

Migrando para o PHP7

Prof. Er Galvão Abbott

Licença de Uso

Resumo:

Você é livre para:

- Compartilhar – Copiar e redistribuir este material em qualquer meio ou formato;
- Adaptar – Remixar, transformar e usar este material como base para qualquer propósito, até mesmo Comercial.

Contanto que você o faça sob as seguintes condições:

Atribuição – Você precisa dar o devido crédito, fornecer um link para a licença e indicar explicitamente que mudanças foram feitas. Você pode fazer isto de qualquer maneira razoável, mas isso não implica, de forma alguma, que o licenciador apóia você ou o seu uso deste material.

Compartilhar da mesma forma – Se você remixar, transformar ou usar este material como base, você precisa distribuir o seu material sob a mesma licença do original.

Link para o texto completo da licença:

<https://creativecommons.org/licenses/by-sa/4.0/legalcode>

Índice

Licença de Uso.....	2
O Fim do PHP5: É hora de migrar!.....	4
Razões para migrar.....	4
PHP 7.0 - Novas <i>Features</i>	5
Operadores.....	5
Null Coalesce: ??.....	5
Spaceship: <=>.....	6
Tipos.....	7
Arrays como valores de Constantes.....	7
Geração de Dados Aleatórios.....	8
random_int(\$min, \$max).....	8
random_bytes(\$len).....	8
Configuração de Sessões.....	9
Tipagem.....	9
STH – Scalar Type Hints.....	9
RTD – Return Type Declarations.....	10
Tipagem Estrita.....	10
Classes Anônimas.....	11
Desserialização filtrada.....	12
Group Use Declarations.....	13
PHP 7.0 - Quebras de Compatibilidade.....	14
Estruturas de Controle.....	14
O fim dos <i>defaults</i> múltiplos.....	14
Demais Quebras de Compatibilidade.....	15
PHP 7.1 - Novas <i>Features</i>	16
Nullable Types.....	16
Void.....	16
Visibilidade de Constantes de Classe.....	17
Múltiplos catch.....	17
Quebras de Compatibilidade Reversa.....	19
Próximas Versões: O que esperar.....	19
PHP 7.2.....	19
Referências.....	19
Bibliografia Recomendada.....	19

O Fim do PHP5: É hora de migrar!

Janeiro de 2019 ficou marcado na história da linguagem como o dia em que foi *major* 5 chegou ao seu *End-of-Life*. Isso significa que, para todos os fins práticos PHP 5 **não existe mais**. Não há mais atualizações oficiais nesta *major*, nem mesmo para questões de segurança.

O que muita gente não sabe é que a migração, pelo menos quando se trata de aplicações "comuns", é um processo relativamente simples. Neste curso veremos mudanças relacionadas à quebras de compatibilidade reversa, bem como melhorias que um sistema PHP5 pode sofrer de forma a ilustrar não apenas a simplicidade da migração, mas também os benefícios que o seu código pode obter com a nova versão.

Razões para migrar

- O fim da versão 5;
- Performance: Com uma nova *engine* o PHP7 é, pelo menos, duas vezes mais rápido em seu processamento do que a versão anterior;
- Além disso, o uso de memória caiu significativamente, quase pela metade quando comparado com a versão anterior;
- Novidades: Diversas novas *features*, como Scalar Type Hinting, Return Type Declarations, Group Use Declarations, entre tantas outras que serão abordadas neste curso só estão disponíveis na versão 7. Como veremos no decorrer do curso, estas novas *features* levam a linguagem a um patamar completamente novo.

Abaixo iniciamos um conteúdo de Referência Rápida para o que muda na versão 7, em cada *minor*. Comentários adicionais e maiores informações serão fornecidas durante o curso.

PHP 7.0 - Novas *Features*

Operadores

Dois novos operadores foram introduzidos na versão 7:

Null Coalesce: ??



Retorna operando a esquerda se este não é nulo, senão o operando a direita.

PHP5:

```
<?php
echo (isset($foo) ? $foo : 'Sem valor'); // Output: Sem valor
```

PHP7:

```
<?php
// Exemplo 1

echo ($foo ?? 'Sem valor'); // Output: Sem valor

// Exemplo 2

$z = 1;
echo ($x ?? $y ?? $z ?? 'Sem valor'); // Output: 1
```

Spaceship: <=>



Retorna 1 se operando a esquerda é maior, 0 se iguais, -1 se o da direita é maior.

PHP5:

```
<?php
$x = 5;
$y = 2;
echo ($x > $y ? 1 : ($x == $y ? 0 : -1)); // Output: 1
```

PHP7:

```
<?php
$x = 5;
$y = 2;

echo $x <=> $y; // Output: 1
```

Tipos

Arrays como valores de Constantes

PHP5: Não era possível, trabalhando-se com algo como:

```
<?php
define('DB_HOST', 'localhost');
define('DB_USER', 'foo');
define('DB_PASS', 'bar');
define('DB_PORT', 3306);
define('DB_NAME', 'my_database');
```

PHP7:

```
<?php
define('DB', [
    'HOST' => 'localhost',
    'USER' => 'foo',
    'PASS' => 'bar',
    'PORT' => 3306,
    'NAME' => 'my_database',
]);
```

Geração de Dados Aleatórios

- `random_int`
- `random_bytes`

`random_int($min, $max)`

 Retorna um número inteiro \geq \$min e \leq \$max


PHP5:

```
<?php
echo mt_rand(1, 10);
```

PHP7:

```
<?php
echo random_int(1, 10);
```

`random_bytes($len)`

 Retorna um dado binário de comprimento \$len

PHP5: Não era possível, tendo-se que implementar uma solução customizada, [como essa](#).

PHP7:

```
<?php
$foo = random_bytes(32);
echo bin2hex($foo); // Output: Um dado aleatório representado em hexadecimal
```


Configuração de Sessões


PHP5: Configuráveis apenas via php.ini, .htaccess, etc...

PHP7:

```
<?php
session_start([
    'name'           => 'mysession',
    'use_only_cookies' => '1',
    // etc ...
]);
```

Tipagem

STH – Scalar Type Hints

 Tipar parâmetros de funções/métodos usando tipos escalares (int, float, string, bool).

PHP5: Não era possível

PHP7:

```
<?php
function foo(int $bar)
{
    return $bar + 1;
}
```

RTD – Return Type Declarations



Tipar retorno de funções/métodos usando tipos escalares (int, float, string, bool) e compostos.

PHP5: Não era possível

PHP7:

```
<?php
function foo(int $bar): int
{
    return $bar + 1;
}
```

Tipagem Estrita



Gerar uma Exception/Erro Fatal ao violar STH e RTD.

PHP5: Não era possível

PHP7:

```
<?php
declare(strict_types = 1);

function foo(int $bar)
{
    return $bar + 1;
}

foo('1'); // Fatal error: Uncaught TypeError: Argument 1 passed to foo() must
be of the type int, string given, called in ...
```

Classes Anônimas



Geração de objetos sem a necessidade de uma classe permanente

PHP5: Não era possível

PHP7:

```
<?php
$obj = new class(30) {
    public $foo = 12;

    public function __construct($bar)
    {
        $this->foo += $bar;
    }
};

echo $obj->foo; // Output: 42
```

Desserialização filtrada



Prover um mecanismo de desserialização mais seguro

```
<?php
class Foo
{
    public $foo = 12;

    public function __construct($bar)
    {
        $this->foo += $bar;
    }

    public function __destruct()
    {
        echo '---> ' . $this->foo . PHP_EOL;
    }
}

$obj = new Foo(30);
$s    = serialize($obj);

var_dump(unserialize($s));
```

Output:

```
class Foo#2 (1) {
    public $foo =>
    int(42)
}
---> 42
---> 42
```

PHP7:

```
// Mesmo código anterior a esta linha  
  
var_dump(unserialize($s, ['allowed_classes' => FALSE]));
```

Output:

```
class Foo#2 (1) {  
    public $foo =>  
    int(42)  
}  
---> 42
```

Group Use Declarations



Reduzir a repetição de código ao utilizar *namespaces*

PHP5:

```
<?php  
use Foo\Bar\Bar;  
use Foo\Bar\Baz;  
use Foo\Bar\Quux;
```

PHP7:

```
<?php  
use Foo\Bar\{  
    Bar,  
    Baz,  
    Quux  
};
```

PHP 7.0 - Quebras de Compatibilidade

Estruturas de Controle

O fim dos *defaults* múltiplos

```
<?php
$a = 2;

switch ($a) {
    case 0:
        echo 'zero';
        break;
    case 1:
        echo 'um';
        break;
    default:
        echo 'Nem zero, nem um';
        break;
    default:
        echo 'Valor desconhecido';
        break;
}
```

Resultado no PHP5 (independente do uso de *break* nos *defaults*):

Nem zero, nem um

Resultado no PHP7:

PHP Fatal error: Switch statements may only contain one default clause in ...

Demais Quebras de Compatibilidade

Estas quebras não necessitam de exemplos de código-fonte: São funcionalidades que simplesmente foram removidas. Veremos alternativas e detalhes em aula:

- Construtores "estilo PHP4"
- Família de funções `ereg_*`
- Família de funções `mysql_*`
- Tags alternativas (`<? , <% , <script language="php">`)
- Comentários utilizando-se o caractere `#`
- *Warning* de configuração `date.timezone` não definida

PHP 7.1 - Novas *Features*

Nullable Types



Possibilidade de usar NULL em STH/RTD, independente da tipagem.

PHP7.1:

```
<?php
declare(strict_types = 1);

function foo(?int $bar)
{
    if (!empty($bar)) {
        return $bar + 1;
    }

    return 'Nenhum valor recebido';
}

foo(NULL); // Output: 'Nenhum valor recebido'
```

Void



Tipar retorno sem valor.

```
<?php declare(strict_types = 1);
function foo(&$bar): void
{
    $bar++;
}

$x = 41;
foo($x);

echo $x; // Output: 42
```


Visibilidade de Constantes de Classe



Controlar o acesso a constantes de classe

```
<?php declare(strict_types = 1);
class Log
{
    private const PATH = '/foo/bar';
}

echo Log::PATH;
```

Resultado: Fatal error: Uncaught Error: Cannot access private const Log::PATH in ...

Múltiplos catch



Possibilitar o *catch* de vários tipos de Exceções

```
<?php declare(strict_types = 1);
function foo(?int $bar)
{
    if (empty($bar)) {
        throw new Exception('Nenhum valor recebido');
    }

    // ...
}

try {
    foo(TRUE);
} catch (TypeError $e) {
    echo 'Erro de tipagem: ' . $e->getMessage();
} catch (Exception $e) {
    echo 'Outro tipo de erro: ' . $e->getMessage();
}

echo PHP_EOL;

//Output: Erro de tipagem: Argument 1 passed to foo() must be of the type int
or null, bool given, called in ...
```

PHP 7.1 - Quebras de Compatibilidade

- Remoção da família de funções `mcrypt_*`

PHP 7.2 - Novas *Features*

Argon2 na `password_hash`



Possibilitar o uso do algoritmo Argon2 nas funções de tratamento de senhas

```
<?php declare(strict_types = 1);  
  
$senha = password_hash('foo', PASSWORD_ARGON2I);
```

Debug de *Prepared Statements* (PDO) - [RFC1](#), [RFC2](#)



Possibilitar a visualização dos dados entrados em *placeholders*

```
<?php declare(strict_types = 1);  
// Conexão, etc ...  
$sql = 'SELECT foo FROM bar WHERE id=:id';  
  
$sth = $dbh->prepare($sql);  
$sth->bindParam(':id', $id, PDO::PARAM_INT);  
  
$sth->execute();  
  
$sth->debugDumpParams();
```

Tipo de Objeto Genérico



Possibilitar que um parâmetro/retorno seja um objeto de qualquer classe

```
<?php declare(strict_types = 1);  
function dumpObject(object $obj)  
{  
    var_dump($obj);  
}  
  
dumpObject(new class() {  
    public $foo;  
  
    public function __construct()  
    {  
        $this->foo = 42;  
    }  
});
```

Libsodium



Prover mais uma alternativa criptográfica além da openssl

Quebras de Compatibilidade Reversa

Próximas Versões: O que esperar

PHP 7.2

Já existem algumas questões interessantes definidas para a próxima minor da versão 7, entre elas:

- Introdução do algoritmo Argon2 para as funções `password_hash`;
- Introdução da biblioteca criptográfica libsodium como nativa na linguagem (anteriormente estava disponível apenas como extensão PECL);

Referências

- PHP Changelog: <http://php.net/manual/en/doc.changelog.php>
- Suporte Oficial a versões da linguagem: <http://php.net/supported-versions.php>
- RFCs: <https://wiki.php.net/rfc>

Bibliografia Recomendada

- [Upgrading to PHP 7](#)
Davey Shafik
Ed. O'Reilly
- [Learning PHP 7](#)
Antonio Lopez
Ed. Packt