PHP

PHP: Hypertext Preprocessor

Douglas Funayama Tavares
Emilia Frigerio Cremasco

Visão Geral

História

O PHP como é conhecido hoje, é na verdade o sucessor para um produto chamado PHP/FI. Criado em 1994 por Rasmus Lerdof, a primeira encarnação do PHP foi um simples conjunto de binários Common Gateway Interface (CGI) escrito em linguagem de programação C. Originalmente usado para acompanhamento de visitas para seu currículo online, ele nomeou o conjunto de scripts de "Personal Home Page Tools" mais frequentemente referenciado como "PHP Tools." Ao longo do tempo, mais funcionalidades foram desejadas, e Rasmus reescreveu o PHP Tools, produzindo uma maior e rica implementação. Este novo modelo foi capaz de interações com Banco de Dados e mais, fornecendo uma estrutura no qual os usuários poderiam desenvolver simples e dinâmicas aplicações web, como um livros de visitas. Em Junho de 1995, Rasmus liberou o código fonte do PHP Tools para o público, o que permitiu que desenvolvedores usassem da forma como desejassem. Isso permitiu - e encorajou - usuários a fornecerem correções para bugs no código, e em geral, aperfeiçoá-lo.

História

Mais tarde no mesmo ano, Rasmus expandiu o PHP incluindo algumas funcionalidades básicas como bem conhecemos hoje. PHP continuou a desfrutar um crescimento e aceitação como uma ferramenta CGI --- mas ainda não como uma linguagem.

Logo depois, Rasmus liberou outra versão que foi o primeiro lançamento a vangloriar-se que era, na época, considerado um avançado script de interface. A linguagem foi desenvolvida para, deliberadamente, ser parecida com C, tornando-a fácil para ser adotada por desenvolvedores habituados com C, Perl e linguagens similares. Em junho de 1998, com muitos novos desenvolvedores ao redor do mundo unindo esforços, PHP 3.0 foi anunciado pelo novo time de desenvolvimento do PHP. Essa foi a primeira versão que se assemelha com o PHP como conhecemos hoje.

Hoje

- PHP 7
- Desenvolvimento Web, mesclada no código HTML
- Desenvolvimento de aplicações do lado do servidor
- Suporte a diversos bancos de dados

Características

- Orientado a objetos
- Imperativo
- Portabilidade
- Tipagem dinamica
- Fracamente tipada
- Interpretada
- Case sensitive
- Fácil aprendizagem
- Suporte para banco de dados
- Software livre

Considerações sobre instalação

Formas de usar o PHP

- Website e aplicações web
- Scripts na linha de comando
- Apps de Desktop

Website e aplicações web

- PHP
- servidor web
- navegador

Scripts em linha de comando

- executável de linha de comando

Apps para desktop (GUI)

- Extensao PHP-GTK

HOW TO PHP

Sintaxe Basica

```
<html>
 <head>
  <title>PHP Teste</title>
 </head>
 <body>
 <?php echo "<p>Olá Mundo</p>"; ?>

 </body>
</html>
```

```
<p>Isto vai ser ignorado pelo PHP.</p>
<?php echo 'Enquanto isto vai ser

interpretado.'; ?>

<p>Isto também vai ser ignorado pelo PHP.</p>
```

```
<?php

echo "Hello world";

// ... mais código

echo "última instrução";

/* o script termina aqui, sem tag de
fechamento PHP */
```

```
<?php

    echo 'Isto é um teste';

?>

<?php echo 'Isto é um teste' ?>

<?php echo 'A última tag de fechamento foi
omitida.';
```

Comentários

```
<?php
    echo 'Isto é um teste'; // Estilo de comentário de uma linha
    /* Este é um comentário de múltiplas linhas
       ainda outra linha de comentário */
    echo 'Isto é ainda outro teste';
    echo 'Um teste final'; # Este é um comentário de uma linha
?>
```

Tipos

- Escalares
  - boolean
  - integer
  - float (double)
  - string
- Compostos
  - array
  - object
- Especiais
  - resource
  - NULL

Tipos

➤ PHP é uma linguagem de tipagem dinâmica, fracamente tipada, que usa *copy-on-write* e contagem de referências (coletor de lixo). Todas as variáveis são representadas pelas estrutura _zval_struct.

```
typedef struct _zval_struct {
    zvalue_value value;       /* variable value */
    zend_uint refcount__gc;   /* reference counter */
    zend_uchar type;          /* value type */
    zend_uchar is_ref__gc;    /* reference flag */
} zval;
```

```
typedef union _zvalue_value {
    long lval;                /* long value */
    double dval;              /* double value */
    struct {
        char *val;
        int len;              /* this will always be set for strings */
    } str;                    /* string (always has length) */
    HashTable *ht;            /* an array */
    zend_object_value obj;    /* stores an object store handle, and
handlers */
} zvalue_value;
```

Tipos

➢ O PHP não obriga (ou suporta) a definição de tipo explícita na declaração de variáveis: o tipo de uma variável é determinado pelo contexto em que a variável é utilizada. Isto significa que, atribuir um valor string a variável $var, fará $var se tornar uma string. Se um valor integer for atribuído a $var, ela se torna um integer.

➢ Conversao de tipos

- (int), (integer) - converte para inteiro
- (bool), (boolean) - converte para booleano
- (float), (double), (real) - converte para ponto flutuante
- (string) - converte para string
- (array) - converte para array
- (object) - converte para objeto
- (unset) - converte para NULL (PHP 5)

Variáveis

➤ As variáveis no PHP são representadas por um cifrão ($) seguido pelo nome da variável. Os nomes de variáveis são case-sensitive.

➤ Um nome de variável válido inicia-se com uma letra ou sublinhado, seguido de qualquer número de letras, números ou sublinhados. Em uma expressão regular, poderia ser representado assim: *'[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]**

➤ *$this* é uma variável especial que não pode ser atribuída.

➤ Não é necessário inicializar variáveis no PHP. Variáveis não inicializadas tem um valor padrão de tipo. O padrão de booleanos é FALSE, de inteiros e ponto-flutuantes é zero, strings (ex.: se utilizados em echo), são definidas como vazia e arrays tornam-se um array vazio.

Tipos

➤ float

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

➤ Números de ponto flutuante tem precisão limitada. Embora dependa do sistema, o PHP geralmente utiliza o formato de precisão dupla do IEEE 754, que trará uma precisão máxima devida a arredondamentos da ordem de 1.11e-16.

Tipos

- boolean

```php
<?php
$foo = True;
?>
```

Tipos

➤ integer

```
<?php
$a = 1234; // numero decimal
$a = -123; // numero negativo
$a = 0123; // octal (equivalente a 83
em decimal)
$a = 0x1A; // hexadecimal (equivalente
a 26 decimal) //
$a = 0b11111111; // numero binario
(equivalent to 255 decimal)
?>
```

➤ Inteiros podem ser especificados em notação decimal (base 10), hexadecimal (base 16), octal (base 8) ou binária (base 2), opcionalmente precedido de sinal (- ou +).

➤ O tamanho de um inteiro depende da plataforma, sendo um número aproximado a 2 bilhões o valor mais comum (número de 32 bits com sinal). Plataformas 64-bit possuem comumente o valor máximo de aproximadamente 9E18, exceto no Windows em versões anteriores ao PHP 7, onde são sempre 32-bit. O PHP não suporta inteiros sem sinal. O tamanho do inteiro pode ser determinado pela constante PHP_INT_SIZE.

Tipos

➤ strings com aspas simples

➤ O tipo string possui o tamanho máximo de 2GB (máximo de 2147483647 bytes)

```
<?php
echo 'isto é uma string comum';
echo 'Você pode incluir novas linhas em strings,
dessa maneira';

echo 'Isto não será substituido: \n uma nova linha';

echo 'Você tem certeza em apagar C:\*.*?';
?>
```

Tipos

- strings com aspas duplas

Sequencias de escape	
\n	Fim de linha
\r	Retorno de carro
\t	TAB orizontal
\v	TAB vertical
\e	escape
\f	Form feed
\\	contrabarra

Tipos

➤ array

Um array em PHP é na verdade um mapa ordenado. Um mapa é um tipo que relaciona *valores* a *chaves*. Este tipo é otimizado para várias usos diferentes: ele pode ser tratado como um array, uma lista (vetor), hashtable (que é uma implementação de mapa), dicionário, coleção, pilha, fila e outros. Assim como existe a possibilidade dos valores do array serem outros arrays, árvores e arrays multidimensionais.

Tipos

➤ array

A chave pode ser um inteiro ou uma string.

O valor pode ser de qualquer tipo.

```php
<?php

array(
    chave  => valor,
    chave2 => valor2,
    chave3 => valor3,
    ...
)

?>
```

```php
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
    100   => -100,
    -100  => 100,
);
var_dump($array);
?>
```

```
array(4) {
  ["foo"]=>
  string(3) "bar"
  ["bar"]=>
  string(3) "foo"
  [100]=>
  int(-100)
  [-100]=>
  int(100)
}
```

Tipos

➤ array

A chave é opcional. Se não for especificada, o PHP utilizará o incremento do tipo inteiro.

```php
<?php
$array = array("foo", "bar", "hello", "world");
var_dump($array);
?>
```

```
array(4) {
  [0]=>
  string(3) "foo"
  [1]=>
  string(3) "bar"
  [2]=>
  string(5) "hello"
  [3]=>
  string(5) "world"
}
```

Tipos

➤ array

Acessando elementos:

```php
<?php
$array = array(
    "foo" => "bar",
    42    => 24,
    "multi" => array(
         "dimensional" => array(
             "array" => "foo"
         )
    )
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);
?>
```

```
string(3) "bar"
int(24)
string(3) "foo"
```

Tipos

➤ array

É possível especificar chaves apenas para alguns elementos.

```php
<?php
$array = array(
         "a",
         "b",
    6 => "c",
         “d”,
);
var_dump($array);
?>
```

```
array(4)
{
  [0]=>
  string(1) "a"
  [1]=>
  string(1) "b"
  [6]=>
  string(1) "c"
  [7]=>
  string(1) "d"
}
```

Tipos

➤ object

```php
<?php
class classe
{
    function funcao()
    {
        echo "Hello World!";
    }
}

$bar = new classe;
$bar->funcao();
?>
```

Tipos

```php
<?php
$foo = 10;              // $foo é um inteiro
$str = "$foo";          // $str é uma string
$fst = (string) $foo; // $fst tambem é uma string

// Isto imprimiria "eles são o mesmo"
if ($fst === $str) {
    echo "eles são o mesmo";
}
?>
```

Tipos

➤ Resource

É uma variável especial, que mantém uma referência a um recurso externo. Recursos são criados e usados por funções especiais.

Ex.: conexões com banco de dados, manipular arquivos, streams, etc.

➤ NULL

Representa uma variável sem valor.

Foi atribuída a constante NULL, ainda não recebeu nenhum valor, foi apagada com unset().

$var = NULL;

Variáveis

➤ Por padrão, as variáveis são sempre atribuídas por valor. Porém, PHP também oferece atribuição por referência.

➤ Para atribuir por referência, simplesmente adicione um e-comercial (&) na frente do nome da variável que estiver sendo atribuída

```
<?php
$foo = 'Bob';              // Atribui o valor 'Bob' a variável $foo
$bar = &$foo;              // Referecia $foo através de $bar.
$bar = "My name is $bar";  // Altera $bar...
echo $bar;
echo $foo;                 // $foo é alterada também.
?>
```

Variáveis

➤ A maioria das variáveis tem somente escopo local. As variáveis globais precisam ser declaradas como globais dentro de uma função, se forem utilizadas em uma.

```php
<?php
$a = 1;
$b = 2;

function Soma()
{
    global $a, $b;

    $b = $a + $b;
}

Soma();
echo $b;
?>
```

```php
<?php
$a = 1;
$b = 2;

function Soma()
{
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}

Soma();
echo $b;
?>
```

```php
<?php
function Teste()
{
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

Variáveis de fontes externas

➤ Quando um formulário é submetido para um script PHP, a informação deste formulário estará automaticamente disponível ao script.

```
<form action="foo.php" method="post">
    Nome:  <input type="text" name="username" /><br />
    Email: <input type="text" name="email" /><br />
    <input type="submit" name="submit" value="Me aperte!" />
</form>
```

```
<?php
echo $_POST['username'];
?>
```

Constantes

➤ Não possuem o sinal de $ antes delas.

➤ Podem ser definidas e acessadas de qualquer lugar sem que as regras de escopo variável sejam aplicadas.

Constantes

```php
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // imprime "Hello world."
?>
```

Constantes

```php
<?php
// Funciona a partir do PHP 5.3.0
const CONSTANT = 'Hello World';

echo CONSTANT;

// Funciona a partir do PHP 5.6.0
const ANOTHER_CONST = CONSTANT.'; Goodbye World';
echo ANOTHER_CONST;

const ANIMALS = array('dog', 'cat', 'bird');
echo ANIMALS[1]; // imprime "cat"

// A partir do PHP 7
define('ANIMALS', array(
    'dog',
    'cat',
    'bird'
));
echo ANIMALS[1]; // imprime "cat"
?>
```

Constantes

➤ Constantes mágicas

Nome	Descrição
__LINE__	O número da linha corrente do arquivo
__FILE__	O caminho completo e nome do arquivo com links simbólicos resolvidos.
__DIR__	O diretório do arquivo.
__FUNCTION__	O nome da função.
__CLASS__	O nome da classe.
__NAMESPACE__	O nome do método da classe.
__METHOD	O nome do namespace corrente.

Operadores

Operadores aritméticos		
Exemplo	Nome	Resultado
-$a	Negacao	Oposto de $a
$a + $b	Adicao	Soma de $a e $b
$a - $b	Subtracao	Diferenca entre $a e $b
$a * $b	Multiplicacao	Produto de $a e $b
$a / $b	Divisao	Quociente de $a e $b
$a % $b	Modulo	Resto de $a dividido por $b
$a ** $b	Exponencial	$a elevado a $b

Operadores

➤ Operadores aritméticos

```php
<?php

echo (5 % 3)."\n";          // imprime 2
echo (5 % -3)."\n";         // imprime 2
echo (-5 % 3)."\n";         // imprime -2
echo (-5 % -3)."\n";        // imprime -2

?>
```

Precedência dos operadores

Associação	Operadores	Informação Adicional
não associativo	*clone new*	clone e new
esquerda	*[*	array()
direita	**	aritmética
direita	++ -- ~ (int) (float) (string) (array) (object) (bool) @	types e incremento/decremento
não associativo	*instanceof*	tipos
direita	*!*	lógicos
esquerda	* / %	aritmética
esquerda	+ - .	aritmética e string
esquerda	<< >>	bits
não associativo	< <= > >=	comparação
não associativo	== != === !== <> <=>	comparação
esquerda	&	bits e referências
esquerda	^	bits
esquerda	|	bits
esquerda	&&	lógicos
esquerda	||	lógicos
direita	??	comparação
esquerda	? :	ternário
direita	= += -= *= **= /= .= %= &= |= ^= <<= >>=	atribuição
esquerda	*and*	lógicos
esquerda	*xor*	lógicos
esquerda	*or*	lógicos

Operadores

- Operadores de atribuição

```php
<?php

$a = ($b = 4) + 5;

?>
```

```php
<?php

$a = 3;
$a += 5;
$b = "Bom ";
$b .= "Dia!";

?>
```

```php
<?php
$a = 3;
$b = &$a; // $b é uma referência de $a

print "$a\n"; // imprime 3
print "$b\n"; // imprime 3

$a = 4; // modificamos $a

print "$a\n"; // imprime 4
print "$b\n"; // imprime 4 também

?>
```

Operadores

Operadores bit-a-bit	
Exemplo	Nome
$a & $b	AND
$a | %b	OR
$a ^ $b	XOR
~$a	NOT
$a << $b	Deslocamento a esquerda
$a >> $b	Deslocamento a direita

Operadores

Operadores de comparação	
$a == $b	Igual
$a === $b	Identico
$a != $b	Diferente
$a <> $b	Diferente
$a !== $b	Nao identico
$a > $b	Se $a é maior que $b
$a < $b	Se $a é menor que $b
$a ⇔ $b	Compara $a e $b

Operadores

➤ Operadores de execução

O PHP suporta um operador de execução: acentos graves (``). Note que não são aspas simples! O PHP tentará executar o conteúdo dentro dos acentos graves como um comando do shell; a saída será retornada (isto é, ela não será simplesmente mostrada na tela; ela pode ser atribuída a uma variável).

```
<?php
$output = `ls -al`;
echo "<pre>$output</pre>";
?>
```

Operadores

Operadores logicos	
$a and $b	AND
$a or $b	OR
$a xor $b	XOR
!$a	NOT
$a && $b	AND
$a || $b	OR

Estruturas de controle

- if
- else
- elseif/ else if
- while
- do-while
- for
- foreach
- break
- continue
- switch
- declare
- return
- require
- include
- require_once/include_once
- goto

Estruturas de controle

➤ if

```php
<?php

if ($a > $b) {
  echo "a is bigger than b";
  $b = $a;
}

?>
```

```php
<?php

if ($a > $b) {
  echo "a is greater than b";
} else {
  echo "a is NOT greater than b";
}

?>
```

```php
<?php
if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a is smaller than b";
}
?>
```

Estruturas de controle

```php
<?php
/* example 1 */

$i = 1;
while ($i <= 10)
{
    echo $i++;
}

?>
```

```php
<?php


for ($i = 1; $i <= 10; $i++)
{
    echo $i;
}

?>
```

```php
<?php

$i = 0;

do
{
    echo $i;
} while ($i > 0);

?>
```

```php
<?php
$arr = array(1, 2, 3, 4);
foreach ($arr as &$value) {
    $value = $value * 2;
}
// $arr is now array(2, 4, 6, 8)
unset($value); /* break the
reference with the last element */
?>
```

Estruturas de controle

```php
<?php
$arr = array('one', 'two', 'three',
'four', 'stop', 'five');
while (list(, $val) = each($arr))
{
    if ($val == 'stop')
    {
        break;
    }
    echo "$val<br />\n";
}
?>
```

```php
<?php
while (list($key, $value) = each($arr))
{
    if (!($key % 2)) {
        continue;
    }
    do_something_odd($value);
} ?>
```

```php
<?php
if ($i == 0) {
    echo "i equals 0";
} elseif ($i == 1) {
    echo "i equals 1";
} elseif ($i == 2) {
    echo "i equals 2";
}

switch ($i) {
    case 0:
        echo "i equals 0";
        break;
    case 1:
        echo "i equals 1";
        break;
    case 2:
        echo "i equals 2";
        break;
}
?>
```

Estruturas de controle

```
<?php
declare(encoding='ISO-8859-1');
// code here
?>
```

Estruturas de controle

```
vars.php
<?php

$color = 'green';
$fruit = 'apple';

?>

test.php
<?php

echo "A $color $fruit"; // A

include 'vars.php';

echo "A $color $fruit"; // A green apple

?>
```

➤ a declaração *require* é idêntica a *include* exceto que em caso de falha, produzirá um erro fatal de nivel E_COMPILE_ERROR.

➤ a declaracao *require_once* é identica a *require* exceto que será verificado se o arquivo já foi incluído.

➤ a declaracao *include_once* é identica a *include* exceto que será verificado se o arquivo já foi incluído

Estruturas de controle

➤ O PHP oferece uma sintaxe alternativa para algumas estruturas de controle; *if*, *while*, *for*, *foreach*, e *switch*. Em cada caso, basicamente a sintaxe alternativa é trocar a chave de abertura por dois pontos (:) e a chave de fechamento por *endif;*, *endwhile;*, *endfor;*, *endforeach;*, ou *endswitch;*, respectivamente.

```
<?php
if ($a == 5):
    echo "a equals 5";
    echo "...";
elseif ($a == 6):
    echo "a equals 6";
    echo "!!!";
else:
    echo "a is neither 5 nor 6";
endif;
?>
```

Funções

➤ Todas as funções têm escopo global.

➤ As funções não precisam ser criadas antes de serem referenciadas, exceto quando uma função é condicionalmente definida.

➤ PHP não suporta sobrecarga de funções.

➤ Não é possível cancelar ou alterar funções.

➤ Número variável de argumentos e argumentos padrões são suportados por funções.

➤ Por padrão a passagem de parâmetros é por valor, mas é possível passar parâmetros por referência

Funções

➤ Pseudo-código

```
<?php
function foo ($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Exemplo de função.\n";
    return $valor_retornado;
}
?>
```

➤ Funções dentro de funções

```
<?php
function foo()
{
  function bar()
  {
    echo "Eu não existo até foo() ser
chamada.\n";
  }
}

foo();

bar();

?>
```

Funções

➤ Funções definidas condicionalmente

```php
<?php

$makefoo = true;

bar();

if ($makefoo) {
  function foo()
  {
    echo "Eu não existo até que o programa passe por aqui.\n";
  }
}


if ($makefoo) foo();

function bar()
{
  echo "Eu existo imediatamente desde o programa começar.\n";
}
?>
```

Funções

➤ Argumentos de funções

```
<?php
function takes_array($input)
{
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
?>
```

```
<?php
function add_some_extra(&$string)
{
    $string .= ' e alguma coisa mais.';
}
$str = 'Isto é uma string,';
add_some_extra($str);
echo $str;    // imprime 'Isto é uma string, e alguma
coisa mais.'
?>
```

```
<?php
function makecoffee($type =
"cappuccino")
{
    return "Fazendo uma xícara de
café $type.\n";
}
echo makecoffee();
echo makecoffee(null);
echo makecoffee("espresso");
?>
```

Funções

➤ Funções variáveis

```
function foo() {
    echo "Chamou foo()<br>\n";
}

function bar($arg = '')
{
    echo "Chamou bar(); com argumento '$arg'.<br />\n";
}

// Essa eh uma funcao wrapper para echo()
function echoit($string)
{
    echo $string;
}

$func = 'foo';
$func();        // Chama foo()

$func = 'bar';
$func('test');  // Chama bar()

$func = 'echoit';
$func('test');  // Chama echoit()
?>
```

Funções

➤ Funções variáveis

```php
<?php
class Foo
{
    function MetodoVariavel()
    {
        $name = 'Bar';
        $this->$name(); // Isto chama o método Bar()
    }

    function Bar()
    {
        echo "Bar foi chamada!";
    }
}

$foo = new Foo();
$funcname = "MetodoVariavel";
$foo->$funcname();  // Isto chama $foo->MetodoVariavel()

?>
```

Funções

➤ Número variável de argumentos

```php
<?php
function sum(...$numbers) {
    $acc = 0;
    foreach ($numbers as $n) {
        $acc += $n;
    }
    return $acc;
}

echo sum(1, 2, 3, 4);
?>
```

Funções

➢ Declarações de tipo

É possível que funções demandem que parâmetros sejam de certos tipos ao chamá-los.

Tipos válidos: instance of, self, array, boolean, integer, float, string.

➢ Declaração de tipos de retorno

O PHP 7 acrescenta suporte a declaração de tipo de retorno

➢ Tipagem estrita

É possível habilitar o modo estrito arquivo a arquivo. No modo estrito somente uma variável do exato tipo especificado na declaração será aceito, ou uma exceção TypeError será lançada.

Funções

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

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

var_dump(sum(1, 2));
var_dump(sum(1.5, 2.5));
?>
```

```php
<?php
function sum($a, $b): float {
    return $a + $b;
}

// Note que um float será retornado.
var_dump(sum(1, 2));
?>
```

```
int(3)

Fatal error: Uncaught TypeError: Return value of sum() must be of the type integer, float
returned in - on line 5 in -:5
Stack trace:
#0 -(9): sum(1, 2.5)
#1 {main}
  thrown in - on line 5
```

Classes

➢ A partir do PHP 5, o modelo de objetos foi reescrito para permitir melhor performance e mais funcionalidades. Esta é uma grande modificação do PHP 4. PHP 5 tem um modelo de objetos completo.

➢ O PHP trata objetos da mesma maneira que referências ou manipuladores, significando que cada variável contém uma referência a um objeto ao invés de uma cópia de todo o objeto.

Classes

➤ Definição de uma classe

```php
<?php
class SimpleClass
{
    // declaração de propriedade
    public $var = 'um valor padrão';

    // declaração de método
    public function displayVar() {
        echo $this->var;
    }
}
?>
```

➤ Criando uma instância

```php
<?php

$instance = new SimpleClass();

// Também pode ser feito com uma variável:
$className = 'SimpleClass';
$instance = new $className(); // new SimpleClass()

?>
```

Classes

➤ Atribuição de objetos

```php
<?php

$instance = new SimpleClass();

$assigned   =  $instance;
$reference  =& $instance;

$instance->var = '$assigned will have this value';

$instance = null;
?>
```

➤ Pseudo-variável $this

```php
class A
{
    function foo()
    {
        if (isset($this)) {
            echo '$this está definida
(';
            echo get_class($this);
            echo ")\n";
        } else {
            echo "\$this não está
definida.\n";
        }
    }
}
```

Classes

➤ Construtores e destrutores

```php
<?php
class BaseClass {
   function __construct() {
       print "In BaseClass constructor\n";
   }
}

class SubClass extends BaseClass {
   function __construct() {
       parent::__construct();
       print "In SubClass constructor\n";
   }
}

// In BaseClass constructor
$obj = new BaseClass();

// In BaseClass constructor
// In SubClass constructor
$obj = new SubClass(); ?>
```

```php
<?php
class MyDestructableClass {
   function __construct() {
       print "In constructor\n";
       $this->name = "MyDestructableClass";
   }

   function __destruct() {
       print "Destroying " . $this->name .
"\n";
   }
}

$obj = new MyDestructableClass();
?>
```

Classes

➤ Visibilidade de atributos

```php
<?php
/**
 * Define MinhaClasse
 */
class MinhaClasse
{
    public $publica = 'Public';
    protected $protegida = 'Protected';
    private $privada = 'Private';

    function imprimeAlo()
    {
        echo $this->publica;
        echo $this->protegida;
        echo $this->privada;
    }
}
```

```php
$obj = new MinhaClasse();
echo $obj->publica; // Funciona
echo $obj->protegida; // Erro Fatal
echo $obj->privada; // Erro Fatal
$obj->imprimeAlo(); // Mostra Public, Protected
e Private
```

Classes

➤ Visibilidade de métodos

```
class MinhaClasse
{
    public function __construct() { }

    public function MeuPublico() { }

    protected function MeuProtegido() { }

    private function MeuPrivado() { }

    // Esse é public
    function Foo()
    {
        $this->MeuPublico();
        $this->MeuProtegido();
        $this->MeuPrivado();
    }
}
```

Classes

➤ Herança

```php
<?php

class Foo {
    public function printItem($string)
    {
        echo 'Foo: ' . $string . PHP_EOL;
    }

    public function printPHP()
    {
        echo 'PHP is great.' . PHP_EOL;
    }
}

class Bar extends Foo{
    public function printItem($string)
    {
        echo 'Bar: ' . $string . PHP_EOL;
    }
}
```

```php
$foo = new Foo();
$bar = new Bar();
$foo->printItem('baz'); // Output: 'Foo:
baz'
$foo->printPHP();       // Output: 'PHP
is great'
$bar->printItem('baz'); // Output: 'Bar:
baz'
$bar->printPHP();       // Output: 'PHP
is great'

?>
```

Classes

➤ Abstração de classes

```
abstract class AbstractClass
{
    // Force Extending class to define this
method
    abstract protected function getValue();
    abstract protected function
prefixValue($prefix);

    // Common method
    public function printOut() {
        print $this->getValue() . "\n";
    }
}
```

```
class ConcreteClass1 extends AbstractClass
{
    protected function getValue() {
        return "ConcreteClass1";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}
```

Classes

- Interfaces de objetos

```
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}
```

```
class Template implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name .
'}', $value, $template);
        }

        return $template;
    }
}
```

Exceções

```php
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('Divisão por zero.');
    }
    return 1/$x;
}

try {
    echo inverse(5) . "\n";
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Exceção capturada: ',  $e->getMessage(), "\n";
}

// Execução continua
echo "Olá mundo\n";
?>
```

Programação concorrente

- Existem várias extensões em PHP que podem ser usadas ao criar programas concorrentes
- pthreads
- Semaphore

Programação concorrente

- pthreds
  - Threaded
  - Thread
  - Worker
  - Collectable

Programação concorrente

➤ Thread

```php
<?php
class AguardaRand extends Thread {

    protected $id;

    public function __construct($id) {
        $this->id = $id;
    }
    public function run() {
        $tempo_rand = mt_rand(1, 4);
        sleep($tempo_rand);
        printf(
            "Sou a thread %d e aguardei %d segundos\n",
            $this->id,
            $tempo_rand
        );
    }
}
```

```php
$vetor = array();
for ($id = 0; $id < 10; $id++) {
    $vetor[] = new AguardaRand($id);
}

foreach ($vetor as $thread) {
    $thread->start();
}

// Encerrar o script
exit(0);
```

Programação concorrente

```
Sou a thread 5 e aguardei 1 segundos
Sou a thread 8 e aguardei 1 segundos
Sou a thread 2 e aguardei 2 segundos
Sou a thread 0 e aguardei 3 segundos
Sou a thread 9 e aguardei 3 segundos
Sou a thread 1 e aguardei 4 segundos
Sou a thread 3 e aguardei 4 segundos
Sou a thread 4 e aguardei 4 segundos
Sou a thread 6 e aguardei 4 segundos
Sou a thread 7 e aguardei 4 segundos
```

Programação concorrente

➤ Semaphore

```
<?php
$key = 123321;
$maxAcquire = 1;
$permissions =0666;
$autoRelease = 1;

$semaphore = sem_get($key, $maxAcquire, $permissions,
$autoRelease);

if(!$semaphore) {
echo "Failed on sem_get().\n";
exit;
}
```

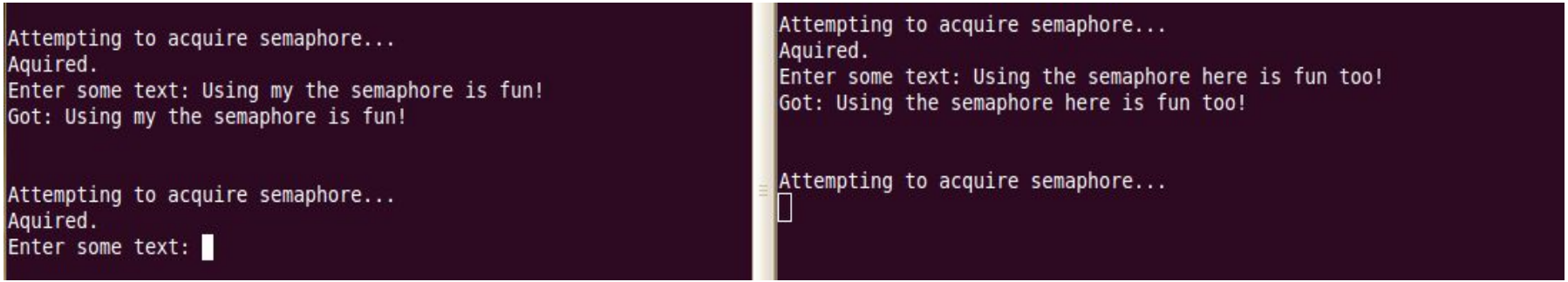
```
for($i = 0; $i < 2; $i++) {
echo "\nAttempting to acquire
semaphore...\n";
                              sem_acquire($
semaphore);

echo "Aquired.\n";
echo "Enter some text: ";
$handler = fopen("php://stdin", "r");
$text = fgets($handler);

fclose($handler);
                              sem_release($
semaphore);

echo "Got: $text \n";
}
?>
```

Programação concorrente



Banco de Dados

```php
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>
```

Comparação entre linguagens

Critérios gerais

Critérios gerais	PHP	C	Java
Aplicabilidade	Parcial	Sim	Parcial
Confiabilidade	Parcial	Não	Sim
Aprendizado	Sim	Sim	Parcial
Eficiência	Sim	Sim	Parcial
Portabilidade	Parcial	Não	Sim

Critérios gerais

Critérios gerais	PHP	C	Java
Paradigma	OO e Estruturado	Estruturado	OO
Evolutibilidade	Parcial	Não	Sim
Reusabilidade	Sim	Parcial	Sim
Interação	Sim	Sim	Parcial
Custo	Depende	Depende	Depende

Critérios específicos

Característica	PHP	C	Java
Escopo	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador/sistema	Programador	Sistema
Passagem de parametros	Lista variável, por valor e cópia de referência	Lista variável e por valor	Lista variável, por valor e cópia de referência

Critérios específicos

Critérios específicos	PHP	C	Java
Encapsulamento e proteção	Sim	Parcial	Sim
Sistemas de tipos	Parcial	Não	Sim
Verificação de tipos	Dinâmica	Estática	Estática/Dinamica
Polimorfismo	Todos	Coerção e sobrecarga	Todos
Exceções	Sim	Não	Sim
Concorrência	Biblioteca de funções	Biblioteca de funções	Sim

Onde aprender PHP?

Referências

- Manual do PHP: http://php.net/manual/pt_BR/index.php
- W3Schools Tutorial: http://www.w3schools.com/php/