PHP - Módulo 1: Apresentação

Por Valdir Dias

Introdução

Antes de começarmos a falar do PHP, é interessante abordar o passado. Há muito tempo atrás, a informação na Internet era encontrada por meio das ferramentas Gopher (http://www.gopher.org). Talvez para a época isso fosse suficiente, mas hoje é bem difícil encontrar usuários dessa técnica.

O Gopher reinava absoluto até que surgiu a Web e o primeiro navegador, o Mosaic, que possuía a capacidade de exibir o texto e imagens, o que faria com que a Internet se aproximasse mais dos simples mortais. Surgia um novo jeito de formatar a informação contida nos servidores de forma mais agradável e ilustrada. Mas ainda assim, a informação era estática.

Tudo muito bonito, mas ainda faltava dinamismo às páginas. Assim surgiu o CGI, (Common Gateway Interface) programas executados sob requisição do navegador e a saída deste programa era encaminhada de volta ao navegador. Por exemplo: para exibir a data em uma página, era necessário executar um programa CGI nos servidor. Este programa lia a data do sistema e informava ao navegador que cuidava da exibição na tela. Para entender um pouco mais de CGI consulte os tutoriais disponíveis aqui.

Esta dupla HTML/CGI durou até fins de 1998, início de 1999, sendo utilizada até hoje.

E, como sempre, alguém inventou algo melhor. Os programas CGI eram escritos, em sua maioria, com as linguagens C e PERL. E, por isso, causavam uma série de problemas nos servidores, que não vêm ao caso, mas para exemplificar: imagine um site que receba 10 visitas por segundo. Se ele executar um programa CGI em cada uma dessas visitas, serão abertos 10 processos por segundo, o que podia fazer com que o servidor gastasse mais tempo gerenciando estas "threads" do que servindo páginas, função para o qual fora projetado.

E para resolver este problema tiveram a brilhante idéia de inserir os comandos que seriam o programa CGI na página HTML, de modo que o servidor, ao enviar esta página, executasse o bloco do comando, no mesmo processo. Deste modo, para exibir a data em nossa página, bastava inserir, em algum ponto desta página, o comando para que a data fosse inserida.

Aqui começamos o assunto. Hoje podemos usar comandos da linguagem, que é grátis, portável, aberta, escalável.

Como você já deve estar curioso, vamos ver o código de uma página com uma porção PHP.

```
1 <html>
2 <body bgcolor=white>
3 <center>
4 < 5 </center>
6 </body>
7 </html>
```

Vamos "destrinchar" esta página:
Se você não entendeu as três primeiras linhas, vale a pena dar uma olhada no tutorial HTML. A novidade está na linha 4. Note que ela se assemelha com uma TAG HTML, mas tem suas particularidades, por exemplo, um sinal de interrogação para abrir e para fechar. Isto faz com que esta página, quando estiver sendo enviada pelo servidor, passará pelo interpretador do PHP, de modo que apenas código HTML puro chegue ao navegador. Se o usuário acionar o "View Source" de seu navegador, verá o seguinte:

```
1 <html>
2 <body bgcolor=white>
3 <center>
4 Hello World!
5 </center>
6 </body>
7 </html>
```

Este exemplo, não tem muita utilidade, mas serve para ilustrar a idéia por trás do PHP.

Vamos ver algo mais útil:
Digamos que em nosso site exista uma página secreta, que só pode ser aberta por quem possuir o código de acesso. Como o HTML não traz nenhuma função para isso, teremos de fazer esta validação usando um outro programa. Este programa deverá ler o valor do campo LOGIN e SENHA do HTML e exibir a página secreto.htm se o par login/senha estiver correto ou a página senhainvalida.htm caso contrário. Pelo que já vimos, isto pode ser feito usando um CGI ou um script in-line (ASP, PHP, etc). Começaremos usando um CGI em PERL:

Código da página login.htm

```
<html>
<body bgcolor=white>
<center>
<form action=login.pl method=post>
Login: <input type=text name=login><br>
Senha: <input type=text name=senha><br>
<input type=submit value=" Entrar ">
</form>
</center>
</body>
</html>
```

Vamos ver como ficaria o código do CGI login.pl

```
#!/usr/bin/perl
read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'}); @pares = split(/&/, $buffer);

foreach $par (@pares) {
($campo, $valor) = split(/=/, $par);


$valor =~ tr/+/ /;
$valor =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
$valor =~ s///g;


$FORM{$campo} = $valor;
}

if (($FORM{'login'} ne 'scott') || ($FORM{'senha'} ne 'tiger'))
{
print "Location: senhainvalida\.htm\n\n";
} else {
print "Location: paginasecreta\.htm\n\n";
}
```

E vejamos como seria o código em PHP. Para isto, o valor ACTION do formulário teria que ser mudado para login.php3.

```
<?php
If (($login <> "scott") || ($senha <> "tiger")) {
Header("Location: senhainvalida.htm\n\n");
} else {
Header("Location: paginasecreta.htm\n\n");
}
?>
```

Vamos discutir o código acima:

A primeira linha é padrão, ela abre um bloco de código PHP. A segunda, faz a comparação dos valores de login e senha. Repare que para recuperar o valor dos campos de formulário basta referenciar aos nomes destes campos como variáveis do PHP. Isto significa que a variável $login conterá o valor digitado no campo LOGIN do HTML e a $senha a mesma coisa, não sendo necessário tratar todo o "Buffer" do HTTP, como é feito no PERL.

Dependendo do resultado da comparação da linha 2, o programa seguirá pela linha 3 ou 5. Se o login ou a senha não estiver correto, é executada a linha 3 que devolve um comando para o navegador abrir a página senhainvalida.htm, do contrário, o fluxo é desviado para a linha que dá o camando para o navegador exibir a página secreta.

Lógico que este não é um método seguro de se proteger uma página, mas em termos didáticos é suficiente.

Veja no próximo módulo

Bom, já vimos algumas coisas interessantes do PHP. No próximo módulo, iremos tratar de variáveis e constantes, além de ver mais algumas coisas relacionadas à tratamento de formulários. Até lá!