Programação para Internet

Módulo 5

Introdução à Web Dinâmica com PHP

Prof. Dr. Daniel A. Furtado - FACOM/UFU

Websites Estáticos

- Conteúdo sempre o mesmo
- HTML, CSS, JavaScript, imagens, etc.
- Sem acesso a banco de dados

Websites Dinâmicos

- Conteúdo produzido dinamicamente
- Linguagem *server-side*: PHP, Python, Java, etc.
- Normalmente acessam bancos de dados
- Realizam processamentos adicionais no servidor

O que é PHP?

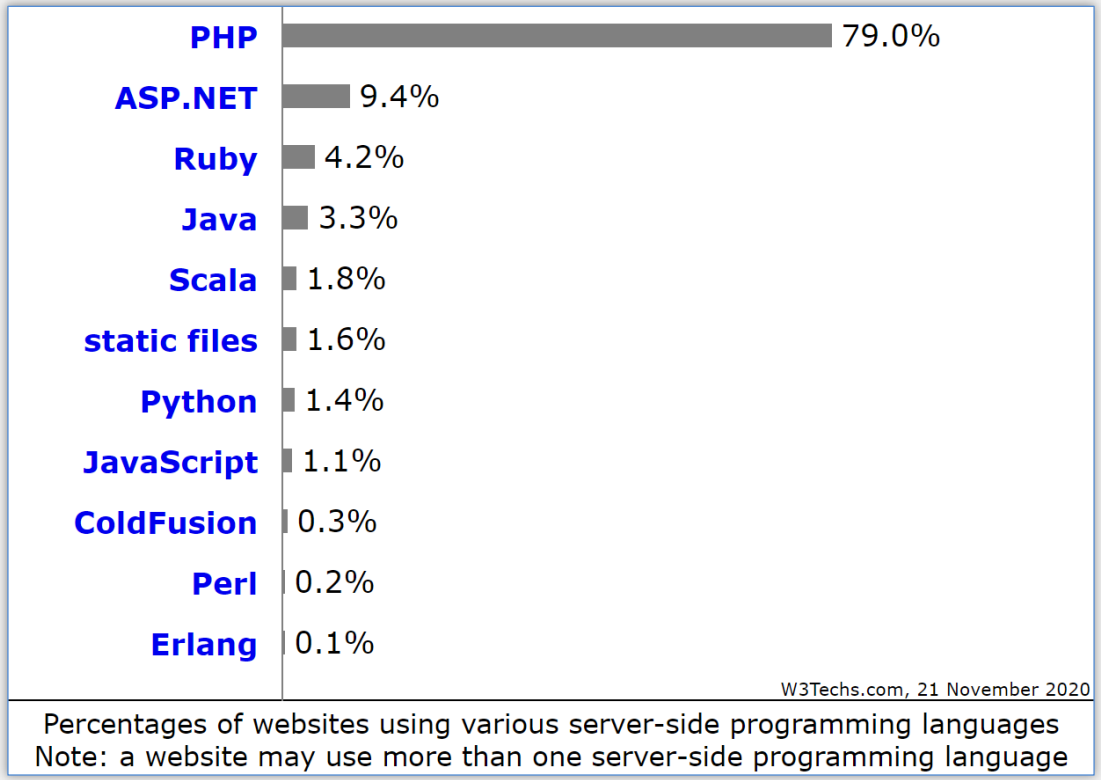
- PHP é uma linguagem de script *server-side*
- Desenvolvimento de websites dinâmicos
- Acrônimo recursivo para **P**HP: **H**ypertext **P**reprocessor

Porque PHP?

- Gratuito e *Open Source*
- Pode ser executado em várias plataformas
  - Windows, Linux, Unix, Mac OS
- Compatível com vários servidores web
  - Apache HTTP, NGINX, Microsoft IIS, etc.
- Suporta vários sistemas de banco de dados
- Alto desempenho
- Variedade de frameworks: Laravel, Symfony, etc.
- Serviços gratuitos de hospedagem com suporte a PHP
- Fácil aprendizado
- Popularidade

Linguagens Server-Side

PHP é utilizado em 79% dos websites que se tem conhecimento da linguagem *server-side*



Fonte: W3Techs (novembro/2020)

Exemplos de Websites Utilizando PHP

- Facebook
- Wikipedia
- Yahoo
- Flickr
- WordPress

O que pode ser feito com PHP?

- Gerar conteúdo de página dinamicamente
- Manipular arquivos no servidor
- Receber, validar e manipular dados de formulários
- Enviar e receber cookies
- Adicionar, atualizar ou remover dados em banco de dados
- Controle de acesso e manipulação de sessões

Algumas opções para começar com PHP

- Serviço de hospedagem com suporte a PHP
- Instalação manual em servidor
  - Instalação de servidor web (ex. Apache HTTP)
  - Instalação do PHP (php.net/downloads)
- Pacotes de instalação
  - Ex.: **WAMP** (**W**indows, **A**pache, **M**ySQL and **P**HP)

Exemplos de IDE para Desenv. com PHP

- Eclipse IDE for PHP Developers
  - Gratuita
  - www.eclipse.org/downloads/packages
- PhpStorm
  - Teste por 30 dias
  - Gratuita para estudantes

Hello World com PHP

Arquivo PHP

- Extensão .php
- Arquivo pode conter código PHP, HTML, CSS, etc.
- PHP é um pré-processador de hipertexto
- Trechos de código PHP são pré-processados no servidor
- Saída gerada pelo código PHP é mesclada com restante
- Resultado final é enviado para o usuário (navegador)
- Código PHP deve ser inserido dentro de `<?php ?>`

Hello World

```php
<?php

   for ($i = 0; $i < 10; $i++) {
      echo "Hello World!";
   }

?>
```

Arquivo exemplo1.php
contendo apenas código PHP

Hello World

```
<html>
<body>
  <h1>Página dinâmica com PHP</h1>

  <?php
    for ($i = 0; $i < 5; $i++) {
      echo "<p> Hello World! </p>";
    }
  ?>

</body>
</html>
```

Arquivo exemplo2.php
contendo código PHP e HTML

```
<html>
<body>
 <h1>Página dinâmica com PHP</h1>

 <p> Hello World! </p>
 <p> Hello World! </p>
 <p> Hello World! </p>
 <p> Hello World! </p>
 <p> Hello World! </p>

</body>
</html>
```

Saída produzida (página dinâmica)

Hello World

```
<html>
<body>
 ...
 <h1>Página dinâmica com PHP</h1>
 <p> Parágrafo 1 </p>
 <?php
   echo "<p> Parágrafo 2 </p>";
 ?>
 <p> Parágrafo 3 </p>
 <?php
   echo "<p> Parágrafo 4 </p>";
 ?>
</body>
</html>
```

```
<html>
<body>
  ...
  <h1>Página dinâmica com PHP</h1>
  <p> Parágrafo 1 </p>
  <p> Parágrafo 2 </p>
  <p> Parágrafo 3 </p>
  <p> Parágrafo 4 </p>

</body>
</html>
```

É possível ter múltiplos trechos de código PHP no arquivo (exemplo3.php)

Saída produzida (página dinâmica)

Recursos da Linguagem

Observações Gerais

- Declarações terminam com o ponto-e-vírgula
- Os tipos das variáveis são definidos automaticamente
- Comentários de linha: // comentário
- Comentários de bloco: /* comentário */
- Sensível a maiúsculas/minúsculas
  - Nomes de variáveis
  - Constantes
  - Chaves de arrays associativos
- Não sensível a maiúsculas/minúsculas
  - Palavras reservadas da linguagem
  - Nomes de métodos de classes
  - Nomes de funções

Operadores

Operador	Significado
+	Adição (e concatenação)
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira
++	Incremento
--	Decremento
=	Atribuição
+=	Atribuição com soma
-=	Atribuição com sub.
.	Concatenação
**	Exponenciação

Operador	Significado
==	Comparação por igualdade
===	Comparação por igualdade, incluindo valor e tipo
!= ou <>	Diferente
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a
&& ou and	"E" lógico
|| ou or	"Ou" lógico
!	Negação lógica

Estruturas Condicionais e de Repetição

```php
if (expressão) {
   // operações se verdadeiro
}
else {
   // operações se falso
}
```

```php
switch (expressao) {
    case condicao1:
        // operações
        break;

    case condicaoN:
        // operações
        break;
    ...
    default:
        // operações
}
```

```php
for ($i = 0; $i < 10; $i++)
{
   // operações
}
```

```php
foreach ($array as $elem)
{
   // operações
}
```

```php
while (expressao)
{
   // operações
}
```

```php
do {
   // operações
} while (expressao)
```

Variáveis

- Começa com o símbolo $
- Declarada na primeira atribuição
- Tipo obtido automaticamente
- Exemplo: $nome = "fulano";

Variáveis

```php
$x = 5;              // variavel do tipo integer
$y = -6;             // variavel do tipo integer
$z = 3.14;           // variavel do tipo float
$str = "bla bla";    // variavel do tipo string
$cond = true;        // variável booleana
$pares = [2, 4, 6];  // array
```

Constantes

- Definidas por meio da função **define**
- Possuem escopo **global**
- Por convensão, utilize maiúsculas

```php
<?php
    define('PATH_UPLOADS', '/arqs_uploads');
    define('DB_NAME', 'mysqlTeste');
    define('DB_PSWD', '123456');
?>
```

OBS: Dentro de classes, utilize ***const*** *PI = 3.14;*

Strings

- Com aspas duplas
  - Conteúdo da string é avaliado
  - Pode conter nomes de variáveis
- Com aspas simples
  - String não avaliada
  - Apenas texto
- Sintaxe Heredoc
  - Texto delimitador no início e no fim
  - Strings longas, múltiplas linhas, sem aspas
  - Conteúdo da string é avaliado

Strings e construtor ***echo***

```php
$idade = 15;
$mes = 10;
$dist = 30;

echo "A idade eh $idade";  // a saída será: A idade eh 15
echo 'A idade eh $idade';  // a saída será: A idade eh $idade

// Para imprimir uma variável sem espaço/caracter especial depois
// so seu nome, coloque o nome da variável entre chaves
echo "Distância de {$dist}km";   // Distância de 30km

// É possível separar os 'argumentos' por vírgula
echo $idade, $mes;                        // saída: 15 10
echo "Idade: ", $idade, "Mês: ", $mes;    // saída: Idade: 15 Mês: 10

// Também é possível concatenar os 'argumentos' com o operador .
echo "Idade: " . $idade . "Mês: " . $mes;
```

String ***Heredoc*** *(PHP < 7.3)*

```
$str = <<<BLOCO_HEREDOC

Texto com múltiplas linhas.
Aqui dentro posso utilizar aspas simples '
e também aspas duplas "
assim como nomes de variáveis como $exemplo

BLOCO_HEREDOC;
```

- BLOCO_HEREDOC poderia ser qualquer outro identificador
- Não deve haver espaços depois de <<<BLOCO_HEREDOC
- Não deve haver outros caracteres na linha de fechamento (PHP < 7.3)
  - **O identificador de fechamento não pode ser indentado!**

String ***Heredoc*** *(PHP ≥ 7.3)*

```php
$maxSal = 10;
$str = <<<SQL
    SELECT *
    FROM Funcionario
    WHERE salario > $maxSal and
          codDepart = 'Vendas'
    SQL;

echo $str;
```

A partir do PHP 7.3 o identificador de fechamento **pode ser indentado**

```
SELECT *
FROM Funcionario
WHERE salario > 10 and
      codDepart = 'Vendas'
```

Saída correspondente ao exemplo acima

String ***Heredoc*** *(PHP ≥ 7.3)*

```
$maxSal = 10;
$str = <<<SQL
  SELECT *
    FROM Funcionario
    WHERE salario > $maxSal and
          codDepart = 'Vendas'
    SQL;
```

Erro de sintaxe. *Primeira linha da string não começa com a quantidade mínima de espaços, conforme delimitador de fechamento.*

Funções

```php
function nomeDaFuncao($par1, $par2, ...) {
    // operações
    // operações
    // operações
}
```

Forma geral

```php
function max($a, $b) {
   if ($a > $b)
      return $a;
   else
      return $b;
}
```

Função de exemplo

```php
$maior = max(2, 5);
```

Chamada da função

Funções

```
function max(float $a, float $b) {
    if ($a > $b)
        return $a;
    else
        return $b;
}
```

Também é possível indicar os tipos dos parâmetros
Alguns tipos possíveis: `int`, `float`, `bool`, `string`, `object`

Passando Argumentos por Referência

```php
function atualizaGanhos(&$sal, &$comissão) {
    $sal = $sal * 1.2;
    $comissão = $comissão * 1.3;
}
```

Variáveis Globais

```php
<?php

    $x = 10;    // $x é global

    function exemplo() {
        echo $x;     // erro
        global $x;
        echo $x;     // 10
    }

?>
```

Necessário utilizar `global` para acessar variável global dentro de funções

Array Indexado

```php
$pares = [2, 4, 6];
```

Utilizando colchetes

```php
$pares = array(2, 4, 6);
```

*Com construtor **array***

```php
$pares[0] = 2;
$pares[1] = 4;
$pares[2] = 6;
```

Também pode ser declarado com atribuição direta às posições

```php
$n = count($pares);
for ($i = 0; $i < $n; $i++) {
    echo $pares[$i];
};
```

Percorrento array
*A função **count** retorna o número de elementos*

Array Indexado

```php
$pares = [];

for ($i = 2; $i < 100; $i=$i+2) {
    $pares[] = $i;
};
```

É possível adicionar elementos no final do array suprimindo o índice

Percorrendo *Array* com *foreach*

```php
$pares = [2, 4, 6, 8];
foreach ($pares as $par) {
    echo $par;
};
```

Array Associativo

```php
$alunos = array(
    "GSI010" => "Augusto",
    "GSI011" => "Camila",
    "GSI012" => "Pedro"
);

echo $alunos['GSI011'];   // a saída será 'Camila'
echo $alunos[0];          // ocorrerá um erro!
```

- Cada elemento é acessado por meio da chave (*key*)
- A chave pode ser *string* ou *inteiro*
- O valor pode ser de qualquer tipo

Arrays Super Globais

Arrays associativos especiais que podem ser acessados de qualquer lugar

Array	Descrição
$GLOBALS	Permite acessar as variáveis globais do script
$_GET	Acesso a campos de formulários submetidos com GET
$_POST	Acesso a campos de formulários submetidos com POST
$_FILES	Acessar a arquivos anexados a formulários
$_COOKIE	Acesso a cookies enviados ao servidor
$_SESSION	Acesso à variáveis de sessão
$_SERVER	Informações sobre o servidor, navegador, etc.

Arrays **Super Globais**

Array	Descrição
`$_SERVER['PHP_SELF']`	Nome do arquivo do script que está em execução
`$_SERVER['REQUEST_METHOD']`	Método da requisição HTTP utilizado para acessar a página
`$_SERVER['HTTP_USER_AGENT']`	Contém informações sobre o navegador
`$_SERVER['REMOTE_ADDR']`	Endereço IP do usuário que está acessando a página

Classes e Objetos

```php
class Circulo
{
    // declaração de atributos
    private $raio = 0;
    private $area = 0;

    // construtor
    function __construct($raio)
    {
        $this->raio = $raio;
        $this->area = 3.14*$raio*$raio;
    }

    // declaração de método
    public function mostraArea()
    {
        echo $this->area;
    }
}
```

```php
<?php

$circ1 = new Circulo(5);
$circ1->mostraArea();

?>
```

Processando Formulários

Recebendo Formulários

Formulário HTML - **Método POST**

```
<form action="processaForm.php" method="POST">
    Nome: <input type="text" name="nome" >
    Email: <input type="email" name="email">
</form>
```

```
<?php
    $nome = $_POST["nome"];
    $email = $_POST["email"];
    echo "Sr. $nome, seu e-mail é $email";
?>
```

Arquivo processaForm.php

Atenção: Utilize o atributo **name** para fornecer um nome de identificação para o campo.

Posteriormente, utilize essa identificação para recuperar o dado no PHP.

Um erro comum é utilizar o atributo id para esse propósito.

Recebendo Formulários

Formulário HTML - **Método GET**

```
<form action="processaForm.php" method="GET">
    Nome: <input type="text" name="nome">
    Email: <input type="email" name="email">
</form>
```

```
<?php
    $nome = $_GET["nome"];
    $email = $_GET["email"];
    echo "Sr. $nome, seu e-mail é $email";
?>
```

Arquivo processaForm.php

Formulários com Campos Vazios

```
<form action="processaForm.php" method="POST">
    Nome: <input type="text" name="nome">
    Email: <input type="email" name="email">
</form>
```

```
<?php
    $nome = $email = "";

    if (isset($_POST["nome"])
        $nome = $_POST["nome"];

    if (isset($_POST["email"])
        $email = $_POST["email"];

    echo "Sr. $nome, seu e-mail é $email";
?>
```

Campos vazios não são submetidos pelo método GET
A função **isset** pode prevenir erros nessas situações.

Formulários com Campos Vazios

```php
<?php
  $nome = isset($_POST["nome"]) ? $_POST["nome"] : "";
  $email = isset($_POST["email"]) ? $_POST["email"] : "";
  echo "Sr. $nome, seu e-mail é $email";
?>
```

Outra forma comum de uso da função **isset** com operador **?**

```php
<?php
  $nome = $_POST["nome"] ?? "";
  $email = $_POST["email"] ?? "";
  echo "Sr. $nome, seu e-mail é $email";
?>
```

Forma equivalente utilizando operador **??**

Validação dos Campos

```php
<?php
    $nome = trim($_POST["nome"]);

    // Não deixe de validar os campos
    if (empty($nome))
        $errorMsg = "O nome é obrigatório";

?>
```

A validação dos campos no lado servidor é recomendada mesmo que eles já tenham sido validados no lado cliente.
A função **trim** remove espaços do início e do fim da string.

Cuidado com Ataques **XSS**!

```
<html>
<body>
    <h1>Site Vulnerável à Ataques XSS</h1>
    <?php
        $nome = $_GET["nome"];
        $email = $_GET["email"];
        echo <<<HTML
        <table>
            <tr>
                <td> $nome </td>
                <td> $email </td>
            </tr>
        </table>
        HTML;
    ?>
</body>
</html>
```

Cross-Site Scripting (XSS)
Ataque malicioso onde um código potencialmente prejudicial é injetado utilizando a URL ou campos de formulário

Cuidado com Ataques **XSS**!

```
<html>
<body>
    <h1>Prevenindo XSS</h1>
    <?php
        $nome = $_GET["nome"];
        $email = $_GET["email"];
        ...
        $nome = htmlspecialchars($nome);
        $email = htmlspecialchars($email);
        echo <<<HTML
        <table>
            <tr>
                <td> $nome </td>
                <td> $email </td>
            </tr>
        </table>
        HTML;
    ?>
...
```

htmlspecialchars
Pode ser utilizada para "sanitizar" um conteúdo produzido pelo usuário, ***antes de inserir*** *esse conteúdo na página HTML*

htmlspecialchars *converte, por padrão:*

- *< em <*
- *> em >*
- *& em &*
- *" em "*

Mas Atenção: *não é uma boa prática armazenar o conteúdo (no banco de dados, por exemplo), após passar pela função* ***htmlspecialchars***

Recebendo Parâmetros pela URL

Exemplo de link gerado dinamicamente
com passagem de parâmetros pela URL

```
<a href="detalhes.php?codProd=1"> Notebook </a>
<a href="detalhes.php?codProd=2"> Celular </a>
```

```
<?php
    $codigoProduto = $_GET["codProd"];
    // busca pelo produto
?>
```

Resgate do parâmetro no arquivo detalhes.php

Manipulando Senhas

```php
<?php

// o hash gerado terá 60 caracteres, mas pode aumentar
$senhaHash = password_hash($senha, PASSWORD_DEFAULT);

// gravar no BD o hash da senha e não a senha
gravaNoBD($senhaHash);

?>
```

Cadastro do Usuário

*Uso da função **password_hash** do PHP para gerar um hash de senha seguro utilizando o algoritmo **BCrypt** antes de grav no BD*

```php
<?php

// resgatar do BD a senha hash do usuário
$senhaHash = resgataSenhaHashDoBD();

if (password_verify($senhaFornecida, $senhaHash) {
    // Login efetuado com sucesso!
}

?>
```

Validação de Login

*Uso da função **password_verify** para comparar a senha fornecida com o hash da senha armazenado.*

Redirecionando com PHP

```php
<?php
    // pagina.php
    header("Location: nova-pagina.php");
    exit();
?>
```

- **header** envia um cabeçalho HTTP
- Deve ser chamada antes do script produzir qualquer saída
  - Antes de código HTML, antes de ***echo***, etc.
- Não esqueça do **exit**() depois de redirecionar com header

Uso Inadequado da Função header

```
<html>
<body>
  <h1>Ex. de uso INADEQUADO da função header</h1>

  <?php
    header("Location: nova-pagina.php");
    exit();
  ?>

</body>
</html>
```

Não faça isso!

Neste exemplo a função header não terá efeito, pois o arquivo PHP produz uma saída HTML antes de sua chamada (por causa do código HTML no início do arquivo)

include e *require*

- **include** e **require** incluem outro arquivo PHP no script
- Arquivo incluído pode conter variáveis, funções, classes, etc.
- **include** gera um *warning* em caso de falha na inclusão
- **require** gera um *erro fatal* e encerra o script
- Não são funções!

```php
<?php
    // mysqlConnection.php pode conter
    // dados de conexão com o MySQL,
    // por exemplo
    include "mysqlConnection.php";
?>
```

include_once e *require_once*

- **include_once** e **require_once** são similares a *include* e *require*
- Porém, não incluem o arquivo caso ele já tenha sido incluído
- **include** e **require** produzem warnings/erros nessa situação

Referências

- https://www.php.net/docs.php
- NIXON, R. ***Learning PHP, MySQL & JavaScript****:* With jQuery, CSS & HTML5. 5. ed. O'Reilly Media, 2018