



Novas Funcionalidades no PHP 5.3

Douglas V. Pasqua

Zend Certified Enginner

douglas.pasqua@gmail.com

Objetivo



- Conhecer as principais funcionalidades incluídas no PHP 5.3
- Exemplos de códigos PHP das novas funcionalidades
- Melhores práticas de programação ao se trabalhar com os novos recursos

Tópicos



- Namespaces
- Static Late Bindings
- Lambda e Closures
- Arquivos Phar
- mysqlnd – MySQL Native Driver
- Outras melhorias

Namespaces – Características



- Agrupar classes, funções e constantes em pacotes (namespaces)
- Possibilita mais de um classe/função/constante com o mesmo nome, porém em diferentes namespaces
- Diminui o número de prefixos usados nos nomes das classes/funções e constantes
- Deixa o código mais limpo e legível

Namespaces – Definição



- Determinando um namespace para um arquivo PHP
- A diretiva namespace deve ser declarada antes de qualquer outro código php ou caracteres na saída padrão

```
<?php  
namespace exemplo;
```

Namespaces – Exemplo



→ Arquivo UsuarioBlog.php

```
<?php
namespace Blog;

class Usuario {
    private $nome;

    public function setNome ($nome) {
        $this->nome = $nome;
    }

    public function getNome() {
        return "Usuário Atual do Blog é " . $this->nome;
    }
}
```

Namespaces – Exemplo



→ Arquivo UsuarioCms.php

```
<?php
namespace Cms;

class Usuario {
    private $nome;

    public function setNome ($nome) {
        $this->nome = $nome;
    }

    public function getNome() {
        return "Usuário Atual do Cms é " . $this->nome;
    }
}
```

Namespaces – Exemplo



→ Acessando as classes “Usuario” de acordo com o namespace.

```
// incluindo os arquivos das classes 'Usuarios'  
require_once("UserBlog.php");  
require_once("UserCms.php");
```

Namespaces – Exemplo



→ Acessando as classes “Usuario” de acordo com o namespace.

```
// incluindo os arquivos das classes 'Usuarios'  
require_once("UserBlog.php");  
require_once("UserCms.php");  
  
// Instanciado a classe Usuario no namespace Blog  
$usuario = new \Blog\Usuario();  
$usuario->setNome("Douglas");  
echo $usuario->getNome() . PHP_EOL;  
// Usuário Atual do Blog é Douglas
```

Namespaces – Exemplo



→ Acessando as classes “Usuario” de acordo com o namespace.

```
// incluindo os arquivos das classes 'Usuarios'
require_once("UserBlog.php");
require_once("UserCms.php");

// Instanciado a classe Usuario no namespace Blog
$usuario = new \Blog\Usuario();
$usuario->setNome("Douglas");
echo $usuario->getNome() . PHP_EOL;
// Usuário Atual do Blog é Douglas

// Instanciado a classe Usuario no namespace Cms
$usuario1 = new \Cms\Usuario();
$usuario1->setNome("Douglas");
echo $usuario1->getNome() . PHP_EOL;
// Usuário Atual do Cms é Douglas
```

Namespaces – Utilizando



→ Nome qualificado absoluto

```
$a = new \Blog\Usuario();  
$b = new \Cms\Usuario();
```

Namespaces – Utilizando



→ Nome qualificado absoluto

```
$a = new \Blog\Usuario();  
$b = new \Cms\Usuario();
```

→ Nome não-qualificado

```
namespace Blog;  
$a = new Usuario(); // \Blog\Usuario
```

Namespaces – Utilizando



→ Nome qualificado absoluto

```
$a = new \Blog\Usuario();  
$b = new \Cms\Usuario();
```

→ Nome não-qualificado

```
namespace Blog;  
$a = new Usuario(); // \Blog\Usuario
```

→ Nome qualificado relativo

```
namespace Empresa;  
$a = new Blog\Usuario(); // \Empresa\Blog\Usuario  
// Erro pois classe não existe neste namespace
```

Namespaces – Import e Alias



→ A finalidade do Alias é criar um nome mais curto para referenciar um elemento

→ A palavra-chave para import e alias é “use”

```
use \Blog\Usuario as BlogUsuario;  
use \Cms\Usuario;
```

```
$u1 = new BlogUsuario(); // alias referenciando  
\Blog\Usuario();  
$u2 = new Usuario(); // alias referenciando  
\Cms\Usuario();
```

```
$u3 = new \Blog\Usuario(); // FQN não são afetados por  
Alias e import
```

Namespaces – Espaço Global



→ A classe/função/constante pertencem ao namespace “global” quando não for especificado nenhuma namespace

```
<?php
class Usuario {
    private $nome;

    public function setNome ($usuario) {
        $this->nome = $usuario;
    }

    public function getNome() {
        return "Usuário Atual Global é " . $this->nome;
    }
}
```

Namespaces – Espaço Global



→ Quando pretendemos usar um elemento definido no espaço global, utiliza o prefixo “\” seguido pelo nome do elemento

```
$usuario = new \Usuario();  
$usuario->setNome("Douglas");  
echo $usuario->getNome(); // Usuário Atual Global é  
Douglas
```

Namespaces



→ Além de classes, namespaces se aplicam para funções e constantes

```
<?php
namespace A\B\C;
const TESTCONST = true;

// função fopen no namespace A\B\C
function fopen() {
// chamando a função Global fopen
    $f = \fopen(...);
    // constante no namespace A\B\C
    return TESTCONST;
}
```

Namespaces `__NAMESPACE__`



- A constante `__NAMESPACE__` contém o nome da namespace corrente
- No namespace “Global” a constante terá o valor “vazio”
- Além da constante `__NAMEPSACE__` existe a palavra-chave “namespace” que pode ser usada para referenciar o namespace atual

```
<?php
namespace Foo\Bar;
echo __NAMESPACE__; // exibe Foo\Bar
namespace\func(); // chama a função em Foo\Bar\func();
```

Namespaces - Múltiplos



→ É possível trabalhar com múltiplos namespaces em um mesmo arquivo

```
namespace Projeto\Blog {
const teste_constante = 1;
    class Usuario() { ... }
}
namespace Projeto\Cms {
const teste_constante = 1;
    class Usuario() { ... }
}

namespace {
const teste_constante = 1;
    class Usuario() { ... }
}
```

Namespaces - autoload



- É possível identificar o namespace na função de autoload do PHP.
- Autoload no PHP permite carregar classes dinamicamente
- A função de autoload recebe o FQN, que inclui o namespace

```
<?php
```

```
$usuario = new \Projeto\Cms\Usuario();
```

```
function __autoload($class) {  
    $class = 'classes/' . str_replace('\\', '/', $class) . '.php';  
    require_once($class);  
    // classes/Projeto/Cms/Usuario.php
```

```
}
```

Late Static Bindings



→ Pode ser usado para referência a classe chamada (run-time) em um contexto de herança de método estático

Late Static Bindings



```
<?php
class A {
    public static function who() {
        echo __CLASS__;
    }
    public static function test() {
        self::who(); // self:: referencia a classe do qual o
método test pertence
    }
}

class B extends A {
    public static function who() {
        echo __CLASS__;
    }
}

B::test(); // Será impresso "A" e não "B"
```

Late Static Bindings



```
<?php
class A {
    public static function who() {
        echo __CLASS__;
    }
    public static function test() {
        static::who(); // static:: referencia a classe
                        chamada em tempo de execução
    }
}

class B extends A {
    public static function who() {
        echo __CLASS__;
    }
}

B::test(); // Dessa vez será impresso "B"
```

Lambda e Closures



- Lambda é o mesmo que funções anônimas.
- Podem ser definidas em qualquer lugar da aplicação
- As funções anônimas podem ser atribuídas à uma variável
- A função deixará de existir quando a variável sair fora do escopo
- A utilização mais comum são em funções que aceitam um callback, como `array_map`, `usort`, etc.
- É equivalente a função `create_function`. Lambda são mais rápidas e tornam o código mais limpo.
- `create_function` é executada em run-time enquanto Lambda são criadas em compile-time

Lambda e Closures



```
<?php
$r = array_map(function($value) {
    return $value * 4;
}, array(2, 4, 6, 8, 10));
```

```
print_r ($r);
```

```
?>
```

```
8
```

```
16
```

```
24
```

```
32
```

```
40
```

Lambda e Closures



```
<?php
$a = array(10, 4, 5, 3, 1);
usort($a, function($a, $b) {
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
});
print_r($a);
?>
```

Array

```
(
    [0] => 1
    [1] => 3
    [2] => 4
    [3] => 5
    [4] => 10
)
```

Lambda e Closures



- Closures adicionam recursos à Lambdas
- Closures permite a interação de variáveis externas
- A importação das variáveis externas são feitas através da palavra-chave “use”

```
<?php
$string = "Olá teste!";
$closure = function() use ($string) { echo $string; };

$closure();
?>
```

Olá teste !

Lambda e Closures



- Por padrão as variáveis importadas são passadas por valor
- É possível passar variáveis por referência usando “&”

```
<?php
$a = 10;
$closure = function() use (&$a) { $a += 50; };

$closure();
var_dump ($a); // int(60)

$closure();
var_dump ($a); // int(110)

$closure();
var_dump ($a); // int(160)
```

Lambda e Closures



→ Além de programação procedural, closures podem ser utilizados em programação orientada a objetos

```
<?php
class DebugAll {
    public function __invoke($var) {
        var_dump($var);
    }
}
$obj1 = new DebugAll;
$obj1(50);
$obj1(100);
?>
int(50)
int(100)
```

Lambda e Closures



→ Exemplo de como podemos deixar um código mais limpo usando closures

```
<?php
```

```
$db = mysqli_connect("server", "user", "pass");
```

```
Logger::log('debug', 'database', 'Conectando com a database');
```

```
Logger::log('debug', 'database', 'Inserindo informações no  
banco de dados');
```

Lambda e Closures



→ Exemplo de como podemos deixar um código mais limpo usando closures

```
<?php
```

```
$db = mysqli_connect("server", "user", "pass");  
Logger::log('debug', 'database', 'Conectando com a database');
```

```
Logger::log('debug', 'database', 'Inserindo informações no  
banco de dados');
```

```
// definindo o closure
```

```
$logdb = function ($string) { Logger::log('debug', 'database',  
$string); };
```

```
$logdb('Conectando com a base de dados');
```

```
$logdb('Inserindo informações no banco de dados');
```

Arquivos Phar



- Possibilita distribuir a aplicação/biblioteca em um único arquivo
- Semelhante aos arquivos JAR do Java
- Funcionalidade nativa no PHP 5.3
- A criação e manipulação são feitas através de código PHP
- Praticamente não há perda de desempenho

Arquivos Phar - Criando



- É necessário setar a diretiva *phar.readonly* para *Off* no *php.ini*
- Por motivos de segurança esta diretiva é habilitada por padrão

```
$p = new Phar('/project/app.phar', 0, 'app.phar');  
$p->startBuffering();
```

Arquivos Phar - Adicionando



```
// método addFile  
$p->addFile("/project/userBlog.php");  
$p->addFile("/project/userCMS.php", "/lib/CMS.php");
```

Arquivos Phar - Adicionando



```
// método addFile
$p->addFile("/project/userBlog.php");
$p->addFile("/project/userCMS.php", "/lib/CMS.php");

// ArrayAccedss SPL
$p['index.php'] = file_get_contents("/project/index.php");
$p['teste.txt'] = "Teste 123 456";
```

Arquivos Phar - Adicionando



```
// método addFile
$p->addFile("/project/userBlog.php");
$p->addFile("/project/userCMS.php", "/lib/CMS.php");

// ArrayAccess SPL
$p['index.php'] = file_get_contents("/project/index.php");
$p['teste.txt'] = "Teste 123 456";

// Incluir vários arquivos de uma vez, recursivo
$p->buildFromDirectory('/project', '/\*.php/');
```

Arquivos Phar - Stub



→ Stub é o arquivo inicial que será carregado ao executar o pacote

```
$p->setStub('<?php Phar::mapPhar(); include  
"phar://app.phar/index.php"; __HALT_COMPILER(); ?>');
```

→ `Phar::mapPhar()` - Lê e inicializa o pacote Phar

→ O código stub deve terminar com `__HALT_COMPILER()`

Arquivos Phar - Stub



→ Utilize o método *createDefaultStub* dentro do método *setStub* para especificar o arquivo que será executado ao executar o pacote

```
$p->setStub($p->createDefaultStub('index.php'));
```

Arquivos Phar - Exemplo



```
<?php
$p = new Phar('teste.phar', 0, 'teste.phar');
$p->startBuffering();

$p['index.php'] = '<?php echo "Olá mundo"; ?>';

$p->setStub($p->createDefaultStub('index.php'));
$p->stopBuffering();
?>
```

```
# php criarTestePhar.php
```

```
# php teste.phar
Olá Mundo
```

Arquivos Phar - Manipulando



→ Acessando através de include

```
include 'teste.phar';  
include 'phar://teste.phar/index.php';
```

Arquivos Phar - Web



```
<?php
$phar = new Phar('exemplo.phar');
$phar['index.php'] = '<?php echo "Hello World Index"; ?>';
$phar['admin.php'] = '<?php echo "Hello World Admin"; ?>';
$phar->setStub('<?php Phar::webPhar(); __HALT_COMPILER(); ?>');
?>
```

- ➔ **Phar::webPhar()** atua como um frontcontroller redirecionando as chamadas web para dentro do pacote

Arquivos Phar - Web



→ <http://exemplo.org/exemplo.phar>

“Hello World Index”

→ <http://exemplo.org/exemplo.phar/admin.php>

“Hello World Admin”

Mysql Native Driver



- É uma alternativa ao extensão libmysql
- Funciona com as versões do Mysql ≥ 4.1
- Driver integrado com Zend Engine
- Performance melhorada em diversas funções
- Não necessidade de linkar com bibliotecas externas
- Utiliza “PHP License”
- Facilidade na compilação

MySQL Native Driver



- Driver pode ser utilizado nas três extensões do MySQL (mysql, mysqli, PDO_MySQL)
- Para usar o mysqlnd compilando o PHP a partir do código fonte utilize os parâmetros:
- `--with-mysql=mysqlnd / --with-mysqli=mysqlnd / --with-pdo-mysql=mysqlnd`
- Na distribuição oficial do PHP para Windows o driver mysqlnd vem habilitado por padrão
- Ainda não há suporte à SSL

Outras Melhorias



- Função *getopt*
- Error Levels e Funções Deprecated
- `__callstatic`
- Variáveis Estáticas
- Novas funções
- Goto
- NowDoc
- `__DIR__`
- Recursos SPL

Melhorias getopt



- A função *getopt* que funcionava somente no Linux esta disponível para Windows
- Agora é possível usar o caracter de atribuição “=” ao setar argumentos na linha de comando

```
# php teste_opt.php -i="valor teste123"
```

- O valor do argumento pode ser opcional

```
# php teste_opt.php -i
```

Melhorias Error Levels



- **E_ALL** agora inclui **E_STRICT**
- **E_DEPRECATED** – Usado para indicar funções que serão descontinuadas em funções futuras do php
- Exemplos de funções que serão descontinuadas: **ereg**, **ereg_replace**, **split**, **session_register**, **register_globals**, **magic_quotes_gpc**

Melhorias `__callstatic`



- `__callstatic` é um método mágico invocado quando se faz uma chamada para um método estático inexistente
- Semelhante à `__call`, porém para métodos estáticos

```
class Teste {
    static function __callStatic($nome, $args) {
        echo $nome . "(" . implode(",", $args) . ")";
    }
}
Teste::funcao('a', 'b');

// funcao(a, b)
```

Melhorias Variáveis Estáticas



```
<?php
$class_name = "Teste";
$static_method_name = "funcao";
$class_name::$static_method_name();
?>
string(32) "static function called"
```

Melhorias Novas Funções



→ array_replace e array_replace_recursive

```
<?php
$base = array("laranja", "banana", "maça", "goiabada");
$repl = array(0 => "abacaxi", 4 => "morango");

$bacia = array_replace($base, $repl);
print_r($bacia);
?>
```

```
Array (
    [0] => abacaxi
    [1] => banana
    [2] => maça
    [3] => goiabada
    [4] => morango
)
```

Melhorias Goto



→ Operador Goto pode ser usado para pular para outra seção do programa

```
<?php  
goto b;  
echo 'A';
```

```
b:  
echo 'B';  
?>
```

B

→ HereDoc

```
<?php
$variavel = 'teste';
$bar = <<<CODIGOPHP
Teste código php $variavel
CODIGOPHP;
echo $bar; // Teste código php teste
```

→ NowDoc

```
<?php
$bar = <<<'CODIGOPHP'
Teste código php $variavel
CODIGOPHP;
echo $bar;
?>
Teste código php $variavel
```

Melhorias **__DIR__**



→ Constante **__DIR__** indica o diretório corrente onde o script esta localizado

→ Antes do PHP 5.3

```
echo dirname(__FILE__);
```

→ A partir do PHP 5.3

```
echo __DIR__;
```

Melhorias SPL



→ Bibliotecas SPL inclusas no PHP 5.3: SplFixedArray, SplStack, SplQueue, SplHeap, SplMinHeap, SplMaxHeap, SplPriorityQueue

<http://www.slideshare.net/tobias382/new-spl-features-in-php-53>



Obrigado por Assistir !

Mais informações:

<http://dpasqua.wordpress.com>

E-mail: douglas.pasqua@gmail.com

Fone: 11 9236-4184