

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Introdução ao PHP – Parte 4

Como criar variáveis globais e estáticas e transferir valores entre páginas

Essa é a última parte de nosso mini-curso *Introdução ao PHP* onde estamos vendo os principais detalhes da linguagem. Neste artigo aprenderemos como trabalhar com mais de uma página PHP transferindo informações de uma página para outra através de variáveis globais e estáticas, além de formulários.

Veremos também como criar *packages* (“pacotes”), que na verdade são páginas PHP contendo variáveis e funções a serem utilizadas. Entenderemos como o PHP se comporta ao efetuar “includes” de arquivos contendo código de funções e variáveis.

Entendendo as variáveis

As variáveis não são “visíveis” na aplicação inteira. Este é um aspecto muito importante, pois, caso contrário, os programadores acabariam tendo complicações. Ao construir aplicações grandes é importante saber onde e quais variáveis podem ser vistas e utilizadas. Começemos

com um exemplo simples na **Listagem 1**.

Em primeiro lugar, *\$a* é inicializada e exibida na tela. Então a função *display_data* é chamada, e tenta exibir *\$a* novamente e o que temos é o resultamos a seguir:

```
Mensagem principal: Olá Mundo!  
dados:
```

Como podemos observar *display_data* não consegue exibir nada, pois *\$a* não está visível para ela. A variável *\$a* foi inicializada no script principal e, portanto, não é visível para funções chamadas ele. Por este motivo, se desejarmos acessar *\$a* em *display_data*, devemos então, declará-la como global assim como podemos ver na **Listagem 2**.

Com a ajuda da palavra-chave *global*, é possível ler e modificar qualquer variável global:

```
Mensagem principal: Olá Mundo!  
dados: Olá Mundo!  
Mensagem principal: Olá Adriano
```

Ewald Geschwinde e
Hans-Juergen Schoenig

Como podemos ver o conteúdo de *\$a* pode ser modificado por *display_data*. Outro modo de acessar a variável seria utilizar um *array* chamado *\$GLOBALS* como podemos ver na **Listagem 3**.

Com a ajuda de *\$GLOBALS*, é possível acessar qualquer variável que foi definida globalmente. Executando o *script*, a saída será exibida na tela:

```
Mensagem Principal: Olá Mundo!
dados:Olá Mundo!
```

Outro aspecto importante a ser considerado, especialmente quando lidamos com projetos muito grandes, é que podemos incluir outros arquivos no script usando o comando *include*, desta forma variáveis hospedadas em outros arquivos podem ser visualizadas.

Com a ajuda do comando *include*, os arquivos que contêm funções e variáveis são acrescentados a um *script* PHP e serão visíveis pelo interpretador. Na segunda linha da **Listagem 3** estamos incluindo o arquivo *vars.php* presente na mesma pasta onde o script atual está sendo executado.

Experimente criar um arquivo *.php* e digitar nele o código a seguir. Note que definimos uma variável chama *\$a* cujo o conteúdo é "Olá Mundo!".

```
<?php
# Defining a string
$a = "Olá Mundo!";
?>
```

Crie um novo arquivo e insira o código da **Listagem 3** antes mencionada. Salve-o e execute-o no browser. É recomendável colocar arquivos que contêm somente definições de variáveis em um diretório separado, para que o mesmo não seja misturado com arquivos que contêm outros códigos-fonte. Além do mais aumenta a segurança do site. Executando o *script*, veremos que o resultado será o mesmo do exemplo anterior:

```
Mensagem Principal: Olá Mundo!
dados: Olá Mundo!
```

Incluir um arquivo é simples, mas o que acontece quando o arquivo incluso em *script.php* contém, por sua vez, arquivos outros arquivos inclusos? Vejamos no código a seguir uma nova abordagem. Para entendermos como

Listagem 1. Exemplo de variável não visível a toda aplicação

```
<?php
$a = "Olá Mundo!";
echo "Mensagem principal: $a<br>\n";
display_data();

function display_data(){
    echo "dados: $a<br>\n";
}
?>
```

Listagem 2. Declaração de variável global

```
<?php
$a = "Olá Mundo!";
echo "Mensagem principal: $a<br>\n";
display_data();
echo "Mensagem principal: $a<br>\n";

function display_data(){
    global $a;
    echo "dados: $a<br>\n";
    $a = "Olá Adriano";
}
?>
```

Listagem 3. Usando o array \$GLOBALS

```
<?php
include("inc/vars.php");
echo "Mensagem Principal: $a<br>\n";
display_data();

function display_data(){
    echo "dados:". $GLOBALS["a"];
}
?>
```

Listagem 4. Exemplo de variáveis estáticas

```
<?php
for($i=1; $i<=6; $i++){
    echo display_data($i)."<br>\n";
}

function display_data($i){
    static $var;
    $var .= "$i";
    return $var;
}
?>
```

o PHP se comporta nesses casos, crie dois novos arquivos PHP com os nomes "base_a.php" e "base_b.php". No primeiro arquivo, *base_a.php*, digite o código a seguir:

```
<?php
$a = "Olá Mundo!";
?>
```

Já no arquivo *base_b.php* digite:

```
<?php
$b = "Somos do Brasil!";
?>
```

Basicamente estamos definindo, separadamente, o valor de duas variáveis: *\$a* e *\$b*. Em seguida modifique o script do arquivo *vars.php* conforme segue:

```
<?php
include("base_a.php");
include("base_b.php");
?>
```

Desta vez o *vars.php* não contém a definição das variáveis, mas inclui dois arquivos que contêm esta definição de *\$a*

e *\$b*. Por fim crie um novo arquivo PHP que implementa o código a seguir:

```
<?php
include("vars.php");
echo $a<br>";
echo $b;
?>
```

Nesse caso estamos incluindo o arquivo *vars.php* que contém inclui os dois arquivos criados anteriormente que por sua vez contêm os valores das variáveis *\$a* e *\$b*. O que acontece em seguida é que ambas variáveis são exibidas no browser. Isso acontece porque as variáveis incluídas no arquivo *vars.php* também são adicionadas nesse script. Veja a seguir:

```
Olá mundo!
Somos da América
```

Entendendo e usando variáveis estáticas

As variáveis estáticas, assim como muitos outros componentes do PHP, foram emprestadas do C/C++. Essas

variáveis são utilizadas para construir funções que serão capazes de lembrar-se de valores. Especialmente para as recorrências, característica essa muito importante em qualquer sistema. Para demonstrar o que podemos fazer com variáveis estáticas, incluímos um breve exemplo na **Listagem 4**.

Uma repetição garante que `display_data` será chamada seis vezes. A variável `$var` foi definida como uma variável estática, o que significa que lembrará do valor que teve da última vez em que `display_data` foi chamado. Sempre que a função for chamada, o valor atual de `$i` é acrescentado a `$var`. Se executarmos a página, veremos que o tamanho de `$var` cresce sempre que `display_data` é processado conforme podemos ver a seguir:

```
1
12
123
1234
12345
123456
```

As variáveis estáticas têm tanto desvantagens quanto vantagens. Erros de execução provocados por variáveis está-

ticas são, às vezes, difíceis de encontrar, e pode ser muito trabalhoso procurar erros que só são encontrados por depuração extensiva.

Criação de Packages (Pacotes)

Mesmo escrevendo aplicações bem pequenas, é recomendado empacotar em arquivos separados as funções utilizadas para certas partes das aplicações. Ao escrevermos uma aplicação utilizada para exibir diagramas de estoque, por exemplo, pode ser útil ter bibliotecas separadas de funções relacionadas à recuperação de dados e outras para exibir os dados. Isto nos ajudará a reutilizar o código para outras aplicações. Além do mais, as nossas funções serão mais consistentes, pois nem toda aplicação tem a sua própria implementação.

Escrever código reutilizável é o primeiro passo para obtermos código confiável. Quanto mais freqüentemente um trecho de código é utilizado, mais maduro estará, pois terá sido mais testado. Outro aspecto importante é que os pacotes devem ser independentes dos progra-

mas que os usam. Em outras palavras, não devemos acessar variáveis globais do programa principal a partir de uma biblioteca, pois o código poderá não mais ser reutilizável e o programador que utiliza uma biblioteca deve saber o que está acontecendo dentro da mesma.

Devemos tentar construir módulos pequenos, compreensíveis e independentes, com uma interface fixa e garantir que o usuário de uma biblioteca não irá saber o que está acontecendo dentro da mesma.

O fragmento de código da **Listagem 5** mostra um pequeno exemplo. O objetivo é escrever um módulo que troca o conteúdo de duas variáveis. Vejamos antes o programa principal:

No início do `script`, uma biblioteca está sendo incluída e nela duas variáveis são definidas. Estas são as variáveis que desejamos trocar. Então a função `swapdata_ref` é chamada e os valores são passados para a função. Depois disto, as variáveis são exibidas para que possamos verificar que foram realmente trocadas.

Até agora, o `swapdata_ref` é uma caixa preta, e não será necessário saber como as variáveis são trocadas, porque a interface da função foi claramente definida. Contudo, daremos uma espiada no que a função faz internamente observando a **Listagem 6**.

Uma referência é passada para a função e os valores serão trocados utilizando uma variável temporária chamada `$c`. Como podemos notar a função não tem nenhum retorno. Passada a `swapdata_ref` uma referência ela é capaz de acessar as variáveis diretamente (utilizando os endereços internos dos valores), sem necessidade de acessá-las utilizando variáveis globais.

Este modo de resolver o problema é muito cômodo, porque `swapdata_ref` não precisa saber como é que são chamadas as variáveis na função chamadora. Os endereços dos valores que têm de ser processados serão destinados aos nossos novos nomes quando a função for chamada. Executando o arquivo, podemos ver que as variáveis foram trocadas:

```
a: 37
b: 23
```

Listagem 5. Fragmento de código de biblioteca

```
<?php
include("lib/transform_data.php");
$a = 23;
$b = 37;
swapdata_ref($a, $b);
echo "a: $a<br>";
echo "b: $b";
?>
```

Listagem 6. Código da função `swapdata_ref`

```
<?php
function swapdata_ref(&$x, &$y){
    $c = $x;
    $x = $y;
    $y = $c;
}
?>
```

Listagem 7. Formulário em HTML

```
<html>
<body>
    Um simples formulário
    <br><br>
    <form action="reaction.php" method="POST">
        <input type="text" name="nome" size="10"><br><br>
        <input type="submit" name="submitbutton">
    </form>
</body>
</html>
```

Listagem 8. Código do script `reaction.php`

```
<?php
if($nome){
    echo "nome: $nome";
}
else{
    echo "Nada foi digitado no campo.";
}
?>
```

Como em outras funcionalidades, as referências foram herdadas do C e são uma característica poderosa, pois seria pouco confortável trabalhar com variáveis globais ou passagem de enormes quantidades de dados para uma função.

Trabalhando com formulários

Até agora, vimos como escrever aplicações PHP simples. Contudo, para construir *websites* interativos, é necessário permitir que o usuário digite informações através dos, assim chamados, formulários. O conteúdo de tais formulários pode ser transmitido ao servidor e um *script* PHP pode reagir baseado nos dados enviados a ele. A interação com um servidor *Web* pode ser feita via dois métodos chamados *GET* e *POST*.

Os métodos *GET* e *POST* são descritos no documento *RFC2068* que descreve como devem se comportar. Isso faz parte da regulamentação do protocolo HTTP e pode ser lido no link www.faqs.org/rfcs/rfc2068.html que contém mais de 160 páginas. Nesta seção aprenderemos sobre

as idéias principais por trás do *GET* e *POST* e aprenderemos também a utilizar esses dois métodos eficientemente. Vejamos o que cada um significa:

- *Get*: O método *GET* é o método de solicitação padrão para recuperar dados de um servidor *Web*. Quando chamamos um *website*, o *GET* é utilizado para obter o documento selecionado. Normalmente, as chamadas deste tipo não têm efeitos colaterais e é por este motivo que *GET* deve ser utilizado; Os navegadores assumem que *GET* não tem nenhum efeito colateral e se a página não mais estiver no *cache* do navegador, será então recuperada novamente. Contudo, se a solicitação original fosse via o *POST*, o usuário receberia uma mensagem alertando que o documento não está mais no *cache*. Mais adiante veremos como trabalhar com ambos métodos;

- *Post*: O *POST* é o método padrão para submeter dados ao servidor *Web* fornecido via formulário. No caso do *POST*, a solicitação sempre contém uma seção *body* ("corpo") onde a informação

relacionada à mesma é fornecida. Esta informação é codificada como uma *query string* ("string de consulta").

Normalmente, os desenvolvedores *Web* usam o *POST* mesmo quando nenhum dado no servidor tiver sido modificado;

Criando formulários

Depois que vimos quais métodos são utilizados para recuperar dados de um servidor *Web*, daremos uma olhada em um formulário simples, que pode ser utilizado para enviar o texto a um arquivo PHP. A **Listagem 7** mostra um formulário simples em HTML. Salve o arquivo como "formulario.htm".

Podemos ver que um formulário foi definido na quinta linha através da *Tag* `<form>` do HTML. Quando o usuário clicar no botão *Submit*, os dados serão enviados para o *script reaction.php*. O formulário contém dois componentes. O primeiro componente é o campo de texto chamado *nome*. O segundo componente é o botão para submeter o formulário. O

Cursos Online

Assinatura

ClubeDelphi PLUS

Mais conteúdo DELPHI por muito menos!

A Revista **Clubedelphi** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade .

Conheça abaixo os cursos já disponíveis:

www.devmedia.com.br/curso/cdplus

- Aplicações Client/Server com dbExpress e Firebird
- Sistema SysCash
- Criando uma aplicação multi-camadas completa com Delphi
- Aplicações client/server com Windows Forms no Delphi 2006
- Aplicações WEB com IntraWeb e Delphi 7 (Delphi Win32)

A sua melhor opção de aprendizagem!

Assine a **Clubedelphi** e
Comece já seu treinamento!
www.devmedia.com.br/assine



Mais Informações: www.devmedia.com.br/central - Tel.: 21 2220-5375 / 2220-5435

Listagem 9. Formulário de cadastro

```
<html>
<body>
  Formulário mais complexo
  <br><br>
  <form action="reaction.php" method="POST">
    <input type="text" name="nome" size="10"><br>
    <hr>
    <input type="checkbox" name="box_1">
      display time<br><br>
    <input type="radio" name="myradio" value=1>Value 1<br>
    <input type="radio" name="myradio" value=2>Value 2<br>
    <input type="radio" name="myradio" value=3>Value 3<br><br>
    <input type="password" name="passwd">Enter passwords here<br><br>
    <input type="file" name="filename">enter the filename<br><br>
    <input type="reset" name="resetbutton">
    <input type="submit" name="submitbutton">
  </form>
</body>
</html>
```

Listagem 10. Código alterado do reaction.php

```
<?php
echo "nome: $field_1<br>";
echo "myradio: $myradio<br>";
echo "passwd: $passwd<br>";
echo "filename: $filename<br>";
?>
```

Listagem 11. Novo formulário com Seleção

```
<html>
<body>
  Seleção:
  <br><br>
  <form action="reaction.php" method="POST">
    <select name="fruits[]" multiple size=4>
      <option value="Orange">Orange
      <option value="Banana">Banana
      <option value="Apple">Apple
      <option value="Pineapple">Pineapple
      <option value="Strawberry">Strawberry
      <option value="Cherry">Cherry
      <option value="Coconut">Coconut
    </select><br><br>
    <input type="submit" name="submitbutton">
  </form>
</body>
</html>
```

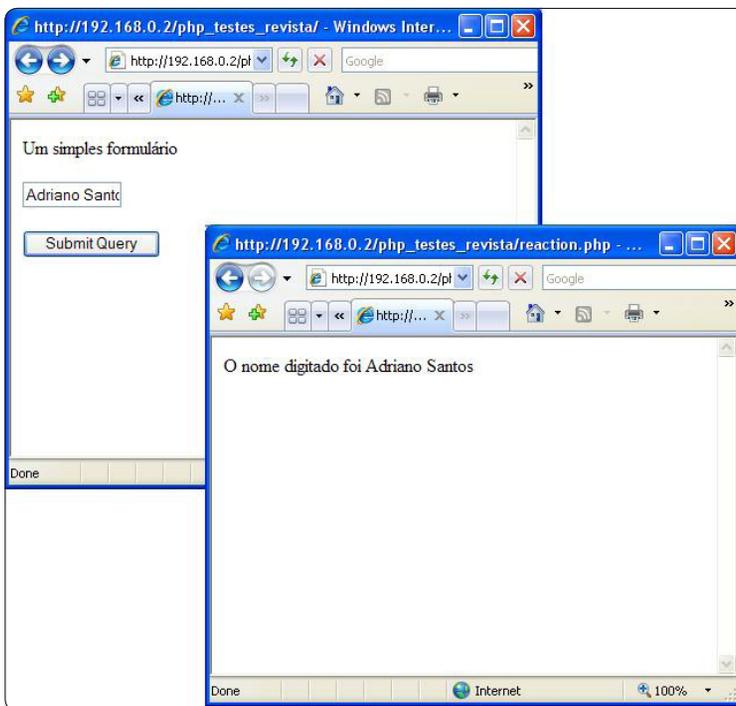


Figura 1. Arquivo de cabeçalho

fim do formulário é marcado pela Tag `</form>`. Crie agora um novo arquivo chamado "reaction.php" e digite o código da Listagem 8. Abra o arquivo *formulario.htm* no browser, digite um texto qualquer e clique no botão. Observe pela Figura 1 que o texto é passado e visto na página *reaction.php*.

A primeira coisa que o *script* faz, é verificar se *\$nome* foi definido, ou seja, digitado. Se o usuário tiver inserido dados no formulário, a variável será automaticamente definida pelo PHP. Em contraste com o Perl, os programadores PHP não precisam extrair as variáveis eles próprios a partir da *query string*, todo o trabalho é feito pelo PHP. Se *\$nome* foi definido, o conteúdo da variável será exibido na tela.

Como podemos ver, a recuperação de dados de um formulário é uma tarefa fácil. A seguir veremos como formulários mais complexos são construídos com o auxílio do HTML. Modifique o *formulario.htm* de maneira que seu código HTML seja igual a Listagem 9. Já o arquivo *reaction.php* deverá se tornar igual ao código da Listagem 10. Faça as alterações necessárias e refaça o teste.

O primeiro componente definido no formulário é novamente um campo de texto. O campo tem comprimento 10. Além do tamanho do campo, também seria possível definir o comprimento máximo do texto que se permite que o usuário insira. Isto pode ser feito utilizando o atributo *maxlength*. Depois de definir o campo de texto, um *Checkbox* chamado de *box_1* é definido. Em seguida, podemos ver como os *RadioButtons* são acrescentados a um documento HTML. No exemplo, podemos ver que *myradio* consiste de três componentes. Só um dos três componentes pode ser ativado. Dependendo de qual dos três botões é marcado, o valor apropriado é enviado para *reaction.php*.

Se campos para inserir senhas forem necessários, o componente *Password* será do tipo de componente apropriado. Enquanto estiver digitando, o usuário só verá asteriscos, portanto, ninguém

poderá capturar a senha. Para selecionar um arquivo no disco rígido do usuário, o tipo chamado *file* pode ser utilizado. O componente *file* criará uma caixa de texto e um botão, no qual o usuário irá clicar, caso seja necessária uma janela para selecionar o arquivo usando uma interface gráfica. Finalmente, o botão *Submit* é acrescentado ao formulário. A **Figura 2** mostra o formulário criado pelo código da **Listagem 10**.

O script exibe os valores na tela conforme o que podemos ver em seguida:

```
nome: 23
myradio: 2
passwd: a password
filename: /home/hs/boot.img
```

Repare que os valores na listagem, mostram os dados inseridos no formulário. O importante no exemplo, é que a informação de cada campo no formulário HTML, será armazenada em uma variável após ser submetida a um arquivo PHP.

Às vezes, é necessário dar ao usuário a oportunidade de selecionar mais do que apenas um valor em uma lista. Por isso, o HTML oferece uma solução simples utilizando *select* e *option*. Vamos fazer um novo exemplo. Altere novamente o arquivo *formulario.htm* digitando o código da **Listagem 11**. Em seguida altere também o código do arquivo *reaction.php* usando o código a seguir:

```
<?php
    foreach ($fruits as $var){
        echo "$var<br>";
    }
?>
```

Agora o usuário pode escolher algum dos frutos apresentados na lista. Desde que mais de um fruto pode ser selecionado, a estrutura de dados utilizada pelo PHP deve ser um *array*. No nosso exemplo, chamamos este *array* de *fruits[]*. Depois de submeter o conteúdo do formulário, o *array* conterá todos os valores que o usuário selecionou.

Passando parâmetros para um script

Em muitos casos, é interessante chamar um *script* e passar-lhe alguns parâmetros. Técnicas como estas, muitas vezes serão necessárias para os *banners*, pois a pessoa que pagou pelo *banner* quer

saber de onde vêm os cliques. Com este objetivo, os parâmetros serão passados para um *script* que contém a informação de qual *website* contém o *banner*. Vejamos um *script* muito simples, que não faz nada exceto exibir duas variáveis:

```
<?php
    echo "a: $a<br>\n";
    echo "b: $b\n";
?>
```

O objetivo é passar parâmetros para o script via uma URL como por exemplo:

```
http://localhost/test/script.php?a=234&b=197
```

O *script* chamado *script.php*, localizado em um diretório denominado *test* no servidor Web local, é chamado com o parâmetro *a=237* e *b=197*. Os nomes do *script* e os parâmetros são separados utilizando um caractere "?" (sinal de interrogação). A lista de parâmetros é separada utilizando o caractere "@" (arroba). Executando *script.php*, o resultado será exibido pelo navegador:

```
a: 234
b: 197
```

Nós já vimos que as marcas de sinal

de interrogação e arroba são utilizados como delimitadores. O que acontece se desejamos passar um desses sinais para o *script*? O servidor Web irá confundir-se, pois terá que descobrir os símbolos que são utilizados como delimitadores daqueles que são partes de uma string. A recomendação é: todos os caracteres "especiais" têm que ser evitados para que o servidor Web possa analisar a URL facilmente.

O PHP provê uma função chamada *urlencode*, que toma conta deste tipo de situação. Vamos escrever um pequeno *script* que gera uma lista de URLs: Portanto crie uma nova página e nela digite o código da **Listagem 12**.

Depois de exibir o código HTML, cinco *strings* serão definidas e atribuídas a um array chamado *\$messages*. Cada uma das *strings* contém certos caracteres que têm que ser evitados pelo PHP. No passo a seguir, as conexões serão geradas utilizando o *urlencode*. Vamos chamar o PHP via linha de comando e verificar o aspecto do código HTML gerado pelo *script*. Digite na barra de endereços uma das *urls* a seguir:

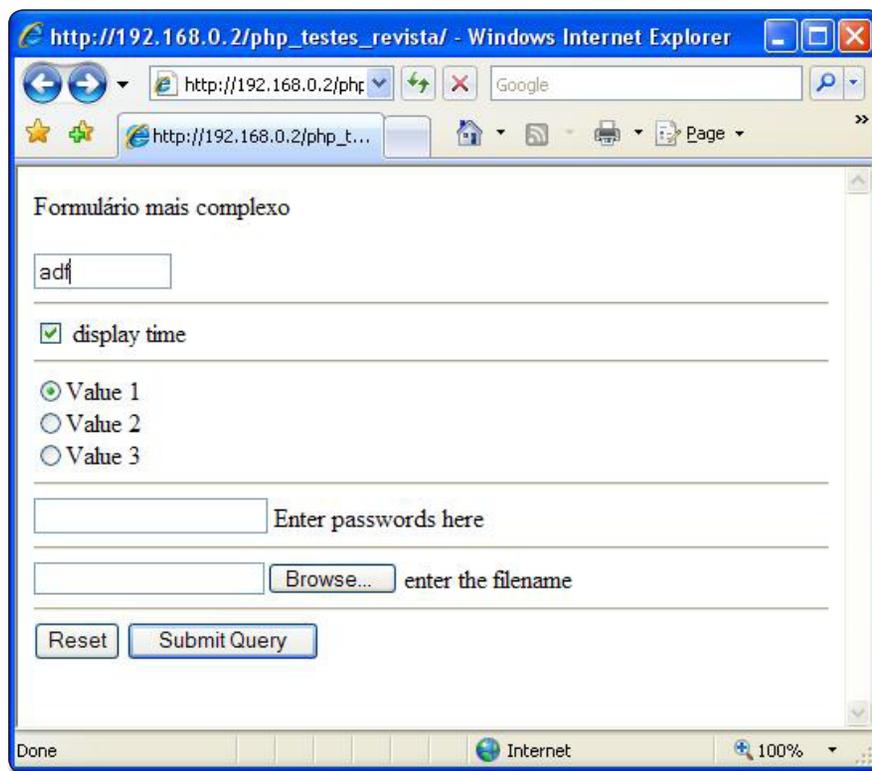


Figura 2. Formulário um pouco mais complexo

```

http://localhost/reaction.
php?a=Pat+and+Shelley
http://localhost/reaction.
php?a=Pat+%26+Shelley
http://localhost/reaction.
php?a=Pat+%26+Shelley%21
http://localhost/reaction.
php?a=Are+you+sure%3F
http://localhost/reaction.php?a=Hans-
J%FCrgen+Sch%F6nig

```

Como podemos ver, todos os caracteres especiais foram evitados. Executando o *script* usando um navegador *Web*, as *strings* no *array* serão exibidas como links:

```

Pat and Shelley
Pat & Shelley
Pat & Shelley!
Are you sure?
Hans-Jürgen Schönig

```

Se clicarmos no primeiro link, o *script.php* será chamado:

```

a: Pat and Shelley
b:

```

O texto é exibido na tela. Como não foi passado ao *script* nenhum parâmetro para *\$b*, o campo estará vazio.

Trabalho com campos ocultos

Às vezes, é necessário passar parâmetros para um *script* que não devem ser vistos pelo usuário. No HTML, é possível utilizar campos ocultos. A diferença entre campos “normais” e campos ocultos é que os campos ocultos não serão exibidos pelo navegador. Por isso, os parâmetros são facilmente passados de um *script* para o outro sem que o usuário consiga modificar a informação contida em um campo oculto. Supondo que desejamos escrever um *script* que exibe a data e hora em que o *script* anterior foi criado (**Listagem 13**)

O primeiro *script* gera a data e hora atuais e os armazena em *\$timestr*. Agora, um formulário é gerado e o valor de *\$timestr* é utilizado como o valor *default* do chamado *gentime*. As únicas informações exibidas na tela serão: o conteúdo de *\$timestr*, a string “Olá Mundo!” e um botão para submeter a informação. O *reaction.php* é chamado quando o usuário clica no botão. Vejamos o *reaction.php*:

```

<?php
echo "Hora gerada: $gentime";
?>

```

O conteúdo de *\$gentime* é exibido na tela:

```

gentime: 2007 Nov 01: 12:31:57 CET

```

O *script* anterior, foi gerado na data e hora listadas na saída de *reaction.php*. Às vezes, é interessante passar a data e hora de geração do primeiro HTML para todos os arquivos, pois deste modo, é possível descobrir quando um usuário acessou a página.

Conclusão

O PHP é uma linguagem poderosa e fácil de se trabalhar. As vantagens principais do PHP são a sua flexibilidade e a sua simplicidade. As aplicações *Web* podem ser escritas fácil e rapidamente. Isto faz com que o PHP se torne uma boa escolha para a rápida prototipação e o desenvolvimento de aplicações.

Veremos muitos outros artigos sobre a linguagem PHP, porém é de suma importância que se estude cada vez mais o *script*, procurando entender melhor como tudo funciona. Procure desenvolver aplicações simples com formulários, transferência de valores entre as páginas, uso de sessões e uma série de outras funcionalidades disponíveis no PHP. Bons códigos e até a próxima. ●

Listagem 12. Usando o urlencode

```

<?php
echo "<html><body>\n";

$messages[0] = "Pat and Shelley";
$messages[1] = "Pat & Shelley";
$messages[2] = "Pat & Shelley!";
$messages[3] = "Are you sure?";
$messages[4] = "Hans-Jürgen Schönig";

foreach ($messages as $var){
echo '<a href="script.php?a='.urlencode($var).
"> '.$var.'"</a><br>\n";
}
echo '</body></html>';
?>

```

Listagem 13. Script criado com campo hidden, oculto

```

<?php
$curtime = localtime();
$timestr = strftime("%Y %b %d: %T %Z");
echo "timestr: $timestr<br>\n";
echo '
<html>
<body>
Olá mundo!
<form action="reaction.php" method="POST">
<input type="hidden" value="'. $timestr.
'" name="gentime">
<input type="submit" name="submitbutton">
</form>
</body>
</html>
';
?>

```

