

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

## Introdução ao PHP – Parte 3

Criação de funções, tratamento de exceções e uso de arquivos

Funções são a chave para reusabilidade em qualquer linguagem de programação. Com PHP não é diferente, pois através de funções podemos centralizar nossas principais rotinas, tratamentos, regras de negócio etc. Nós já vimos na edição 89 como fazer uso de funções do próprio PHP, porém, até agora não implementamos nossas próprias funções. Nesse artigo aprenderemos a construí-las em PHP.

### Funções Simples

Começemos com um exemplo simples: a *Fórmula de Gauss* para a soma de números inteiros. Com sete anos de idade, Gauss iniciou a escola elementar, e o seu potencial foi notado quase imediatamente. O seu professor, Büttner, e o seu assistente, Martin Bartels, ficaram impressionados quando Gauss somou os números inteiros de 1 até 100, imediatamente deduzindo que a soma é formada por 50 pares de números, cada par somando 101. Assim

nasceu a Fórmula de Gauss para a soma de números inteiros (**Listagem 1**).

Na primeira linha vemos a função de *Gauss* sendo chamada. Um parâmetro deve ser passado à função. Este parâmetro definirá o limite superior da soma que desejamos computar. Se o parâmetro passado para a função for um valor inteiro e se for maior do que o zero, o resultado é computado e retornado para a função principal. O resultado é passado para a função principal utilizando o comando *return*. Neste caso será retornado o valor 10 e a seguir, será exibido na tela o texto:

Soma de 1 até 4: 10

Podemos também chamar funções passando mais de um parâmetro. O exemplo da **Listagem 2** mostra uma versão ligeiramente adaptada da função que vimos anteriormente. Com a ajuda desta função, é possível computar a soma de valores inteiros, iniciando de um valor inicial e atingindo um valor final maior (o valor

Ewald Geschwinde  
Hans-Juergen Schoenig

inicial não está incluído na soma). Se executarmos o script, o resultado exibido na tela será:

Resultado: 45

## Passando arrays para uma função

Às vezes é necessário passar *arrays* inteiros para uma função. O PHP não suporta a sobrecarga (*overload*) de função, como em Delphi, por exemplo, mas com a ajuda de *arrays*, é possível construir soluções de contorno bem simples. O objetivo do exemplo da **Listagem 3**, é computar a média geométrica de um *array* de valores:

Em primeiro lugar, é criado um *array* que contém vários valores. A seguir, a função *geomean* é chamada e o *array* é passado para a mesma. A função *geomean* computa a soma das segundas potências dos valores no *array*. Depois disso, a raiz quadrada é computada e retornada para a função principal. O resultado será exibido na tela:

Resultado: 3.8729833462074

## Retornando mais de um Valor

Até agora, lidamos com funções que retornam apenas um valor. Contudo, podemos ter funções que retornam vários valores. Vejamos no exemplo na **Listagem 4**.

Em primeiro lugar, um *array* é gerado e passado para a função chamada *compute*, a qual executa algumas operações simples e retorna um *array* de valores. A saída é exibida na tela:

```
valor: 5
valor: 20
valor: 400
valor: 9600
valor: 153600
```

Como podemos ver, não faz nenhuma diferença se quisermos retornar uma variável comum ou um *array* de valores, a sintaxe é a mesma.

## Tratando exceções

O tratamento de exceções é uma característica fundamental de quase toda linguagem de programação. O tratamento de exceções não é necessário somente para descobrir erros, mas também para o controle de fluxo. O PHP fornece um modelo poderoso para tratamento de exceções, que pode ser facilmente utilizado por programadores. Para mostrarmos como

### Listagem 1. Fórmula de Gauss

```
<?php
$result = gauss(4);
echo "Soma de 1 até 4: $result<br>\n";
# função para calcular a soma de 1 para $upper
function gauss($upper){
    if(is_int($upper) && ($upper > 0)){
        return($upper*($upper+1)/2);
    }
}
?>
```

### Listagem 2. Fórmula de Gauss com mais de um parâmetro

```
<?php
$lower = 4;
$upper = 10;

$result = gauss($lower, $upper);
echo "Resultado: $result<br>\n";

function gauss($lower, $upper){
    if($upper >= $lower && $lower >= 0){
        return ($upper*($upper+1)/2) - ($lower*($lower+1)/2);
    }
}
?>
```

### Listagem 3. Exemplo de passagem de array para funções

```
<?php
$values = array(4, 3, 19, 23, 15);
$result = geomean($values);
echo "Resultado: $result<br>\n";

function geomean($invalues){
    foreach ($invalues as $val){
        $sum += $val*$val;
    }
    return sqrt($sum);
}
?>
```

### Listagem 4. Retornado mais de um valor na função

```
<?php
$values = array(4, 3, 19, 23, 15);
$result = compute($values);
foreach($result as $val){
    echo "valor: $val<br>\n";
}

function compute($invalues){
    $result = array();
    $sum = 1;
    foreach($invalues as $val){
        $sum += $val*$sum;
        array_push($result, $sum);
    }
    return $result;
}
?>
```

### Listagem 5. Exemplo simples de tratamento de exceções

```
<?php
$a = "uma string";
if(sqrt($a)){
    echo "Resultado: $b";
}else{
    echo "A operação não pode ser executada.";
}
?>
```

### Listagem 6. Exemplo de tratamento apenas com If

```
<?php
$a = 3;
if($a < 5){
    echo '$a é menor que 5';
    exit;
}
echo "Isso não pode ser executado.";
?>
```

realizar o tratamento de exceções, incluímos um exemplo simples. **Listagem 5**

Em primeiro lugar, é definida uma *string*. Então, o script tenta computar a raiz quadrada da *string*, o que não será possível. Por isso, a condição avaliada pela cláusula *if* será *false* e o ramo da cláusula *else* será executado.

Se qualquer coisa não permitida acontecer durante a execução do script, é necessário interromper a execução do programa. Isto pode facilmente ser realizado com a ajuda da função *die* (**Listagem 6**).

Como podemos ver, a variável *\$a* (3) é menor do que 5, portanto a condição é verdadeira e o script pára mostrando a mensagem a seguir:

```
$a é menor que 5
```

A saída do comando *echo* no fim do script não será mais exibida. O comando *exit* não é o único fornecido pelo PHP para interromper um script. Se uma mensagem tiver que ser exibida antes de interromper um script, o comando *die*, como mencionado anteriormente, pode ser utilizado em lugar do comando *exit* como podemos ver a seguir:

```
<?php
$a = 3;
array_push($a, 4) or die ('$a não é um array');
echo "Isso não será executado";
?>
```

Uma mensagem de erro PHP é exibida na tela. Além do mais, a mensagem que foi passada para o comando *die* também será exibida.

```
$a não é um array
```

Para aplicações do mundo real, a mensagem incorreta gerada pelo PHP não será conveniente, pois faz com que o usuário pense que ocorreu um erro na aplicação. Para garantir que nenhum erro seja exibido pelo PHP, podemos utilizar o seguinte:

```
<?php
$a = 3;
@array_push($a, 4) or die ('$a não é um array');
echo "Isso não pode ser executado.";
?>
```

Se executarmos o script, veremos que nenhum erro PHP gerado pela linguagem é mostrado, isso porque incluímos o caractere arroba (@) antes da função *array\_push()* do PHP fazendo com que apenas o erro especificado por nós seja exibido:

```
$a não é um array
```

Como podemos ver, o PHP fornece um sistema flexível para tratamento de exceções, que pode ser facilmente utilizado para interceptar quase todos os erros que ocorrem durante a execução de um *script*.

## Trabalhando com arquivos

Trabalhar com arquivos em PHP é fácil. Os arquivos não têm que estar localizados na máquina local para serem acessíveis. É possível também trabalhar com arquivos remotos. Nesta seção, aprenderemos a trabalhar eficientemente com arquivos locais e remotos.

## Realizando operações básicas com arquivos locais

O trabalho com arquivos é um assunto importante em toda linguagem de programação. O mesmo aplica-se ao PHP. As funções para acessar o sistema de arquivos, permitem que o desenvolvedor construa aplicações altamente sofisticadas. Com a ajuda de arquivos, é possível armazenar informações a respeito do que está acontecendo em nossa aplicação ou, então, podemos acessar fontes de dados externas. Atualmente, muita informação é fornecida em bancos de dados, mas os arquivos são ainda um componente fundamental para as aplicações. Esta seção irá guiar-nos pelas funções do PHP relacionadas a arquivos e sistemas de arquivos. A primeira coisa que devemos saber a respeito de arquivos, é como abri-los e fechá-los como podemos ver a seguir:

```
<?php
$handle = @fopen("data.file", "r") or
die ("Não foi possível abrir o arquivo.");
echo "Arquivo aberto com sucesso.<br>";
fclose($handle);
?>
```

No exemplo anterior são utilizadas as funções *fopen* e *fclose*. Estas funções estão disponíveis em muitas linguagens de programação e foram emprestadas do C. No script, um arquivo chamado *data.file* é aberto para leitura. Isto é feito com a função *fopen*. O primeiro parâmetro define o nome do arquivo que deve ser aberto. O segundo parâmetro diz ao PHP em que modo o arquivo deve ser aberto. O PHP suporta os seguintes modos:

- *r* – Leitura somente;
- *r+* – Abre para leitura e gravação;
- *w* – Abre o arquivo para gravação e cria um arquivo vazio;

- *w+* – Abre o arquivo para leitura e gravação;

- *a* – Abre o arquivo para gravação e posiciona o ponteiro no final do mesmo;

- *a+* – Abre o arquivo para leitura e gravação e posiciona o ponteiro no final do mesmo;

A função *fclose* fecha o arquivo. Vejamos o conteúdo do arquivo que o script processou e imprimiu em tela:

```
Guinther::Brasil::Editor Geral
Adriano::Brasil::Editor Técnico
Gladstone::Brasil::Diretor
João Moraes::Brasil::Leitor
```

Às vezes é necessário descobrir alguma coisa a mais sobre um arquivo. Por isso, o PHP provê um comando chamado *stat*. A **Listagem 7** mostra o uso da função mencionada.

Ao executarmos o script, muita informação é exibida em tela como mostrado a seguir:

```
dispositivo: 773
inode (nó): 470215
inode (nó) de modo de proteção: 33188
número de links: 1
id do usuário proprietário: 500
grupo do usuário proprietário: 500
tipo de dispositivo: 2817
tamanho em bytes: 107
último acesso: 1003663763
última modificação: 1003663300
última mudança: 1003663300
tamanho do bloco de sistema de arquivos: 4096
número de blocos alocados: 8
```

A saída exibida pelo comando *stat* contém a mesma informação que é retornada pela função *stat* do C na qual o PHP se baseou. Para demonstrar que as funções do PHP para trabalhar com sistemas de arquivos, foram baseadas nas funções correspondentes do C, incluímos a estrutura retornada pela função *stat* do C na **Listagem 8**.

Como podemos ver, o conteúdo do resultado gerado pelo PHP é quase igual ao resultado gerado pelo C. Isto mostra claramente que o PHP e o C estão fortemente relacionados. O conhecimento disto facilitará o entendimento do comportamento e das funções do interpretador do PHP.

## Lendo e escrevendo em arquivos

Até agora, aprendemos a abrir e fechar arquivos. Também vimos como descobrir informações a respeito de um arquivo, mas ainda não lemos nem escrevemos dados. Começemos com um exemplo onde podemos ver como os dados são lidos em um arquivo (**Listagem 9**).





exemplo, o *array* é processado linha a linha e a string na linha é dividida após cada conjunto de caracteres ":". Desta maneira, um *array* chamado *\$val* é inicializado sempre que a repetição é processada.

Se quisermos exibir o conteúdo do arquivo na tela, sem processar previamente os dados, podemos utilizar a função *readfile*. A função *readfile(nome\_do\_arquivo)* abre um arquivo e exibe o conteúdo diretamente na tela. Isto é muito útil quando o arquivo já contém o código HTML.

## Realizando operações básicas com arquivos remotos

Na seção anterior, vimos como os arquivos locais são tratados com PHP. Os arquivos remotos são processados de um modo semelhante. Observemos o exemplo que acabamos de ver, mas desta vez os dados vêm de um serviço de hospedagem, ou seja, de um servidor remoto. Na **Listagem 12** podemos ver que há poucas mudanças na leitura do arquivo em relação ao exemplo da seção anterior.

Como podemos verificar, a única modificação que deve ser feita, consiste em que a posição do arquivo é definida agora por uma *URL*. Se tivermos as permissões para ler o arquivo, este poderá ser processado como qualquer outro arquivo. Sempre que o *script* é processado, contudo, o servidor *Web* acrescentará uma linha ao *logfile* (arquivo de log):

```
212.186.25.254 - - [21/Oct/2001:16:30:02
+0200] "GET /test/data.file HTTP/1.0" 200 107
"- " "PHP/4.0.4pl1"
```

### Listagem 12. Leitura do arquivo remotamente

```
<?php
$file = "http://212.186.25.254:/test/data.file";
$data = file($file);
echo "<table border='1'><tr>";
echo "<th>name</th><th>location</th><th>profession</th></tr>";
foreach($data as $line){
    $val = explode(":", $line);
    echo "<tr><td>$val[0]</td><td>$val[1]</td><td>
    $val[2]</td></tr>";
}
echo "</table>";
?>
```

### Listagem 13. Leitura remota do arquivo

```
<?php
$file = "http://212.186.25.254:/test/data.file";
$handle = fopen($file, "a+") or
die ("Não é possível ler o arquivo $file");
fputs($handle, "Patrick:Brasil::Presidente") or
die ("Não é possível escrever no arquivo");
fclose($handle) or
die ("Não é possível fechar o arquivo.");
echo "Funcionou";
?>
```

Como já mencionamos, o trabalho com arquivos remotos funciona da mesma forma como com arquivos locais. Contudo, alguns aspectos importantes têm de ser levados em consideração (**Listagem 13**).

Imaginamos que o *script* acrescentará os dados ao arquivo remoto, mas isto não acontece:

```
[guinther@devmedia.com.br teste]$ categoria
data.file
Guinther::Brasil::Editor Geral
Adriano::Brasil::Editor Técnico
Gladstone::Brasil::Diretor
João Morais::Brasil::Leitor
```

Nada foi acrescentado ao arquivo remoto, embora a saída do script tenha sido "funcionou" e nenhum erro de execução tenha ocorrido. Isso deve ser levado em consideração ao trabalharmos com arquivos remotos, caso contrário ficaríamos intrigados pelo fato da aplicação não retornar nenhum erro e apesar disso, não produzir nenhuma saída.

## Funções adicionais do sistema de arquivos

Como já vimos, o PHP oferece funções para trabalhar com arquivos locais e remotos. Até agora, vimos como executar todas as operações básicas. Nesta seção, veremos que o PHP fornece muito mais funções do que as já vistas:

- *string basename (string path [, string suffix])* – Se passarmos um nome de arquivo incluindo o caminho (*path*) para a função *basename*, será retornado o nome do arquivo sem o caminho;

- *int chgrp (string filename, mixed group)* – *chgrp* pode ser utilizada para alterar o grupo ao qual um arquivo pertence;
- *int chmod (string filename, int mode)* – *chmod* pode ser utilizada para alterar o modo de um arquivo. Esta função não funciona com servidores Windows;
- *int chown (string filename, mixed user)* – *chown* pode ser utilizada para alterar o proprietário de um arquivo;
- *void clearstatcache (void)* – limpa o *cache* de estado do arquivo (*file stat cache*). Normalmente o resultado de *stat* é armazenado em *cache* para economizar o tempo necessário para recuperar as informações de arquivos. Com o auxílio do *clearstatcache*, podemos forçar o PHP a re-executar a função de verificação de status;
- *int copy (string source, string dest)* – Utilizada para copiar arquivos.
- *string dirname (string path)* – *dirname* é a contrapartida da função *basename*. Retorna o caminho de um arquivo;
- *float diskfreespace (string directory)* – Utilizada para obter o espaço vazio disponível no diretório atual;
- *float disk\_total\_space (string directory)* – Retorna a capacidade total de armazenamento disponível na partição do diretório passado para a função;
- *bool fclose (int fp)* – Fecha um arquivo aberto;
- *int feof (int fp)* – Verifica se o ponteiro está posicionado no EOF (*end of file* – fim do arquivo);
- *int fflush (int fp)* – Descarrega em disco o conteúdo do *buffer*;
- *string fgets (int fp)* – Lê um único caractere do arquivo;
- *array fgetcsv (int fp, int length [,string delimiter])* – Lê valores separados por vírgula (CSV – *comma separated values* – valores separados por ponto e vírgula) e retorna um *array* de valores. O delimitador de linha pode opcionalmente ser passado para a função;
- *string fgets (int fp, int length)* – Utilizado para ler do arquivo;
- *string fgetss (int fp, int length [,string allowable\_tags])* – Equivalente ao *fgets*, porém, tenta extrair *tags* HTML;
- *array file (string filename [, int use\_include\_path])* – Retorna o conteúdo de um arquivo em um *array*;
- *bool file\_exists (string filename)* – Retorna *True* caso o arquivo passado exista;

- *int filetime (string filename)* – Recupera a data e hora de acesso de um arquivo;
- *int filemtime (string filename)* – Retorna a data e hora de modificação do número inode (nó);
- *int filegroup (string filename)* – Retorna o grupo ao qual o arquivo pertence;
- *int fileinode (string filename)* – Recupera o número inode do arquivo;
- *int filemtime (string filename)* – Retorna a data e hora da modificação;
- *int fileowner (string filename)* – Retorna o proprietário do arquivo;
- *int fileperms (string filename)* – Retorna as permissões para o arquivo;
- *int filesize (string filename)* – Retorna o tamanho do arquivo;
- *string filetype (string filename)* – Retorna o tipo do arquivo (*fifo*, *char*, *dir*, *block*, *link*, *file* ou *unknown*);
- *bool flock (int fp, int operation [, int wouldblock])* – Utilizado para bloquear arquivos;
- *int fopen (string filename, string mode [, int use\_include\_path])* – Abre um arquivo ou uma URL e retorna um handle para o arquivo;
- *int fpassthru (int fp)* – Escreve todos os dados do arquivo a partir da posição atual do ponteiro até o final do arquivo na saída padrão;
- *int fputs (int fp, string str [, int length])* – Escreve os dados para um arquivo, iniciando pela posição atual do ponteiro do arquivo;
- *string fread (int fp, int length)* – Leitura de arquivo binário-segura (Binary-safe);
- *mixed fscanf (int handle, string format [,string var1...])* – Interpreta a entrada com base em um formato;
- *int fseek (int fp, int offset [, int whence])* – Utilizado para mover o ponteiro do arquivo dentro do mesmo. Pode ser: *SEEK\_SET* (posicionar o ponteiro do arquivo em *offset bytes* de deslocamento a partir do início); *SEEK\_CUR* (posicionar o ponteiro do arquivo em *offset bytes* de deslocamento a partir da posição atual do mesmo); *SEEK\_END* (posicionar o

- ponteiro do arquivo para fim do arquivo mais *offset bytes* de deslocamento);
- *array fstat (int fp)* – Recupera informações sobre o arquivo;
- *int ftell (int fp)* – Utilizado para descobrir em que posição o ponteiro de arquivo pode ser encontrado;
- *int ftruncate (int fp, int size)* – Utilizado para truncar o arquivo para um dado comprimento;
- *int fwrite (int fp, string string, int [length])* – Escrita no arquivo binário-segura (Binary-safe);
- *int set\_file\_buffer (int fp, int buffer)* – Modifica o buffer de arquivo para um dado tamanho. Por padrão, este tamanho é de 8 Kbytes;
- *bool is\_dir (string filename)* – Utilizado para descobrir se filename é um diretório;
- *bool is\_executable (string filename)* – Verifica se um dado arquivo é executável;
- *bool is\_file (string filename)* – Verifica se filename é um arquivo;
- *bool is\_link (string filename)* – Verifica se filename é um link simbólico;
- *bool is\_readable (string filename)* – Verifica se o arquivo pode ser lido;
- *bool is\_writable (string filename)* e *bool is\_writeable (string filename)* – Verifica se o arquivo pode ser gravado;
- *bool is\_uploaded\_file (string filename)* – Retorna True se o arquivo foi carregado utilizando *HTTP POST*;
- *int link (string target, string link)* – Cria um link;
- *int linkinfo (string path)* – Retorna informações sobre um link;
- *int mkdir (string pathname, int mode)* – Cria um novo diretório;
- *bool move\_uploaded\_file (string filename, string destination)* – Utilizado para mudar a localização de um arquivo carregado;
- *array pathinfo (string path)* – Retorna informações sobre o caminho em um *array*;
- *int pclose (int fp)* – Fecha um *pipe* aberto por um *popen*;
- *int popen (string command, string mode)* – Abre um *pipe*;
- *int readfile (string filename, int [use\_in-*

- clude\_path])* – Lê dados a partir de um arquivo e imprime o conteúdo na saída padrão;
- *int rename (string oldname, string newname)* – Renomeia um arquivo;
- *int rewind (int fp)* – Posiciona o ponteiro no início do arquivo;
- *int rmdir (sequência dirname)* – Remove o diretório;
- *array stat (string filename)* – Recupera informações detalhadas sobre o arquivo;
- *array lstat (string filename)* – Retorna informações sobre um link;
- *int symlink (string target, string link)* – Gera um link simbólico para um arquivo;
- *string tempnam (string dir, string prefix)* – Cria um filename temporário único;
- *int tmpfile (void)* – Cria um arquivo temporário com um filename único;
- *int touch (string filename, int [time])* – Ajusta a data e hora da modificação do arquivo para a data e hora atual;
- *int unlink (string filename)* – Apaga um arquivo;

Como podemos ver, o PHP fornece uma lista enorme de funções embutidas. A maior parte das funções funciona com todos os sistemas de arquivos comuns. Principalmente em sistemas Unix, o PHP demonstra um tremendo poder, pois algumas funções só são utilizadas em Unix.

## Conclusão

Nessa terceira parte de introdução ao PHP vimos como é importante saber trabalhar com funções de modo que podemos armazenar nossas principais regras de negócios em rotinas centralizadas. Dessa forma fica muito mais simples fazer a manutenção do sistema e desenvolver novas soluções sempre que necessário.

Vimos também como fazer o tratamento de exceções, premissa básica de qualquer linguagem de programação. Por últimos aprendemos como trabalhar e quais as diferenças entre utilizar arquivos localmente e remotamente. ●

