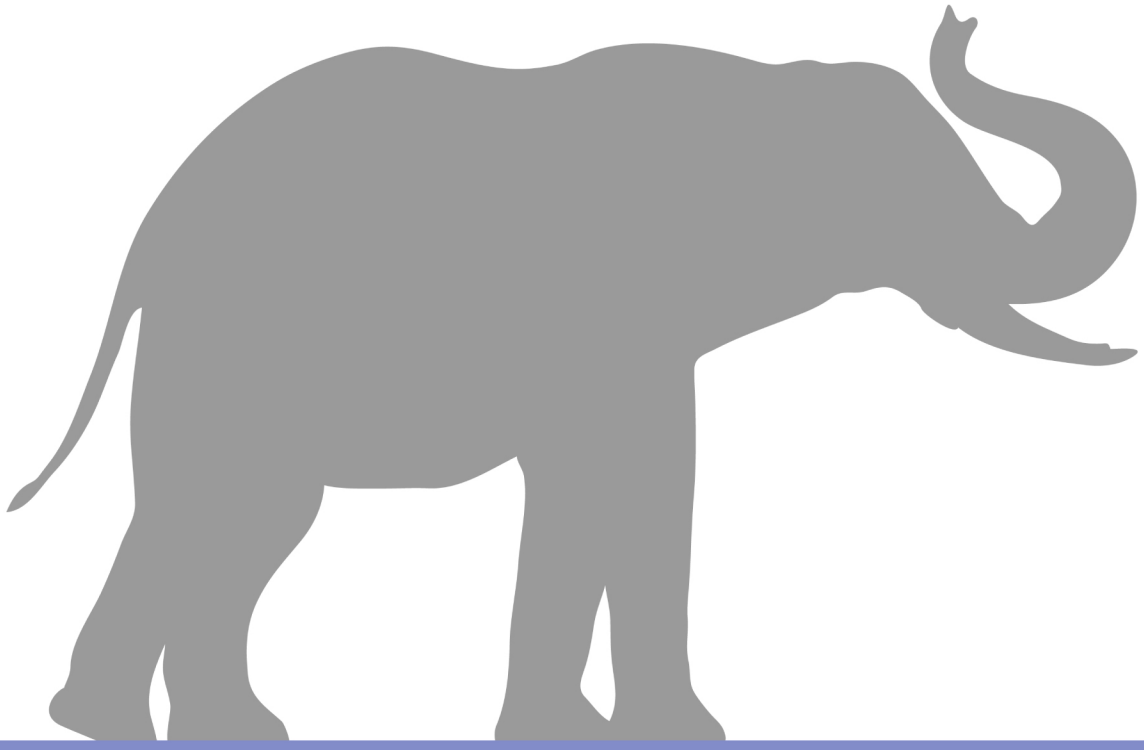


PHP 7

Mudanças e Novidades

Atualizado com mudanças e novidades do PHP 7.1

Jonathan Lamim Antunes

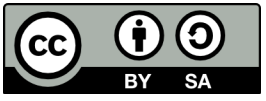
PHP 7 - Mudanças e Novidades

Jonathan Lamim Antunes

Esse livro está à venda em http://leanpub.com/php7-mudancasenovidades

Essa versão foi publicada em 2017-01-21

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License

Para todos que a cada dia me incentivam a compartilhar conhecimento, e principalmente para os que buscam incansavelmente o conhecimento.

Conteúdo

1. O Autor

Jonathan Lamim Antunes

Tudo começou em 2004, quando entrei para a escola técnica. Lá, estudei informática, e aprendi sobre manutenção, redes de computadores e sistemas operacionais, mas o que me atraiu mesmo foram as matérias relacionadas à programação. Aprendi a programar usando Delphi, depois comecei a estudar JavaScript e HTML, e foi aí que me apaixonei por desenvolvimento para web.

Em 2005, concluí o curso técnico e me mudei do interior de Minas Gerais para o Espírito Santo, onde comecei a ter oportunidades de colocar em prática tudo o que já havia aprendido. Comecei então a escrever artigos sobre desenvolvimento web, dar aulas de programação e informática básica, e auxiliar alunos de uma escola particular durante as aulas no laboratório de informática.

Com o passar do tempo, fui me aprofundando nos estudos sobre desenvolvimento web, passei a colaborar com projetos open source e a visão foi se abrindo ainda mais. Quanto mais eu aprendia, mais eu queria ensinar, compartilhar. Já são mais de 300 artigos escritos, muitos horas de aulas ministradas, várias palestras e hangouts, e ainda sinto que posso compartilhar muito mais conteúdo.

Hoje sou CEO da JLamim Tecnologia & Educação, onde trabalhamos com desenvolvimento web e mobile, cursos em diversas áreas, programas de mentoria, consultoria e SEO.

Para conhecer um pouco mais sobre o meu trabalho e meus artigos, veja os links abaixo:

- **Site**: http://www.jonathanlamim.com.br
- **Facebook**: https://www.facebook.com/jonathanLamim
- **Twitter**: https://www.twitter.com/jlamim
- **Instagram**: https://www.instagram.com/jonathanlamim
- **Oficina da Net**: https://www.oficinadanet.com.br/autor/172-jonathan-lamim
- **iMasters**: http://imasters.com.br/perfil/jonathan_lamim

2. Sobre o e-book

Esse ebook foi gerado a partir de uma série de artigos escritos para o blog da Umbler[1], com o intuito de compartilhar com a comunidade de desenvolvedores PHP as mudanças da versão 7 dessa dessa linguagem amada por muitos, e "diminuída" por outros.

Nas páginas desse ebook você verá um pouco do que mudou no PHP 7, o que chegou e o que saiu, e como ele melhorou em termos de performance. O conteúdo não fala sobre 100% das mudanças, mas apresenta os pontos mais importantes da nova versão.

Com o conhecimento adquirido após a leitura desse e-book você já será capaz de atualizar os seus projetos para o PHP 7, e até mesmo escrever novos projetos utilizando o PHP 7.

Atualizações

Janeiro/2017

- Os capítulos 4 e 6 receberam atualizações relacionadas ao PHP 7.1
- Foram adicionados 2 novos capítulos, um falando sobre a extensão **Request** e outro listando mudanças e novidades do PHP 7.1

[1] http://blog.umbler.com

3. Outras Publicações

CodeIgniter - Produtividade na criação de aplicações web em PHP

CodeIgniter - Produtividade na criação de aplicações web em PHP

O CodeIgniter 3.x é um framework MVC open source, escrito em PHP e mantido atualmente pelo British Columbia Institute of Technology e por uma grande comunidade de desenvolvedores ao redor do mundo. Com ele, é possível desenvolver sites, APIs

e sistemas das mais diversas complexidades, tudo de forma otimizada, organizada e rápida. Suas bibliotecas nativas facilitam ainda mais o processo de desenvolvimento, e ainda permitem ser estendidas para que o funcionamento se adapte à necessidade de cada projeto.

Neste livro, você vai criar dois projetos completos, aplicando recursos, bibliotecas, vendo dicas de teoria e boas práticas com o framework CodeIgniter 3.x . Com os conhecimentos adquiridos aqui, você será capaz de desenvolver sites e sistemas com qualidade e rapidez.

Adquira o seu exemplar a partir de R$ 29,90[1]

[1] http://www.livrocodeigniter.com.br

Amazon AWS - Descomplicando a computação na nuvem

Amazon AWS - Descomplicando a computação na nuvem

Quando se trata de grandes aplicações, infraestrutura é um ponto muito importante, pois é preciso pensar em escalabilidade, gerenciamento e, principalmente, os serviços necessários para seu bom funcionamento. AWS ou Amazon Web Services é uma plataforma de serviços na nuvem que oferece soluções para armazenamento, redes e computação, em várias camadas. E o melhor de tudo, você pode administrar todos esses serviços através de uma interface web, ou também por APIs e linha de comando.

Neste livro, você vai aprender a configurar e integrar suas aplicações com os diferentes serviços da Amazon AWS, como o Amazon S3, AWS SDK, EC2, RDS,

ElastiCache, Route 53, CloudFront, CloudWatch, Amazon SES e SNS. Este conjunto de ferramentas lhe possibilitará hospedar e gerenciar facilmente aplicações dos mais variados tamanhos e com um custo possível de ser controlado.

Adquira o seu exemplar a partir de R$ 29,90[2]

O Programador Produtivo - O caminho da produtividade

O Programador Produtivo - O caminho da produtividade

Como programador, uma das coisas mais complicadas é conseguir realizar todas as tarefas em apenas 8 horas diárias de trabalho. E é pra te ajudar a realizar todas as suas tarefas sem a necessidade de hora extra que "O Programador Produtivo" se apresenta.

Faça já o download, é GRÁTIS[3]

[2] http://bit.ly/livro-amazon-aws
[3] http://www.oprogramadorprodutivo.com.br

4. Prefácio

Todo processo evolutivo é cheio de mudanças, pra mais ou pra menos. Em se tratando de uma linguagem de programação temos ainda mais modificações. Algumas dessas modificações vêm pra corrigir bugs, outras para melhorar a performance, algumas pra facilitar a codificação, umas outras como substitutas e tem aquelas mudanças que fazem sumir recursos e funcionalidades.

Nós como programadores devemos estar sempre atentos a essas mudanças, para que possamos manter nossas aplicações com a melhor codificação, o melhor desempenho e o mais seguras possível. Não podemos olhar as mudanças de uma linguagem como sendo maléficas ao desenvolvimento, ainda mais uma linguagem open source, utilizada em larga escala e que possui uma comunidade muito ativa.

Aproveite o conteúdo desse e-book, e compartilhe-o com o maior número de pessoas possível, pois o conhecimento é pra ser compartilhado.

Bons Estudos!

– Jonathan Lamim Antunes

5. Mudanças básicas da linguagem

O PHP 7 traz consigo um grande número de pequenas mudanças na linguagem, novos operadores e funções, e mudanças em funções existentes e construtores.

Operadores

Foram adicionados 2 novos operadores ao PHP 7, o Null Coalesce e o de Comparação Combinada.

Operador Null Coalesce

Esse operador (`??`) retorna o operando à esquerda se não for nulo, caso contrário retorna o da direita. Algo interessante nesse operador é que ele não emitirá um erro se um dos operandos for uma variável não existente, algo similar à função `isset()`.

Exemplo:

```
1  //Usando o operador ternário combinado com a função isset()
2  $resultado = isset($operando) ? $operando : $operando2;
3
4  //Usando o novo operador Null Coalesce
5  $resultado = $operando ?? $operando2;
```

Os dois exemplos fazem a mesma coisa, e retornam o mesmo resultado porém, o segundo é bem mais enxuto do que o primeiro.

Operador de Comparação Combinada

Também chamado de "operador de nave espacial" (`<=>`) ele é o primeiro operador ternário do PHP. Ao invés de retornar apenas `TRUE` ou `FALSE`, ele pode retornar 3 valores distintos:

- **-1**: se o operador da esquerda for menor que o da direita
- **0**: se os operadores forem iguais
- **+1**: se o operador da esquerda for maior que o da direita

Veja no exemplo a seguir o uso desse operador e do modo tradicional, em uma função que retorna a ordem dos argumentos passados a partir da comparação entre eles.

```
1  // Antes do PHP 7
2  function modo_tradicional($a, $b) {
3    return ($a < $b) ? -1 : (($a > $b) ? 1 : 0);
4  }
5
6  // Pós PHP 7
7  function modo_php7($a, $b) {
8    return $a <=> $b;
9  }
```

O resultado final continua o mesmo, mas repare que a função ficou bem mais enxuta com o uso do operador de comparação combinada.

Constant Arrays

Antes do PHP 7, contantes definidas com `define()` só poderiam conter valores escaláveis. Com o PHP 7, você pode definir uma matriz como constante. Veja o exemplo a seguir:

```php
define('MATRIZ',['chave' => 'valor', 'chave2' => 'valor2']);

//Exibindo o valor
echo MATRIZ['chave']; //retorna valor
```

Função intdiv()

Essa nova função executa a divisão de número inteiro de forma eficaz, o inverso do operador módulo da divisão (`%`).

```php
intdiv(8, 3);
//retorna 2
```

Sessions

Agora é possível passar um `array` de configurações para a função `session_start()`.

Também foi adicionado um novo parâmetro de configuração, chamado `session.lazy_write`, que permite que os dados sejam reescritos na sessão caso tenham sido alterados. Esse parâmetro vem configurado por padrão como `TRUE`.

Veja a seguir um exemplo de como usar o `array` de configurações em `session_start()` e desabilitar o `lazy_write`.

```php
session_start(['use_strict' => true, 'lazy_write' => false]);
```

dirname()

A função `dirname()` passou a aceitar um segundo parâmetro no PHP 7, que permite definir quantos níveis de diretório acima você deseja subir. Por exemplo, temos a seguinte estrutura de diretório:

`/home/user/artigos/php`

Queremos subir 2 níveis nesse diretório:

```php
$caminho = "/home/user/artigos/php";
dirname($caminho, 2); //retorna /home/user
```

Conclusão

Essas são algumas das principais mudanças básicas da linguagem, que ao serem aplicadas aos sistemas, já fazem a diferença tanto na codificação quanto na performance.

Por menores e mais simples que sejam, utilizá-las vai fazer com que sua aplicação aproveite ainda mais do que o PHP 7 pode oferecer em termos de desempenho.

6. Recursos Obsoletos

No PHP um recurso *(feature)* é marcado como obsoleto *(deprecated)* para informar aos desenvolvedores que em uma versão futura esse recurso será removido totalmente. Com isso os desenvolvedores possuem tempo para migar suas aplicações antes que elas comecem a apresentar erro por usar recursos obsoletos.

Nos últimos releases do PHP 5.x vários recursos foram marcados como obsoletos, e esses recursos foram removidos no PHP 7.

> No PHP, sempre que um desenvolvedor faz uso de um recurso marcado como *"deprecated"*, ele recebe um erro `E_DEPRECATED`.

Tags Alternativas

Alguns desenvolvedores podem desconhecer esse recurso, mas o PHP possui tags alternativas além de `<?php ?>`, essas tags foram removidas no PHP 7. São elas:

PHP ASP Tags

```
<%
  //Seu código
%>
```

PHP script tags

```
<script language="php">
  //Seu código
</script>
```

Muitos desenvolvedores utilizam as tags estilo ASP para adicionar código PHP dentro do template, se você é um desses, aproveite o momento e passe a utilizar as tags padrões do PHP para evitar problemas futuros na sua aplicação.

Os dois códigos a seguir mostram a impressão de uma variável usando *"short tags"*. O primeiro com as tags obsoletas e o segundo com a tag padrão do PHP.

Código desatualizado

```
1  <%= $minhaVarivel; %>
```

Código atualizado

```
1  <?= $minhaVarivel; ?>
```

Switch com múltiplos "defaults"

Durante a criação das especificações do PHP (PHP language spec) foi encontrado um bug, que permitia definir múltiplas cláusulas `default` em um `switch`. E quando esse bug era explorado, somente o código da última cláusula `default` declarada era executado.

Para resolver esse problema, foi feita a correção no PHP 7 e um erro fatal *(fatal error)* é retornado sempre que o bug é explorado.

```
1  Fatal error: Switch statements may only contain one default clause
```

POSIX - Expressões regulares compatíveis

Esse recurso foi marcado como obsoleto no PHP 5.3, e removido no PHP 7. Ele era muito utilizado para comparar strings. Veja a seguir a lista das funções que foram removidas:

- ereg()

- eregi()
- ereg_replace()
- eregi_replace()
- split()
- spliti()
- sql_regcase()

Se ainda não migrou após um período de 6 anos da marcação como obsoletas, chegou a hora de migrar para a família de funções `preg_`. Veja no exemplo a seguir, o uso da função `preg_eregi()`.

```
1 //Obsoleta
2 ereg('pattern', 'string');
3
4 //Atualizada
5 preg_match('/pattern/', 'string');
```

Família preg_

- **preg_filter** — Executa uma busca usando expressão regular e substitui o conteúdo
- **preg_grep** — Retorna as entradas do array que combinaram com o padrão
- **preg_last_error** — Retorna o código de erro da última regex PCRE executada
- **preg_match_all** — Executa uma busca usando expressão regular para uma correspondência global
- **preg_match** — Executa uma busca usando expressão regular para uma correspondência
- **preg_quote** — Adiciona escape em caracteres da expressão regular
- **preg_replace_callback_array** — Executa uma busca usando expressão regular e substitui o conteúdo usando uma função de callback
- **preg_replace_callback** — Executa uma busca usando expressão regular e modifica usando um callback
- **preg_replace** — Realiza uma pesquisa por uma expressão regular e a substitui.
- **preg_split** — Divide a string por uma expressão regular

Extensão MySQL

A extensão `ext/mysql` foi marcada como obsoleta no PHP 5.5 e removida no PHP 7, e isso inclui todas as funções `mysql_`.

Se você ainda utiliza funções `mysql_`, já é hora de migrar para as funções `mysqli_`, que são parte da extensão `ext/mysqli`. Em sua maioria, as funções `mysqli_` são idênticas às `mysql_`, exceto pelo sufixo **i**.

Na maioria dos casos será necessário mudar apenas o nome da funções, e em mais ou menos 50% dos casos você precisará passar a conexão com o banco de dados como primeiro parâmetro da função.

Veja abaixo uma lista de funções `mysql_` que não possuem equivalentes em `mysqli_`.

- mysql_client_encoding()
- mysql_list_dbs() **(use a query SHOW DATABASES)**
- mysql_db_name()
- mysql_list_fields()
- mysql_db_query()
- mysql_list_processes() **(use a query SHOW PROCESSLIST)**
- mysql_dbname()
- mysql_list_tables() **(use a query SHOW TABLES)**
- mysql_field_flags()
- mysql_listdbs() **(use a query SHOW DATABASES)**
- mysql_field_len()
- mysql_listfields()
- mysql_field_name()
- mysql_listtables() **(use a query SHOW TABLES)**
- mysql_field_table()
- mysql_numfields()
- mysql_field_type()
- mysql_numrows() **(use mysqli_num_rows())**
- mysql_fieldflags()

- mysql_pconnect() **(adicione p: ao nome do host passado para mysqli_connect())**
- mysql_fieldlen()
- mysql_result()
- mysql_fieldname()
- mysql_selectdb() **(use mysqli_select_db())**
- mysql_fieldtable()
- mysql_table_name()
- mysql_fieldtype()
- mysql_tablename()
- mysql_freeresult() **(use mysqli_free_result())**
- mysql_unbuffered_query()

A seguir um exemplo de uso do `mysqli_` para obtenção de uma lista de dados na tabela *agenda*, tendo como parâmetro para a consulta o *nome*.

```
1  $nome = \filter_var($_POST['nome'], FILTER_SANITIZE_EMAIL);
2  $mysqli = new \mysqli('localhost', 'teste', null, 'agenda');
3
4  if (\mysqli_connect_errno()) {
5    // Falha na conexão
6  }
7
8  $sql = "SELECT nome, email, telefone FROM contatos WHERE nome = ?";
9  $query = $mysqli->prepare($sql);
10 $query->bind_param('s', $nome);
11 $result = $query->execute();
12
13 if (!$result) {
14   return false;
15 }
16
17 $result = $query->fetch_result();
18
```

```
while ($row = $result->fetch_object()) {
  //Interage através dos resultados retornados
}
```

Além do `mysqli_` em substituição ao `mysql_`, você pode utilizar a classe PDO, que permite conectar a uma boa variedade de bancos de dados, através de uma API consistente.

Veja a seguir um exemplo de uso da classe PDO para executar a mesma operação que foi feita no exemplo anterior, com `mysqli_`.

```
$nome = \filter_var($_POST['nome'], FILTER_SANITIZE_EMAIL);

try {
  $pdo = new \PDO("mysql:host=localhost;dbname=agenda", "teste");
} catch (\PDOException $e) {
  // Falha na conexão
}

$sql = "SELECT nome, email, telefone FROM usuarios WHERE nome = :nom\
e";
$values = [":nome" => $nome];
try {
  $query = $pdo->prepare($sql);
  $result = $query->execute($values);

  if (!$result || $query->rowCount() == 0) {
    return false;
  }

  foreach ($query->fetch(\PDO::FETCH_OBJ) as $row) {
    //Interage através dos resultados retornados
  }
} catch (\PDOException $e) {
  // Ocorreu algum erro
}
```

Conclusão

É normal que durante o processo de evolução de uma linguagem, recursos sejam adicionados e removidos. Como o PHP 7 era bastante esperado, essas mudanças e remoções acabaram ganhando certa evidência. É importante, como desenvolvedor, ficar atento à documentação de cada verão, para identificar o que foi marcado como obsoleto, e ir se preparando para atualizar o código para versões futuras.

7. Sintaxe uniforme de variáveis

A sintaxe de variáveis do PHP é de certa forma inconsistente, particularmente no que diz respeito a *variáveis-variáveis* e *variáveis-propriedades*.

Por exemplo, veja a sintaxe `$objeto->$array[chave]`, onde estamos sempre esperando que primeiro seja resolvido o `$array[chave]` para depois acessar a propriedade do objeto. Com a sintaxe uniforme de variáveis, toda essa inconsistência foi resolvida.

A partir do PHP 7, todas as variáveis passam a ser avaliadas da esquerda para a direita, sendo um *"problema"* somente em caso de variáveis dinâmicas complexas e propriedades.

Veja nos códigos a seguir que é possível obter o mesmo comportamento do PHP 5.x no PHP 7, ou vice-versa, especificando de forma explícita a ordem das operações com o uso de parênteses `()` e chaves `{}`.

```
1  // Sintaxe
2  $$var['chave']['chave2'];
3
4  // PHP 5.x - Usando um array multidimensional como nome da variável
5  ${$var['chave']['chave2']};
6
7  // PHP 7 - Acesssando um array multidimensional dentro de uma variáv\
8  el-variável
9  ($$var)['chave']['chave2'];
```

```php
// Sintaxe
$var->$prop['chave'];

// PHP 5.x - Usando um array como nome de propriedade
$var->{$prop['chave']};

// PHP 7 - Acessando um array dentro de uma variável-propriedade
($var->$prop)['chave'];
```

```php
// Sintaxe
$var->$prop['chave']();

// PHP 5.x - Usando um array como nome de um método
$var->{$prop['chave']}();

// PHP 7 - Chamando um closure dentro de um array como variável-prop\
riedade
($var->$prop)['key']();
```

```php
// Sintaxe
ClassName::$var['chave']();

// PHP 5.x - Usando um array como nome de método estático
ClassName::{$var['chave']}();

// PHP 7 - Chamando um closure dentro de um array com variável estát\
ica variable
(ClassName::$var)['chave']();
```

Um dos principais benefícios da sintaxe uniforme de variáveis é que ela permite muitas combinações de sintaxe, e muitas dessas novas combinações estão disponíveis para uso.

```
// Chama um closure dentro de um array retornando outro closure
$foo()['bar']();

// Chama uma propriedade referenciando um array literal
[$obj1, $obj2][0]->prop;

// Acessa um caracter pelo índice na string retornada
getStr(){0};
```

Além disso, o PHP 7 suporta - em alguns casos - dois pontos duplos aninhados.

```
// Acessa uma propriedade estática no nome da classe ou objeto dentr\
o de um array
$foo['bar']::$baz;

// Acessa uma propriedade estática no nome da classe ou objeto dentr\
o de um array retornando para o metódo estático chamado em um nome d\
e classe ou objeto
$foo::bar()::$baz;

// Chama um método estático em uma classe ou objeto retornado por um\
a chamada de método
$foo->bar()::baz();
```

Ainda existe uma série de casos ambíguos que não podem ser resolvido, mesmo com a nova sintaxe de variáveis, e mesmo quando há adição de parênteses e chaves.

```php
$foo = 'Foo';
$class = 'CLASS';
$constant = 'BAR';

echo $foo::$class::$constant;
echo $foo::{$class}::$constant;
echo $foo::{"$class"}::$constant;
echo $foo::{"$class"}::{"$constant"};
echo $foo::CLASS::$constant;
echo $foo::CLASS::{"$constant"};
echo $foo::($class)::($constant);
```

Além disso, agora você pode fazer chamadas de métodos e funções dobrando os parênteses.

```php
// Chama o callback retornado por um método
foo()();

// Chama um callback retornado por um método instanciado
$foo->bar()();

// Chama um callback retornado por um método estático
Foo::bar()();

// Chama um callback retornado por outro callback
$foo()();
```

Desde o PHP 5.4 era possível desreferenciar arrays retornados por métodos e funções, agora com o PHP 7 é possível fazer isso com qualquer expressão válida colocada entre parênteses.

```php
// Acessando a chave de um array
(expression)['foo'];

// Acessando uma propriedade
(expression)->foo;

// Chamando um método
(expression)->foo();

// Acessando uma propriedade estática
(expression)::$foo;

// Chamando um método estático
(expression)::foo();

// Chamando um callback
(expression)();

// Acessando um caracter
(expression){0};
```

Isso permite, finalmente, chamar um closure quando ele for definido, e chamar um callback dentro da propriedade de um objeto.

```php
// Define e chama imediatamente um closure sem atribuição
(function() { /* ... */ })();

// Chama um callback dentro da propriedade de um objeto
($obj->callable)();
```

O PHP 7 também dá a possibilidade de chamar métodos usando *array-notation*.

```
// Chama um método estático dinâmico
["className", "staticMethod"]();

// Chama uma método de instância dinâmico
[$object, "method"]();

// Usa uma string escalar como nome de classe
'className'::staticMethod();
```

Uma possibilidade futura da sintaxe de variáveis do PHP 7 é a perspectiva de trabalho com objetos escalares, ou seja, a capacidade de chamar métodos que agem diretamente sobre um valor.

```
// converte os caracteres da string para minúsculo
"string"->toLower();
```

Conclusão

Nesse capítulo você viu um pouco sobre a nova sintaxe de variáveis do PHP 7, e no próximo verá sobre *Type Hints*, a característica excitante do PHP 7.

8. Type Hints

Como se sabe, o PHP é uma linguagem fracamente tipada, que não se preocupa muito com os tipos de dados. Uma mesma variável pode receber valores diferentes a qualquer momento. Uma hora ela pode ser `string`, duas linhas depois se tornar um `array` e por aí vai.

O PHP 7 introduziu recursos de tipagem, que vão permitir definir o tipo de valor que uma variável vai receber, checando se o valor recebido é compatível com o tipo de valor especificado para a variável ou até mesmo o tipo de dados que uma função irá retornar.

No PHP 5 esse recurso já funcionava, mas somente com `arrays` e `objetos`, e no PHP 7 adicionaram o suporte a `float`, `int`, `string` e `boolean`.

Para que a verificação do tipo de dado funcione é preciso ativá-la através da opção `strict_types`.

```
1  declare(strict_types=1);
```

A seguir temos um código de uma função que imprime um texto na tela, a partir de um valor passado como parâmetro. Esse valor deve ser uma string, e caso não seja, será retornado um erro do tipo `Uncaught TypeError`, ao invés de imprimir o valor na tela.

```
1  declare(strict_types=1);
2
3  function ShowText(string $text){
4    echo 'The variable is: ' . $text;
5  }
6
7  ShowText("Type Hints PHP 7");
```

Na primeira linha ativamos a verificação de tipos, em seguida criamos a função, e logo depois chamamos a execução da função.

Repare que na declaração da função a variável recebeu o tipo de dado antes de ser declarada: `string $text`. Se o tipo não tivesse sido informado, ela aceitaria qualquer valor e não retornaria nenhum erro do tipo `Uncaught TypeError`.

Além do PHP fazer a validação do valor passado para a variável, ele também faz uma tentativa de conversão do valor passado para o tipo especificado, o que chamamos de *Coercive Type Hints*.

Veja no exemplo abaixo que a função vai receber 2 parâmetros, um inteiro e uma string, e mesmo passando a variável que deveria ser um número inteiro como string, não será retornado um erro do tipo `Uncaught TypeError`, pois o PHP vai converter a `string` em `int`.

```
1  function SendMessage(int $code, string $message)
2  {
3      echo $code. ' - ' .$message;
4  }
5
6  SendMessage(404, "File Not Found");
7  SendMessage("200", "OK");
```

Veja bem que na primeira chamada da função `SendMessage` os valores das variáveis foram passados corretamente, mas na segunda as duas variáveis foram passadas como `string`. Devendo a primeira variável ser um número inteiro `(int)`, o PHP tentou fazer a conversão, e como ela foi bem sucedida, não foi retornado nenhum erro.

Essa conversão pode causar perda de precisão nos dados e até mesmo a perda de dados. Veja um exemplo onde os parâmetros devem ser número inteiros `(int)`, mas os valores passados são decimais `(float)`.

```
1  function Sum(int $x, int $y)
2  {
3      return $x + $y;
4  }
5  add(97.234, 61.53);
```

Como o tipo de variável foi definido como `int`, o PHP vai tentar fazer a conversão, logo os valores deixarão de ser 97.234 e 61.53 para ser 97 e 61, respectivamente. O valor retornado então não será 158.764, mas sim 158.

Type hint iterable

É comum encontrar situações no desenvolvimento de aplicações com PHP onde a única cosia que se precisa em uma variável é que ela possa passar por uma iteração em um `foreach`.

No PHP 7.0 tem 2 type hints que possibilitam isso: `array` e `Traversable`. O problema é que quando você definia uma variável como sendo do tipo `array`, não era possível passar dados pra ela do tipo `Traversable` e vice-versa.

Para solucionar esse problema o PHP 7.1 traz um novo type hint chamado `iterable`, que possibilita que qualquer parâmetro ou retorno de função que foi definido como `iterable` possa passar por iterações, recebendo tanto um array ou objeto iterável.

Return Type Hints

Assim como é possível determinar o tipo de valor das variáveis e parâmetros de uma função, também é possível determinar o tipo de valor retornado por uma função. A regra é a mesma da apresentada acima para as variáveis e parâmetros, mudando somente a sintaxe de escrita da função.

```
1  function Divide(int $x, int $y): int {
2    return $x / $y;
3  }
4  Divide(6,3);
5  Divide(8,3);
```

A função acima simplesmente retorna o valor da divisão entre 2 número inteiros, sendo que o retorno deve ser um número inteiro.

Duas chamadas à função `Divide` foram feitas, na primeira são passados os valores 6 e 3, que dividos dão 2, um valor inteiro que corresponde ao tipo de valor que a função deve retornar. Já a segunda chamada, que usou os valores 8 e 3, vai retornar um erro, pois o valor da divisão não é `int`, e sim um `float` (2,66666667).

Imagine que a função exemplificada acima, cujo retorno é do tipo `int` precisasse retornar `null` caso ocorresse algum erro no processamento.

O PHP 7.0 não tem suporte para o tratamento desse tipo de situação onde o tipo já foi definido, mas no PHP 7.1 foi adicionada essa possibilidade, onde você pode determinar o tipo de retorno e ainda dar a possibilidade dele ser nulo.

Para poder retornar um tipo específico de dados ou `null`, basta adicionar um ponto de interrogação (?) antes da definição do tipo de retorno.

```
1  function Divide(int $x, int $y): ?int {
2    return $x / $y;
3  }
```

Uma nova sintaxe disponibilizada no PHP 7.1 é a de definição de um parâmetro de função com definição de tipo onde esse parâmetro pode ser opcional.

A aplicação da nova sintaxe para parâmetros opcionais é similar à utilizada na definição de tipo de função vista acima.

```
// PHP 7.0
function verificaTexto(string $textoBase, string $textoCompara = nul\
l): string{
    ...
}

// PHP 7.1
function verificaTexto(string $textoBase, ?string $textoCompara): st\
ring{
    ...
}
```

Existem aplicações que possuem funções que não retornam nada, apenas executam um processamento, e para definir que essa função terá um retorno nulo, usa-se `void`.

```
function processaDados(string $dados): void {
    ...
}
```

> Lembre-se que `void` é diferente de `null`. `void` não retorna nada, enquanto `null` está retornando um valor nulo.

Conclusão

Mesmo com essa implementação de tipagem de dados, ainda há um nível de inconsistência e perda de informação, mas de qualquer forma já é uma evolução. Vamos aguardar e ver o que futuras versões do PHP trarão de novidades e maior consistência em relação a tipagem de dados.

9. Expectations e Assertions

Desde o PHP 4 que a função `assert()` está disponível, e ela serve para adicionar *sanity-checks* no código. Ela deve ser utilizada somente para desenvolvimento, nunca em ambiente de produção, e pode ser facilmente ativada e desativada usando `assert_options()` ou `assert.active` no *php.ini*.

Para usar `assertions` você passa uma `expressão` ou `string` como primeiro parâmetro. Veja no código abaixo um exemplo, e se a `expressão` ou `string` resultar em `FALSE`, será retornado um `warning`.

```
1 assert('$user instanceof \MyProject\User');
2 assert($user instanceof \MyProject\User);
```

O `warning` que pode ser retornado é algo do tipo:

```
1 Warning: assert(): Assertion "$user instanceof \MyProject\User" fail\
2 ed in <file> on line <num>
```

Expectations

No PHP 7 o `assert()` foi expandido para Expectations RFC, permitindo fazer chamadas a assertions zero-coast, que não geram impactos sobre o desempenho. Com essa mudança, não só é possível desativar assertions mas também remover toda a sobrecarga completamente.

Nessa configuração assertions não são compiladas, independente da string ou expressão passadas, portanto, geram um impacto 0 (zero) sobre o desempenho. Isso é diferente caso assertions esteja desativada, que vai resultar em expressões que ainda serão avaliadas, afetando potencialmente a execução e irá ignorar a chamada.

Veja abaixo um exemplo de ativação de assertions:

```
1 //Ativa assertions
2 zend.assertions  = 1
3 //Disativa assertions
4 zend.assertions  = 0
5 //define zero-coast assertions
6 zend.assertions  = -1
```

Conclusão

Com a adição de *assertions zero-coast*, você finalmente terá uma maneira leve para adicionar *sanity-check* ao seu código durante o desenvolvimento, sem afetar a produção. Assertions podem ser um primeiro passo no caminho para a adição de teste unitários para a sua aplicação.

10. Error Handling

Desde o PHP 4 que a manipulação de erros esteve inalterada, exceto pela adição de `E_STRICT` no PHP 5, `E_RECOVERABLE_ERROR` no PHP 5.2 e `E_DEPRECATED` no PHP 5.3.

Antes do PHP 7, se uma classe não fosse instanciada corretamente, um valor `null` ou `object` era retornado. Agora, todas as classes vão lançar uma *exceção* no caso de falha no instanciamento, variando o tipo e a razão da falha.

```
try {
    new MessageFormatter('pt_BR', null);
} catch (\IntlException $e) {

}
```

O fragmento de código acima vai resultar em uma `\IntlException` com a mensagem *Contructor failed.*

Engine Exceptions

Com o PHP 7, quase todos os erros fatais e capturáveis serão tratados como *engine exceptions*. Isso é possível porque uma exceção não capturada ainda resulta em um erro fatal.

Com essa mudança temos alguns benefícios:

- Objetos `__destruct()` são chamados
- Callbacks registrados com `register_shutdown_function()` são chamados
- Assim como acontece com todas as exceções, haverá um rastreamento de pilha, tornando mais fácil a depuração

Error Exceptions

O PHP 7 traz novos tipos de *error exceptions*, cada um com um propósito diferente. Vejamos alguns deles.

AssertionError

Com as melhorias feitas para uso de assertions, se você informa `assert.exception` como 1 no *php.ini*, uma exceção será lançada quando a declaração falhar.

```
1 try {
2     ini_set('assert.exception', 1);
3     assert('true === false', 'Assertion failed');
4 } catch (\AssertionError $e) {
5
6 }
```

O código acima vai resultar em uma exceção com a mensagem passada como segundo parâmetro para a função `assert()`, *Assertion failed*.

> Se você não informar o segundo parâmetro para a função `assert()`, a exceção `\AssertionError` não terá nenhuma mensagem. Ainda sim será possível obter informações como a origem e a informação de que a afirmação falhou.

TypeError

Agora que o PHP permite definir o tipo de variável e retorno de métodos e funções, estes também lançarão exceções informando sobre os erros.

```
function example(callable $callback)
{
   return $callback()
}

try{
  example(new stdClass);
}catch(\TypeError $e){

}
```

O código acima vai resultar em uma exceção com a mensagem `Argument 1 passed to example() must be callable object given, called in <file> on line <num>`, pois foi passado como parâmetro uma classe e não uma função.

ArithmeticError e DivisionByZeroError

Essas duas exceções foram adicionadas para poder lidar com cálculos aritméticos. A primeira, `\ArithmeticError`, será lançada sempre que ocorrer um erro ao executar operações matemáticas. A segunda, `\DivisionByZeroError`, é mais específica, e vai ser lançada sempre que houver uam tentativa de divisão por zero.

```
try{
  1 >> -1;
}catch(\ArithmeticError $e){

}
```

O código acima vai resultar em uma exceção com a mensagem *Bit shift by negative number*.

```
try{
  10 % 0
}catch(\DivisionByZeroError $e){

}
```

O código acima vai resultar em uma exceção com a mensagem *Modulo by zero*, pois há uma tentativa de dividir um valor por zero.

ArgumentCountError Exception

Versões anteriores ao PHP 7.1 permitiam que funções fossem chamadas sem que seus parâmetros fossem informados, e então era emitido um warning informando a falta dos parâmetros.

```
function sum($a, $b)
{
    return $a + $b;
}

sum();
// Warning: Missing argument 1 for sum()
// Warning: Missing argument 2 for sum()

sum(5);
// Warning: Missing argument 2 for sum()

sum(5, 8);
```

Warning não se aplicam a esse caso, pois se o parâmetro é obrigatório, logo a falta dele gera erro de execução. Para resolver isso, no PHP 7.1 foi implementada a exception **ArgumentCountError**.

Caso ocorra o problema citado acima, será lançado um *Fatal Error* e não mais um *Warning*.

```php
function sum($a, $b)
{
    return $a + $b;
}

sum();
// Fatal error: Uncaught ArgumentCountError: Too few arguments to fu\
nction sum(), 0 passed in /index.php on line 6 and exactly 2 expecte\
d in /index.php:1

sum(5);
// não executado

sum(5, 8);
// não executado
```

Agrupamento de catch

Até o PHP 7.0 era necessário usar código duplicado para fazer o tratamento de exceptions de diferentes tipos.

```php
try {
    ...
} catch (PDOException $e) {
    save_log($e->getMessage());
} catch (RuntimeException $e) {
    save_log($e->getMessage());
} catch (Exception $e) {
    ...
}
```

A partir do PHP 7.1 você pode otimizar esse processo agrupando todos os `catch` que possuírem o mesmo tratamento para o erro, basta separá-los por uma barra vertical (|).

```php
try {
    ...
} catch (PDOException | RuntimeException $e) {
    save_log($e->getMessage());
} catch (Exception $e) {
    ...
}
```

Conclusão

Essas novas exceções que vieram com o PHP 7 vão ter grande impacto na forma como você escreve o seu código, mas se você não estiver utilizando `set_error_handler()`, então não terá a necessidade de fazer alterações na sua aplicação para que ela continue funcionando.

11. Mudanças relacionadas a POO

O PHP 5 e seus sucessores trouxeram várias alterações relacionadas a POO, já o PHP 7 quase não trouxe muitos recursos novos nesse sentido, mas mesmo sendo poucos, os que vieram são bastante interessantes.

Context-Sensitive lexer

Permite o uso de palavras-chave como nome de propriedades, métodos e constantes dentro de classes, interfaces e traits.

Isso quer dizer que o PHP vai ter apenas uma palavra-chave reservada, `class`, e somente no contexto de constante da classe.

Veja abaixo a lista de palavras-chave que você poderá utilizar como propriedade, função e nomes de constantes:

```
callable, and, include, function, trait, global, include_once, if, extends, goto, throw, endswitch, implements, instanceof, array, finally, static, insteadof, print, for, abstract, interface, echo, foreach, final, namespace, require, declare, public, new, require_once, case, protected, or, return, do, private, or, else, while, const, try, elseif, as, enddeclare, use, default, catch, endfor, var, break, die, endforeach, exit, continue, self, endif, list, switch, parent, endwhile, clone, yield, class
```

A única exceção é não poder usar `const class`, pois entra em conflito com o nome da constante mágica do PHP 5.5.

Construtores descontinuados do PHP 4

Ao invés de remover os construtores do PHP 4 - que estava sendo pedido por muitos na comunidade - eles foram marcados como obsoletos *(deprecated)* no PHP 7.

Remover esses construtores causaria um problema de compatibilidade com códigos escritos para aplicações em versões anteriores do PHP. Caso venham a ser removidos, isso acontecerá no PHP 8, o que não vai afetar as aplicações e desenvolvedores por um bom tempo.

A utilização dos construtores do PHP 4 foi anulada quando os *namespaces* das classes foram introduzidos no PHP 5.3.

Declarações de grupo

As declarações de grupo são o resultado de um esforço para reduzir a duplicação e simplificar o código escrito. Elas permitem remover a duplicação de prefixos comuns, especificando partes únicas dentro de um bloco `{}`.

Veja nos códigos abaixo uma exemplo de como essas declarações de grupo podem ser utilizadas:

Usando 'group use statements'

```
1  // Original
2  use Framework\Component\ClassA;
3  use Framework\Component\ClassB as ClassC;
4  use Framework\OtherComponent\ClassD;
5
6  // Com group use statements
7  use Framework\{
8       Component\ClassA,
9       Component\ClassB as ClassC,
10      OtherComponent\ClassD
11 };
```

Organização alternativa de 'group use statements'

```
use Framework\Component\{
     ClassA,
     ClassB as ClassC
};
use Framework\OtherComponent\ClassD;
```

Veja que em ambos os códigos as declarações repetidas foram sendo agrupadas, de forma a reduzir o conteúdo duplicado das declarações.

Se você quiser importar funções ou constantes - *feature* que foi adicionada no PHP 5.6 - você deve simplesmente adicionar o prefixo com a função ou constante. Veja a seguir um exemplo de código com uso desse recurso:

```
use Framework\Component\{
     SubComponent\ClassA,
     function OtherComponent\someFunction,
     const OtherComponent\SOME_CONSTANT
};
```

Classe Anônima

Classes anônimas, provavelmente, são o mais perto que chegaremos de uma sintaxe literal de objetos no PHP.

Para criar uma classe anônima você simplesmente combina a nova classe, seguida de uma definição de classe padrão.

A classe anônima é sempre instanciada durante a criação, dando-lhe um objeto dessa classe.

```php
$object = new class("bar") {
    public $foo;

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

O código acima cria um objeto com um método construtor `__construct()`, chamado com o argumento `bar` e com a propriedade `$foo`, atribuindo o valor do argumento ao construtor.

```
object(class@anonymous)#1 (1) {
  ["foo"]=>
  string(3) "bar"
}
```

Classes anônimas podem ser *namespaced*, oferecendo suporte a herança, traits e interfaces, usando a mesma sintaxe de classes regulares.

```php
namespace MyProject\Component;

$object = new class ($args) extends Foo implements Bar {
        use Bat;
};
```

Nomes de classes anônimas

Pode parecer estranho nomear uma classe anônima, afinal, ela é anônima. O nome de uma classe anônima é baseado no endereço de memória da operação que a criou, por exemplo: class@0x7fa77f271bd0

Imagina que estivesse criando classes anônimas dentro de um loop, a classe teria o mesmo endereço de memória de cada iteração desse loop, o que faria com que somente uma classe fosse definida.

Isso significa que se o objeto resultante de duas iterações têm o mesmo valor de propriedades, elas serão iguais (== mas não idênticas ===).

No entanto, mesmo se você definir outra classe anônima com a mesma estrutura, em outro lugar do código, ela terá um nome diferente com base em seu endereço de memória, e então não será igual.

Veja no código abaixo como fica, na prática, o que foi falado sobre o nome de classes anônimas:

```
1  $objects = [];
2  foreach (["foo", "foo", "bar"] as $value) {
3          $objects[] = new class($value) {
4
5              public $value;
6
7              public function __construct($value)
8              {
9                  $this->value = $value;
10             }
11         };
12 }
13
14 $objects[] = new class("foo") {
15
16     public $value;
17
18     public function __construct($value)
19     {
20         $this->value = $value;
21     }
22 };
```

No exemplo foram criadas 3 instâncias de uma classe anônima dentro do loop

`foreach`. Os dois primeiros são passados como `foo` ao construtor, e o terceiro é passado como `bar`. Fora do loop foi criada uma quarta classe anônima, com a mesma definição, e novamente foi passado `foo` para o construtor.

Devido a isso, os 2 primeiros objetos - `$objects[0]` e `$objects[1]` respectivamente - são iguais mas não idênticos. No entanto, nenhum desses 2 objetos é igual ao terceiro - `$objects[2]` - e também não serão iguais ao quarto - `$objects[3]` - já que esse foi definido fora do loop, e apesar de ter estrutura e valores idênticos, é uma classe diferente, com um nome diferente, pois está alocada em outro endereço de memória.

Conclusão

Enquanto o foco do PHP 7 não é de alterar muito o modelo de dados, as alterações adicionadas são substanciais, classes anônimas abre uma série de novas possiblidades de arquitetura. Essas mudanças tornarão o código orientado a objeto mais robusto e fácil de ser escrito.

12. A extensão “Request”

Fazer uso das superglobais `$_GET`, `$_POST` e `$_FILES` em aplicações PHP é um caso comum para muitos programadores, e isso acaba tornando o processo de codificação bastante trabalhoso e massivo.

Além dessas superglobais, com certeza você já teve problemas com o uso de `header()`, `setcookie()` e outras funções, tendo dificuldade para inspecionar o código e localizar a raiz do problema.

Para resolver esses problemas, Paul M. Jones e John Boehr desenvolveram a extensão *Request*, onde você consegue obter *ServerRequest* e *ServerResponse*, como se essas estivessem sendo fornecidas pelo próprio PHP.

ServerRequest

ServerRequest é um objeto que representa as respostas recebidas pelo servidor, e pode substituir `$_GET`, `$_POST` entre outras superglobais.

Com o uso de *ServerRequest* você tem as seguintes possibilidades:

- uso das superglobais sem sessão e com propriedade somente leitura
- propriedades adicionais, com propriedade somente leitura, a partir das superglobais
- retenção imutável de informações específicas da aplicação
- extensibilidade

Fazendo uso de *ServerRequest*

Após instalada a extensão, você precisa apenas instanciar a classe e então fazer uso de seus recursos, totalmente orientados a objeto, para poder utilizar as superglobais através da extensão.

```php
$request = new ServerRequest();
```

Veja a seguir um exemplo de como fazer uso da extensão *Request* para verificar o *contentType* da requisição.

```php
$request = new ServerRequest();
if ($request->contentType == 'application/json') {
    $input = json_decode($request->content, true);
    $request = $request->withInput($input);
}
```

O método `$request->withInput()` cria um clone da instância *ServerRequest* sem modificar o valor da propriedade na instância que foi chamada.

> Todas as propriedades de *ServerRequest* são somente leitura.

Propriedades de ServerRequest

Veja a seguir as propriedades de *ServerRequest.*

Relacionadas às superglobais

- `$env`: cópia de `$_ENV`
- `$files`: cópia de `$_FILES`
- `$get`: cópia de `$_GET`
- `$cookie`: cópia de `$_COOKIE`
- `$post`: cópia de `$_POST`
- `$server`: cópia de `$_SERVER`
- `$upload`: cópia de `$_FILES` complementada com informações de `$_POST`

Relacionadas a HTTP

- `$accept`: um array com os valores de `$_SERVER['HTTP_ACCEPT']`

- `$acceptCharset`: um array com os valores de `$_SERVER['HTTP_ACCEPT_CHARSET']`
- `$acceptEncoding`: um array com os valores de `$_SERVER['HTTP_ACCEPT_ENCODING']`
- `$acceptLanguage`: um array com os valores de `$_SERVER['HTTP_ACCEPT_LANGUAGE']`
- `$forwarded`: um array com os valores de `$_SERVER['HTTP_FORWARDED']`
- `$forwardedFor`: um array com os valores de `$_SERVER['HTTP_FORWARDED_FOR]`, e seu valor é obtido a partir da conversão da string separada por vírgulas
- `$forwardedHost`: valor de `$_SERVER['HTTP_FORWARDED_HOST']`
- `$forwardedProto`: valor de `$_SERVER['HTTP_FORWARDED_PROTO']`
- `$headers`: um array chave/valor obtido de `$_SERVER` para todas as chaves `HTTP_*`
- `$method`: valor de `$_SERVER['REQUEST_METHOD']` ou de `$_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE']` quando apropriado
- `$xhr`: valor booleano que indica uma *XmlHttpRequest*

Relacionadas a conteúdo

- `$content`: valor de `file_get_contents(php://input)`
- `$contentCharset`: valor de `$_SERVER['CONTENT_TYPE']`
- `$contentLength`: valor de `$_SERVER['CONTENT_LENGTH']`
- `$contentMd5`: valor de `$_SERVER['HTTP_CONTENT_MD5']`
- `$contentType`: valor de `$_SERVER['CONTENT_TYPE']`

Relacionadas a autenticação

- `$authDigest`: um array com os valores de `$_SERVER['PHP_AUTH_DIGEST']`
- `$authPw`: valor de `$_SERVER['PHP_AUTH_PW']`
- `$authType`: valor de `$_SERVER['PHP_AUTH_TYPE']`
- `$authUser`: valor de `$_SERVER['PHP_AUTH_USER']`

Relacionadas a aplicação

As propriedades a seguir são imutáveis e somente leitura. Para mudar seus valores é necessário iniciar uma nova instância de *ServerRequest*.

- `$input`: é um JSON com o conteúdo da requisição
- `$params`: é o típico *path-info* ou parâmetros de roteamento
- `$url`: é o resultado do parse executado nas chaves *HTTPS, HTTP_HOST/SERVER_NAME, SERVER_PORT e REQUEST_URI* da superglobal `$_REQUEST`

Métodos

- `withInput(mixed $input)`: define o valor para `$input`.
- `withParam(mixed $key, mixed $val)`: define uma chave/valor para `$params`.
- `withParams(array $params)`: define o valor de `$params`.
- `withoutParam(mixed $key)`: remove uma chave específica em `$params`.
- `withoutParams([array $keys = null])`: remove múltiplas chaves em `$params`.
- `withUrl(array $url)`: define o valor do array `$url`.

ServerResponse

ServerResponse é um objeto que representa as respostas enviadas para o servidor, e pode substituir `header()`, `setcookie()` entre outras funções.

Com o uso de *ServerResponse* você tem as seguintes possiblidades:

- inspecionar *cookies, cabeçalhos (headers) e status* antes do envio
- método auxiliar para construir strings de dados HTTP
- método auxiliar para construir sequências de cabeçalho com vírgula e ponto-e-vírgula
- especificar conteúdo como download usando os cabeçalhos apropriados
- especificar conteúdo como JSON usando os cabeçalhos apropriados
- auto-envio

- mutabilidade
- extensibilidade

O objeto *ServerResponse* não possui propriedades públicas.

Fazendo uso de ServerResponse

O seu uso é muito semelhante ao de *ServerRequest*, veja no exemplo a seguir:

```
1  $response = new ServerResponse();
2
3  $response->setHeader('Cache-Control', [
4      'public',
5      'max-age' => '50',
6      's-maxage' => '100',
7      'no-cache',
8  ]); // Cache-Control: public, max-age=123, s-maxage=456, no-cache
9
10 $response->setHeader('content-type', [
11     'text/plain' => [
12         'charset' => 'utf-8'
13     ],
14 ]); // content-type: text/plain;charset=utf-8
15
16 $response->setHeader('X-Whatever', [
17     'foo',
18     'bar' => [
19         'baz' => 'dib',
20         'zim',
21         'gir' => 'irk',
22     ],
23     'qux' => 'quux',
24 ]); // X-Whatever: foo, bar;baz=dib;zim;gir=irk, qux=quux
```

Métodos de ServerRequest

ServerRequest possui alguns métodos públicos, que são:

HTTP Version

- `setVersion($version)`: define a versão do HTTP para a resposta
- `getVersion()`: recupera a versão do HTTP da resposta

Status Code

- `setStatus($Status)`: define o status do código de resposta HTTP, o equivalente a `http_response_code($status)`
- `getStatus()`: recupera o status do código de resposta HTTP

Headers

- `setHeader($label, $value)`: reescreve informações do cabeçalho HTTP, equivalente a `header("$label: $value", true)`
- `addHeader($label, $value)`: adiciona informações ao cabeçalho HTTP, equivalente a `header("$label: $value", false)`
- `getHeader($label)`: recupera o valor de um cabeçalho específico
- `getHeaders()`: recupera o array de valores do cabeçalho enviado
- `date($date)`: retorna a data formatada no padrão da RFC 1123, e o parâmetro `$date` pode ser uma string date-time ou um objeto DateTime

Cookies

- `setCookie(...)`: equivalente a `setcookie()`, com os mesmos parâmetros
- `setRawCookie(...)`: equivalente a `setrawcookie()` com os mesmos parâmetros
- `getCookies()`: retorna o array com os cookies enviados

Content

- `setContent($content)`: define o conteúdo da resposta, podendo ser uma string, objeto ou outro tipo de dado
- `setContentJson($value, $options = 0, $depth = 512)`: método que pode substituir `json_encode($value)` na definição de um conteúdo JSON a ser entregue ao usuário, já adicionando `content-type: application/json` ao cabeçalho
- `setContentDownload($fh, $name, $disposition = 'attachment', array $params = [])`: método para definir o conteúdo a ser disponibilizado para download, já preparando e adicionado as devidas informações ao cabeçalho da requisição
- `getContent()`: retorna o conteúdo da resposta

Sending

- `send()`: faz o envio da resposta para versão, status, cabeçalhos e cookies utilizando funções nativas do PHP, como `header()`, `setcookie()` e `setrawcookie()`.

Conclusão

O uso da extensão *Request* vai facilitar muito quando houver a necessidade de trabalhar com as variáveis superglobais do PHP, pois você vai poder fazer uso das mesmas variáveis, porém em formato de objeto.

Mais informações sobre a extensão podem ser vistas em https://pecl.php.net/package/request e o código-fonte no repositório oficial, em https://gitlab.com/pmjones/ext-request.

13. O que o PHP 7.1 trouxe de novidades

No capítulo 4 foram apresentadas novidades relacionadas a type hints e no capítulo 6 sobre o tratamento de `Exceptions`.

Nesse capítulo serão apresentadas outras novidades bastante interessantes que o PHP 7.1 trouxe consigo.

Índices negativos em strings

Como se sabe o PHP possui funções que permitem manipular strings através de indíces, como se a string fosse um array.

Se você quer obter um único caracter na string, pode utilizar a função `substr($palavra, 1)`, que nesse caso retornará o segundo caracter da string passada através da variável `$palavra`. Caso precisasse saber o último caracter, bastaria utilizar -1. Mas e se você precisasse retornar o penúltimo caracter?

No PHP 7.0 teria que fazer algo como:

```
1  $palavra = 'escritor';
2
3  $penultimo = $palavra[strlen($palavra) - 2];
4  //ou
5  $penultimo = substr($palavra, -2, 1);
```

Com o PHP 7.1 você só precisa fazer isso:

```php
$palavra = 'escritor';

$penultimo = $palavra[-2];
```

Outras funções para o uso com strings foram atualizadas para suportarem o uso de valores negativos para os índices. Veja algumas delas a seguir:

- `file_get_contents`
- `iconv_strpos`
- `mb_strpos`
- `mb_stripos`
- `mb_strimwidth`
- `strpos`
- `stripos`
- `substr_count`

Especificação de visibilidade para constantes de classe

Até o PHP 7.0 só era possível fazer uso de constantes de classe que fossem públicas, mas no PHP 7.1 é possível definir constantes de classe como sendo `public`, `protected` ou `private`.

Caso a visibilidade de uma constante de classe não seja definida, ela será pública.

Modificações na variável $this

O PHP 7.1 traz modificações na variável `$this` que a tornam mais seguro e acabam com problemas de ambiguidade.

Veja a seguir uma lista do que **NÃO É** mais permitido em relação à variável `$this`:

- que seja utilizado parâmetro chamado `$this` em funções

- a criação de atributos estáticos chamados `$this`
- importação do valor de `$this` em escopo usando o comando `global`
- usar `$this` como valor capturado de iteração em um `foreach`
- atribuição de valores para a variável `$this` fazendo uso de variável com nome variável
- atribuição de valores para a variável `$this` a partir de uma outra variável que faz referência a `$this`
- criação de variáveis com nome `$this` a partir de funções ue criam variáveis
- criação de variável `$this` fora do escopo de classe

Algumas outras mudanças foram:

- o método `__call` exibe corretamente o valor da variável `$this`
- a variável `$this` não terá o seu valor retornado através da função `get_defined_vars`

Float mais preciso

No PHP 7.1 passa-se a utilizar o "modo zero" para arredondar valores `float`, de modo que a diretiva `serialize_precision` passar a ter como valor default -1.

A função `json_encode` também sofreu mudanças para que melhore a precisão no número de casas decimais em valores float, passando de 14 para 17. Ao invés de seguir utilizando o tipo de precisão EG, agora ela usa PG.

Módulo Curl

O módulo Curl ganhou 2 melhorias, uma delas é o suporte ao *Server Push* do HHTP/2, e a outra é a criação de funções para que cada um dos tipos de resource criado pelo módulo curl atraveś de `curl_init`, `curl_multi_init` e `curl_share_init` possam ter suas próprias funções de modo que possam obter o código do último erro, assim como da última mensagem de erro.

Veja a seguir as funções que foram criadas:

- `curl_multi_errno`
- `curl_share_errno`
- `curl_share_strerror`

Strings Aritméticas

No PHP 7.1 não é mais possível executar cálculo aritmético se o valor for passado como string. Por exemplo:

```
1 var_dump('2' + '2'); // int(4)
2 var_dump('2' + 'string');
3 //Warning: A non-numeric value encountered in index.php on line 2
4 //int(1)
```

Até o PHP 7.0 não era exibido o **warning**.

14. Continue estudando

O conteúdo sobre PHP 7 não termina aqui, há muito a ser aprendido, e você pode encontrar outros conteúdos sobre o assunto nos links abaixo:

- Blog da JLamim[1]
- Blog da Umbler[2]
- Oficina da Net[3]
- iMasters[4]
- PHP.net[5]
- A semana PHP[6]

[1] http://blog.jlamim.com.br
[2] http://blog.umbler.com
[3] http://www.oficinadanet.com.br
[4] http://www.imasters.com.br
[5] http://php.net/manual/pt_BR/migration70.php
[6] http://www.asemanaphp.com.br