PHP 8 - PARTE 3

Prof. Ruan Carlos Martins

19/08/2022

STATIC

A palavra chave static pode ser usada para declarar propiedades e métodos de uma classe que podem ser acessados sem que seja necessário criar um objeto a partir dessa classe.

```php
<?php
class Teste
{
    //instance members - um membro por cada objeto criado a partir da classe
    public $nome;
    function teste(){
        //código aqui
    }

    //static ou class members - apenas existentes uma vez na classe
    static $idade;
    static function mover(){
        //código aqui
    }
}
```

STATIC

Como se fazem as implementações ?

```php
<?php
class Operacoes
{
    static $valor1;
    static $valor2;

    static function adicionar()
    {
        return self::$valor1 + self::$valor2;
        //return Operacoes::$valor1 + Operacoes::$valor2; //Alternativa
    }
}


Operacoes::$valor1 = 10;
Operacoes::$valor2 = 20;
echo Operacoes::adicionar();
```

STATIC

UM EXEMPLO PRÁTICO DO USO DE STATIC

```php
<?php

class Operacoes
{
    static function numeroAleatorio($min, $max){
        return rand($min,$max);
    }

    static function calcularFormula($a,$b){
        //(A x 2) + (B + A)
        return ($a*2)+($b+$a);
    }
    static function criarUmNome(){
        $nomes = ['Ruan','Raissa', 'Tiago'];
        $apelidos = ['Martins', 'Alfonco', 'Carvalho'];
        return $nomes[rand(0,count($nomes)-1)]. ' ' . $apelidos[rand(0,count($a
    }
}
```

STATIC

UM EXEMPLO PRÁTICO DO USO DE STATIC

```php
<?php

// Continuação do exemplo anterior

echo Operacoes::numeroAleatorio(0,100);
echo '<br>';
echo Operacoes::calcularFormula(10,20);
echo '<br>';
echo Operacoes::criarUmNome();
```

CONST, DEFINE E DEFINED

CONSTANTS

```php
<?php
/* As constantes são variáveis cujo valor atribuído na
sua definição não pode ser alterado ao longo
do script do PHP.
O PHP permite definir constantes de duas formas:
com o termo const e como o método define.
*/

//CONST
/* O const é usado para definir constantes no
contexto de classes.
Ao contrário das propiedades public, protected ou
privade, as propiedades const estão sempre visíveis
(são públicas), como não podem ser alteradas no
valor, não existe o risco de ficarem expostas.
*/
/* Para definir uma constante, o nome deve ser sempre maiúsculo,
podendo ser usados __
Não é necessário o sinal do dollar como nas
variáveis normais.
*/
```

CONST, DEFINE E DEFINED

CONSTANTS

```php
 <?php
class  Circulo
{
    const PI = 3.14;
    //Todas as constantes têm valor atribuído sempre que são definidas
}

// é possível apresentar sem instanciar
echo Circulo::PI;
echo '<br>';

//ou com instanciação
$c = new Circulo();
echo $c::PI;
```

CONST, DEFINE E DEFINED

O termo const não deve ser aplicado a variáveis locais ou a parâmetros. Desde o PHP 5,3 o termo const pode ser usado para criar constantes globais. Essas constantes são definidas no escopo global do script e ficam automaticamente acessíveis.

```php
<?php
const APP_NAME = 'Minha Aplicação';
echo APP_NAME;

echo '<br>';

//dentro de uma função
function teste(){
    echo APP_NAME;
}
teste();

/* Não é possível concatenar constantes da mesma
forma que fazemos com variáveis
*/
echo "<br>Nome: {APP_NAME}";
```

CONST, DEFINE E DEFINED

A função define permite definir constants globais e locais, mas não permite ser usada dentro do contexto de uma classe.

```php
<?php
define('APP_NAME','Minha aplicação');
define('VERSAO', '1.0.0');
define('MOSTRAR_ERROS', true);
define('PI', 3.14);

echo APP_NAME;
echo '<br>';

/* Por norma devemos definir os nomes das
constantes sempre com maiúscula.
Em versões anteriores da linguagem era
possível usar um terceiro parâmetro para
indicar que a constante era case insensetive.
Desde o PHP 7.3 esta opção já não é mais válida.
*/
define('CONSTANTE', 100, true);
echo CONSTANTE . '<br>';
echo constante . '<br>';
```

CONST, DEFINE E DEFINED

Para verificarmos se uma constante já existe

```php
 <?php
if(!defined('APP_NAME')){
    define('APP_NAME', 'Minha App');
}
echo APP_NAME;
echo '<br>';

//ou mais comum ainda...
defined('CONSTANTE') or define ('CONSTANTE', 'valor');
// Com o PHP 5.6 passou a ser possível definir uma constante com um array

const NOMES = ['Ruan', 'Ana', 'Joao'];
echo NOMES[0];
echo '<br>';

//Com PHP 7, passou a ser possível, usar arrays no define
define('NAMES', ['Ruan','Ana', 'Joao']);
echo NAMES[2];
```

CONSTANTES MÁGICAS

As constantes mágicas são 8 e são designadas assim porque o seu valor varia automaticamente dependendo onde estão a ser usadas. Vamos perceber como funcionam.

```php
<?php
echo __LINE__ . '<br>';//indica o número da linha de código na script.
echo __FILE__ . '<br>';//identifica o caminho completo do script.
echo __DIR__ . '<br>';//identificar a pasta onde o script está alojado

teste();
function teste(){
    $a = true;
    echo __FUNCTION__ . '<br>'; //indica o nome da função
}
```

CONSTANTES MÁGICAS

```php
<?php

class MinhaClasse
{
    function identificar(){
        echo __CLASS__ . '<br>';//indica o nome da classe
        echo __METHOD__ . '<br>';//indica o nome do método
    }
}

$a = new MinhaClasse();
$a->identificar();

// __TRAIT__ Está relacionado com um mecanismo de reutilização de código.

echo __NAMESPACE__; //indica o nome do namespace atual
```

CLASSES ABSTRATAS

Uma classe abstrata é contruída por uma implementação parcial a partir das quais outras classes podem crescer.

```php
<?php
/* Quando uma classe é declarada como abstrata,
isso significa que ela tem métodos incompleto
que, obrigatoriamente têm que ser implementados
nas classes que a herdam.

As classes abstratas não podem ser instanciadas.
Servem apenas para poderem ser herdadas por outras
classes.
*/
abstract class Forma
{
    public $largura = 100;
    public $altura = 200;

    abstract public function area();

    function altura(){
        return $this->altura;
    }
```

CLASSES ABSTRATAS

```php
<?php

abstract class Forma
{
    public $largura = 100;
    public $altura = 200;

    abstract public function area();

    function altura(){
        return $this->altura;
    }
}
//$quadrado = new Forma(); # isto não é permitido.
```

CLASSES ABSTRATAS

```php
<?php
class Retangulo extends NumberFormatter
{
    public function area(){
        return $this->largura * $this->altura;
    }
}
$r = new Retangulo();
echo $r->area();
echo '<br>';

/* Uma classe abstrata permite então uma implementação
parcial de métodos e a definição de um modelo
de implementação de outros métodos.
Isto permite em modelos de programação orientada
a objetos, juntamente com outro mecanismo, designado
por interfaces, a estruturação do código que segue os
melhores padrões de escrita.
É uma matéria fundamentalmente destinada a quem
constrói código para ser implementado por
outros programadores.
*/
```

TRAITS E A SUA UTILIDADE

São um grupo de métodos que podem ser inseridos dentro de classes. Foram adicionados à linguagem na sua versão 5.4 para aumentar a reutilização de código.

```php
<?php

/* Os traits são definidos com expressão trait,
seguido pelo nome e por um bloco de código.
As regras para dar nomes a traits são as mesmas
que usamos nas classes.*/

trait MinhasHabilidades
{
    public function falar($mensagem){
        echo "Eu digo: $mensagem";
    }

    public function saltar($metros){
        echo "Eu salto $metros metros.";
    }
}
```

TRAITS E A SUA UTILIDADE

```php
<?php

/* As classes que necessitarem de usar estes
métodos do trait, apenas terão que ter o
seguinte:
*/

class Humano
{
    use MinhasHabilidades;
}
$h = new Humano();
$h->falar('Olá Mundo!!');
echo '<br>';
$h->saltar(3);
```

INTRODUÇÃO AO INCLUDE

A grande maioria dos projetos de programação vão sempre necessitar de ter o seu código "partido"entre diferentes ficheiros. Salvo raras exceções, um projeto terá dezenas ou centenas de ficheiros para que a aplicação funcione. É aqui que entram os mecanismos de importação de scripts dentro de outros scripts. A importação pode ser feita recorrendo ao uso da instrução include.

```php
 <?php
 /*
NOTA: o include, à semelhança de echo, são
construções especiais do PHP e não requerem
a utilização de parênteses.
*/
/*Esta instrução vai inserir o código de programação
do script config.php dentro do script atual.
*/
//configuração em outro aquivo.php
define('NOME_APLICAÇÃO', 'A minha aplicação');
//----------------
include 'config.php';
echo NOME_APLICACAO;
```

INTRODUÇÃO AO INCLUDE

CAMINHO PARA O INCLUDE

```php
<?php
/*
Podemos "incluir" um script dentro de outro usando:
1. nenhum caminho e apenas o nome do script a ser incluído
2. um caminho relativo ao script atual;
3. um caminho absoluto - que indica a localização
exata do script no filesystem.
*/
//1. apenas quando o script a incluir está na mesma pasta do script atual.
//2. caminho relativo à pasta do script atual

include 'inc/dados.php';
echo '<br>';
echo $nome;
```

CAMINHO PARA O INCLUDE

```php
<?php

/*Quando necessário "andar para trás" na árvore
de pastas usamos ../
*/

include '../outro_script.php';
echo '<br>';
echo $valor;

//3. caminho absoluto

include 'C:\laragon\www\include_require\1\inc\dados2.php';
echo '<br>';
echo $home->format('d-m-Y');

// em outro aquivo
//---------------
$hoje = new DateTime();
```

UTILIZAÇÃO DO REQUIRE

O require é muito semelhante ao include. As mesmas regras de definição dos caminhos são aplicadas. A diferença entre o include e o require é que, no caso do include falhar, o código avança com um aviso. No caso do require falhar, a aplicação termina com um erro.

```php
<?php
Além do include, existem ainda outras 3
formas de importação de scripts:
    require
    include_once
    require_once

    require 'teste.php';
    echo 'momento 2';

/* como escolher entre os dois tipos de intrução ?
É aconselhável o uso do require, uma vez que,
acontecendo um erro de importação do script,
a aplicação não irá avançar com erros.
*/
```

INCLUDE ONCE, REQUIRE ONCE

A expressão *includeonce* funciona exatamente como o include, no entanto se o script já foi anteriormente incluído, o PHP não o volta a incluir.

```php
<?php

include_once 'config.php';
echo 'AAA';
include_once 'confing.php';
/* Esta linha não vai carregar novamente o mesmo script */
echo 'BBB';
/*
No caso de require_once, funciona exatamente
como o require, mas com o mesmo comportamento
do include_once: se o ficheiro já foi incorporado
anteriormente, não voltará a ser incorporado.
*/
require_once 'config.php';
echo 'CCC';
require_once 'config.php';
echo 'DDD';
```

INCLUDE ONCE, REQUIRE ONCE

É possível executar uma expressão de retorno dentro de um script importado.

```php
<?php
$paises = require_once 'dados.php';
echo 'Países';
echo '<pre>';
print_r($paises);

//OU
echo 'Países';
echo '<hr>';
foreach($paises as $pais){
    echo $pais . '<br>';
}
//Outro ficheiro de dados (dados.php)
return {
    'BRASIL',
    'PORTUGAL',
    'FRANÇA',
    'ALEMANHA'
};
```

DECLARATION TYPES

Quando dissemos que o PHP não é uma linguagem típificada, isto é, as variáveis não têm necessariamente que ser definidas com um tipo de valor podemos usar para parâmetros de funções, propiedades de classes e tipos de retorno de funções.

```php
<?php
// Vejamos no caso de um array como parâmetro de uma função:

function falar(array $mensagem){
    echo $mensagem;
}
//falar ('Ruan'); // não é possível.
//----------------------------------------

function conversar (string $mensagem){
    echo $mensagem;
}
//aqui vai ser feita uma conversão de inteiro para string
conversar(2500);

// conversar (['a']); //não é possível
```

DECLARATION TYPES

As declarações de tipo foram adicionadas na versão 5.1 do PHP e na versão 5.4 foram acrescentados mais tipos, como por exemplo o callable.

Outros tipos apenas foram adicionados na versão 7 e na versão 8, foi acrescido o tipo mixed.

O tipo callable tem que ser uma função, método ou projeto.

Podemos, por exemplo, usar uma função anônima.

```php
<?php
$falar = function($mensagem) {echo 'A minha mensagem é:' . $mensagem};

function minha_funcao(callable $funcao, $dados){
    $funcao($dados);
}
minha_funcao($falar, 'Esta é  minha mensagem.');

/* Tipos bool, int, float e string foram adicionados
na versão 7 do PHP.
É contudo necessário ter em conta que o PHP faz conversões
automáticas de tipos.
*/
```

DECLARATION TYPES

Também é possível atribuir tipos de retorno de dados de uma função.

```php
<?php

function funcao(): array {
    return [
        1,2,3
    ];
}

function funcao2():string {
    return [1,2,3];
    // erro
}

/* É uma forma interessante de forçar a construção
de um código com menos erros.
*/
```

STRINCT TYPING

O Comportamento do PHP é tentar converter os tipos declarados.

```php
 <?php
function falar(string $mensagem){
    echo $mensagem;
}

falar('Olá mundo!');//é possível
falar(2500);// é possível

function somar(int $v1, int $v2){
    return $v1 + $v2;
}
echo somar(10,20); // é possível
echo somar('a','b'); // é possível

// Mas podemos "obrigar" o PHP a seguir a regra estritamente definida.

declare(strict_types=1); //usar no topo do código
```

STRINCT TYPING

A partir do PHP 7.1, passou a ser possível usar um tipo de declaração nullable. Com isso, além do tipo de valor "normal"que a variável pode ter, também pode ter o valor nulo. Usamos para isso um prefixo ? antes do tipo do valor

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

function falar(?string $mensagem){
    echo $mensagem;
}
falar('Minha mensagem');
falar(null); //também passa a ser possível
```

STRINCT TYPING

No PHP8 aparece a possibilidade de definirmos mais do que um tipo para o mesmo argumento. Os tipos podem ser separados por barras verticais.

```php
 <?php
function conversar(int|string $mensagem){
    echo $mensagem;
}
conversar('Olá mundo!');
conversar(2500);

//ou no caso dos tipos de retorno

function calcular_quadrado_de(int|float $v1): int|float {
    return $v1 * $v1;
   // return 'Ruan';
}
echo calcular_quadrado_de(25);
echo calcular_quadrado_de('Ruan');//erro
```

TYPE CONVERSIONS

Embora o PHP procure automaticamente fazer uma conversão de dados entre diferentes tipos, é uma boa prática explicitar isso no código. Uma variável do tipo int não é o mesmo que uma string e vice-versa.

```php
<?php
$meu_boleano = true;
echo $meu_boleano; // resultado numa string vazia.
//echo (int)$meu_boleano;

/* Designados esta operação como conversão
explícita (explicit cast) Podemos fazer converter
os principais tipos de valores
(string), (int), (array), etc.
*/
```

TYPE CONVERSIONS

```php
<?php

/* Vejamos o caso da conversão de um array num objeto.
*/

$nomes = [
    'joao',
    'Ruan',
    'Carlos'
];
$nomes = (object)$nomes;
echo '<pre>';
print_r($nomes1);

/*Por exemplo, podemos converter uma
variável num array com único valor
*/

$nome = "Ruan";
$os_nomes = (array)$nome;
print_r($os_nomes);
```

TESTANDO VARIÁVEIS, ISSET, EMPTY, IS NULL E UNSET

Testando inúmeras vezes no nosso código a necessidade de verificar ou testar a existência de variáveis ou do seu valor.
Para isso o PHP contém um conjunto de construções internas que nos permitem fazer essa verificação.

```php
<?php

//ISSET
/* Esta construção permite verificar se
uma variável está ou não definida.
Ela retorna verdadeiro se a variável existe,
e false se não existe.
*/

$a = 1;
if(isset($a)){
    echo 'A variável existe<br>';
} else {
    echo 'A variável NAO exite<br>';
}
```

TESTANDO VARIÁVEIS, ISSET, EMPTY, IS NULL E UNSET

```php
<?php
/* Se tivermos uma variável definida, mas
o seu valor for null, é considerada como uma
variável inexistente.
*/
$b = null;
echo isset($b) ? 'SIM' : 'NÃO';

//EMPTY
/* Verificar se a variável tem um valor vazio
(null, 0, false, string vazia) e retorna verdadeiro
ou falso*/
$b = 'Ruan';
empty($b); //false

$c = false;
empty($c); //true

$nomes = [];
empty($nomes); //true
```

TESTANDO VARIÁVEIS, ISSET, EMPTY, IS NULL E UNSET

```php
<?php
//IS_NULL
/* Verifica se uma variável tem valor null ou não */

$b = 'Ruan';
is_null($b); //false

$c = null;
is_null($c); // true

/* Com PHP 8 houve uma mudança na forma
como o is_null funciona.
Anteriormente, ao testar com is_null uma
variável inexistente, era apresentado resultado
verdadeiro com um aviso.
Com PHP 8 passa a existir um erro de tipo.
*/
is_null($d); //erro
```

TESTANDO VARIÁVEIS, ISSET, EMPTY, IS NULL E UNSET

```php
<?php
//UNSET

/* Permite "Destruir" uma variável.
Exitem duas formas de o fazer, mas com
ligeiras diferenças de performace e atuação
no sistema.
*/
$a = 'Ruan';
unset($a);

$b = 'Carlos';
$b = null;

/* No primeiro caso, a variável é "liberada"
e no próximo ciclo de limpeza (garbage collector)
ela é removida da memória.
No segundo caso a variável vai persistir na memória,
mas sem valor atribuido apesar de libertar memória quando
à inexistência de valor.
Sugere-se a utilização do unset nestes casos.
*/
```

NULL COALESCING OPERADOR

Com o PHP 7 foi introduzido o operador de coalescência de Nulos. Coalecência significa aglutinação ou junção de itens separados. Basicamente este operador é um atalho para alguns casos do operador condicional ternário.

```php
<?php

Vejamos exemplos:
*/

$x = null;
$nome = $x ?? 'Sem nome';
//Este exemplo é semelhante a:

$nome1 = isset($x) ? $x : 'sem nome';

/*Portanto, se x for null, então é atribuido o
à frente dos dois sinais de interrogação.
*/
```

NULL COALESCING OPERADOR

```php
<?php
/* Com o PHP 7.4, foi acrescentado o operador de atribuição
a seguir ao operador coalescência.
Isto permite um cenário ainda mais simples para definir
variáveis nulas ou sem valor atribuído.
*/

$apelido = null;
$apelido ??= 'Apelido deconhecido';
//Neste caso apelido = 'Apelido desconhecido'

echo $nome;
echo '<br>';
echo $apelido;
```

VERIFICAR TIPOS DE VARIÁVEIS

O PHP contém um conjunto vasto de funções que permitem avaliar que tipo de dados estão guardados dentro de uma variável.

```php
<?php
/* Aqui estão alguns exemplos: */
$nome = 'Meu nome';
$idade = 23;
$acordado = true;
if(is_array($nome)){
    echo 'É um array.'; //não vai aparecer
}

if(is_bool($acordado)){
    echo 'É um valor boleano';//vai aparecer
}
/*
is_float() - Informa se a variável é do tipo float
is_int() - Informa se a variável é do tipo inteiro
is_bool() - Verifica se a variável é um boleano
is_object() - Informa se a variável é um objeto
is_array() - Verifica se a variável é um array */
```

VARIÁVEIS PRINT R, VAR DUMP E VAR EXPORT

```php
<?php
/* O PHP tem 3 funções fundamentais para
para obtermos informações a partir de
variáveis do nosso código.
    print_r
    var_dump
    var_export
*/

// PRINT_R

/* Permite ver dados de uma variável de
uma forma simples de ler.
É frequentemente usada para efeitos de debug.
*/
$a = "Ruan";
print_r($a);
echo '<br>';
$b = [1,2,3];
print_r($b);
```

VARIÁVEIS PRINT R, VAR DUMP E VAR EXPORT

```php
<?php

//VAR_DUMP
/* Tem um resultado semelhante ao print_r,
mas para além dos valores apresenta também
a informação sobre o tipo de valores.
*/

$nome = "Ruan";
var_dump($nome);
echo '<br>';

$valores = [1,2,3];
var_dump($valores);
```

VARIÁVEIS PRINT R, VAR DUMP E VAR EXPORT

```php
<?php

//VAR_EXPORT

/* Apresenta informações sobre uma variável
num estilo que pode ser usado como código PHP
*/
$nome = "Ruan";
var_export($nome);

echo '<br>';
$valores = [1,2,3];
var_export($valores);

echo '<br>';
$numeros = [];
for($i=0; $i < 10; $i++){
    $numeros[]=rand(0,100);
}
var_export($numeros);
```