie si le Node possède un attribut
 - `$element->getAttribute('attribut')` : Renvoie l'attribut
- ... La documentation est indispensable au début



- \$dom est un DomDocument
- \$dom->getElementById() : Renvoie le DomElement correspondant par l'attribut id (recommandé)
 - Pour pouvoir utiliser un identifiant de noeud, il faut au préalable valider le document envers sa DTD.
 - Si le document est du HTML, il devra comporter un DocType
 - Si c'est du XML : il devra comporter une DTD
- \$dom->getElementsByTagName() : Renvoie une DomNodeList de DomElement correspondant au tag demandé
- Valider un document XML (ou HTML)

```
$dom = new DOMDocument();  
$dom->validateOnParse = true;  
$dom->loadXML($xml);
```

```
$dom = new DOMDocument();  
$dom->loadXML($xml);  
$dom->validate();
```



- Lecture d'un flux XML
- Modification du titre et ajout d'une balise « category »

```
$dom = new DOMDocument();  
$dom->loadXML($xml);  
$title = $dom->getElementsByTagName('title');  
$title->item(0)->nodeValue = "Bonnes pratiques PHP 8";  
$element = $dom->createElement('category', 'PHP');  
$rootElement = $dom->getElementsByTagName('document');  
$rootElement->item(0)->appendChild($element);  
$dom->save('/tmp/document.xml');
```

PHP XML - Xpath : un langage pour XML



- XPath permet d'effectuer des requêtes pour accéder à des données dans un flux XML.
- Xpath est performant, sa syntaxe est un peu complexe mais le retour sur apprentissage est très bon
- Un excellent tutoriel pour commencer avec XPath :
- <http://www.zvon.org/xxl/XPathTutorial/General/examples.html>

```
$result = $simpleXml->xpath('/document/chapter/paragraph');  
foreach ($result as $paragraph) { ... }
```

```
$dom = new DomDocument();  
$dom->load(...);  
$xpath = new DomXPath($dom);  
$result = $xpath->query('/document/chapter/paragraph');  
foreach ($result as $paragraph) { ... }
```



- DOM et SimpleXML sont interchangeables
 - `$domElement = dom_import_simplexml($simpleXmlElement)`
 - `$sxml = simplexml_import_dom($domNode)`

```
<?php
$s = new SimpleXMLElement( '<members></members>' );
$s->member[0]['id'] = 1;
$s->member[0]['id'] = 1;
$s->member[0]->name = 'coucou';
$s->member[0]->city = 'paris';
$s->member[1]['id'] = 2;
$s->member[1]->name = 'anaska';
$s->member[1]->city = 'los angeles';

$dom = dom_import_simplexml($s);
$dom = $dom->ownerDocument;
$dom->formatOutput = true;
$dom->encoding = 'utf-8';
echo $dom->saveXml();
```




- Parser basé sur des événements (opentag, closetag, data...)

```
<?php

class xml
{
    public $parser;
    public $tags = [];
    function __construct()
    {
        $this->parser = xml_parser_create();
        xml_set_object($this->parser, $this);
        xml_set_element_handler($this->parser, "tag_open", "tag_close");
        xml_set_character_data_handler($this->parser, "cdata");
    }
    function parse($data)
    {
        xml_parse($this->parser, $data);
    }
    function tag_open($parser, $tag, $attributes)
    {
        $this->tags[] = $tag;
    }
    function cdata($parser, $cdata)
    {
        $lastTag = $this->tags[count($this->tags) - 1];
        echo "$lastTag: $cdata\n";
    }
    function tag_close($parser, $tag)
    {
        array_pop($this->tags);
    }
}
```



- Pratiquement aucune consommation mémoire

```
$xml_parser = new xml();  
$xml = <<<STR  
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE contacts SYSTEM "contacts.dtd">  
<contacts>  
  <contact>  
    <prenom>Jean</prenom>  
    <nom>Dupont</nom>  
    <adresse>  
      <ville>Paris</ville>  
      <codePostal>75013</codePostal>  
    </adresse>  
  </contact>  
  <contact>  
    <prenom>Eric</prenom>  
    <nom>Martin</nom>  
    <adresse>  
      <ville>Marseille</ville>  
      <codePostal>13001</codePostal>  
    </adresse>  
  </contact>  
</contacts>  
STR;  
$xml_parser->parse($xml);
```



PHP JSON



- JSON, JavaScript Object Notation est la sérialisation d'un objet JavaScript
- Seuls les types string, number, boolean, array, object littéral sont sérialisables, les fonctions et prototype sont perdus
- On se sert de ce format pour échanger des données entre 2 programmes ou pour créer de la config
- Le format résultant est proche de Object Literal, les clés sont obligatoirement entre guillemets "", un code JSON est une syntaxe Object Literal valide

```
{  
  "name": "My Address Book",  
  "contacts": [  
    {  
      "firstName": "Bill",  
      "lastName": "Gates"  
    },  
    {  
      "firstName": "Steve",  
      "lastName": "Jobs"  
    }  
  ]  
}
```



- JSON est souvent utilisé en remplaçant d'XML pour des fichiers de configuration ou des échanges réseaux (entre un client et un serveur par exemple), car plus lisible, moins verbeux et plus simple à manipuler dans le code
- L'exemple représentant la même structure de données, contient hors espaces et retours à la ligne : 115 caractères en JSON contre 217 en XML

```
{  
  "name": "Address Book",  
  "contacts": [  
    { "firstName": "Bill", "lastName": "Gates" },  
    { "firstName": "Steve", "lastName": "Jobs" }  
  ]  
}
```

```
<addressBook>  
  <name>My Adress Book</name>  
  <contacts>  
    <contact>  
      <firstName>Bill</firstName>  
      <lastName>Gates</lastName>  
    </contact>  
    <contact>  
      <firstName>Steve</firstName>  
      <lastName>Jobs</lastName>  
    </contact>  
  </contacts>  
</addressBook>
```



- Pour encoder et décoder on utilise les méthodes `json_encode` et `json_decode`

```
<?php
echo json_encode([ 'name' => 'Romain' ]); // {"name": "Romain"}

echo json_encode([ 'name' => 'Romain' ], JSON_PRETTY_PRINT);
// {
//     "name": "Romain"
// }
```

```
<?php
$obj = json_decode( ' {"name": "Romain"} ' );
echo $obj->name;

$obj = json_decode( ' {"name": "Romain"} ', true);
echo $obj[ 'name' ];
```



Sécurité PHP



- Tout ce qui est extérieur à l'application est suspect (Requête HTTP, Base de données, Services Web...)
- Préférer des whitelists aux blacklists
- Filtrer les données avec `filter_input` ou les casts



- Les injections SQL proviennent d'entrées utilisateurs mal protégées et qui sont envoyées à la base de données.
- `$req = "SELECT * FROM admin WHERE login='$login' AND password='$pass'";`
- `$req = "SELECT * FROM admin WHERE login='admin' -- ' AND password='n importe'";`
- Généralement il y a des fonctions spécifiques à la base de données. Il est préférable de les utiliser. Sinon : `"addslashes()"`

```
<?php
/* MySQL data validation */
$str = mysqli_escape_string(stripslashes($_GET['input']));
/* PostgreSQL validation */
$str = pg_escape_string(stripslashes($_GET['input']));
$str = pg_escape_bytea(stripslashes($_GET['input'])); // pour des données binaires
$str = $pdo->quote($_GET['input']);
/* les extensions MSSQL, ODBC, OCI n'ont pas leur propre fonction d'échappement
* addslashes() devra être utilisé */
```



- Dans la plupart des cas il est préférable de ne pas autoriser l'exécution de commandes sur votre système via PHP.
- Si vous le devez il faut faire très attention.
- Deux fonctions permettent de vous prémunir :
 - Escapeshellcmd()
 - Escapeshellarg()
- Quand PHP est en module apache, toutes les commandes seront exécutées avec les droits de l'utilisateur Apache.

Sécurité PHP - Cross-Site Scripting (XSS)



- Cette vulnérabilité est le résultat d'une entrée utilisateur mal échappée et affichée à l'écran, grâce à Javascript
- Javascript est extrêmement puissant, et destructeur : il permet de modifier l'affichage de la page, de lire les cookies, de faire une redirection ...
- Pour se protéger :
 - htmlspecialchars() - encode les caractères HTML pour éviter une injection HTML
 - htmlentities() - encode tous les caractères (é,è,@) pour lesquels il y a un équivalent entité HTML
 - strip_tags() - enlève tout ce qui ressemble à un tag HTML (attention aux attributs XHTML !)

Sécurité PHP - Cross-Site Scripting (XSS)



- Exploitation de la confiance d'un site en un utilisateur (dont les privilèges sont élevés)
- Utilisée afin de faire faire quelque chose à un site Possibilité de créer une passerelle Internet<->Intranet
-



- La directive `open_basedir` permet de restreindre l'accès d'un utilisateur à une arborescence.
- Il ne lui sera pas possible d'aller se balader grâce à des `../../..`
- Tout ceci reste théorique, il a été prouvé qu'on pouvait "bypasser" l'`open_basedir` sur certaines versions de PHP
-



PHP Performance

PHP Performance - Checklist



- mettre à jour PHP
- désactiver xdebug
- activer opcache
- composer dump-autoload -o
- cache de requêtes SQL
- cache de config
- .htaccess dans le virtual host
- tester avec Apache Bench (ab)
- utiliser un Profiler
- décomposer les requêtes SQL lentes avec EXPLAIN