Xdebug

Fazer o debug de aplicações PHP sempre foi um grande problema. Utilizar echo e var_dump não é a melhor maneira, além de não ser nada elegante. Existe uma forma de realizar o debug de aplicações PHP de forma mais correta e útil. Esta forma é usando-se a extensão chamada xdebug.

Instalação

Pode-se instalar o xdebug compilando-se o código-fonte ou usando uma versão binária. Para Windows o mais fácil é fazer o download de um arquivo dll no site

http://pecl4win.php.net/ext.php/php_xdebug.dll

e salvá-lo no diretório ext do PHP.

Para Linux é possível fazer a instalação usando-se o comando pecl:

sudo pecl install xdebug-beta

Com este comando é feito o download dos fonte e o xdebug é compilado e instalado.
Depois de instalado é preciso configurar o PHP para reconhecer a nova extensão. No arquivo php.ini é preciso adicionar a seguinte linha:

Para windows:
```
zend_extension_ts="d:/programas/xampp/php/ext/php_xdebug.dll"
```

Para Linux:
```
zend_extension="/usr/lib/php5/xdebug.so"
```

Após adicionar uma das linhas acima, que indica o caminho completo do arquivo da extensão, é preciso configurar o Xdebug, adicionando as linhas:

```
xdebug.default_enable = On
xdebug.collect_includes = On
xdebug.collect_params = On
xdebug.collect_return = On
xdebug.collect_vars = On
xdebug.dump_globals = On
```

OBS: Sempre que for modificado o arquivo php.ini é necessário reiniciar o servidor Apache.

Usando o Xdebug

O primeiro teste é criar um script PHP com erro e executá-lo. O Xdebug vai mostrar diversas informações importantes sobre o erro.
O Xdebug adiciona várias opções ao php. Uma delas é refazer o var_dump. Quando a extensão está ativa, o comando var_dump mostra informações mais detalhadas das variáveis. Para testar é só executar o var_dump passando como parâmetro qualquer variável ou objeto do script:

```
$dados = array(
   'um' => 'uma string',
   'dois' => array(
      'two.one' => array(
         'two.one.zero' => 210,
         'two.one.one' => array(
            'two.one.one.zero' => 3.141592564,
            'two.one.one.one'  => 2.7,
         ),
      ),
   ),
   'tres' => $t,
   'quatro' => range(0, 5),
);
var_dump( $dados );
```

Proteção de memória

É possível configurar o Xdebug para proteger o servidor em caso de um loop infinito, que poderia consumir muitos recursos. Por default esta o Xdebug configura para 100 iterações. Para modificar esta características deve-se adicionar a seguinte linha no php.ini:

xdebug.max_nesting_level=4

Assim, o programa abaixo vai invocar a função a() somente quatro vezes e depois irá gerar um erro:

```
<?php
a();
function a() {
      echo "lixo";
      a();
}
?>
```

(!) Fatal error: Maximum function nesting level of '4' reached, aborting! in /home/www/aula/teste/index.php on line *9*

Call Stack

#	Time	Memory	Function	Location
1	0.0005	58964	{main}( )	../index.php:0
2	0.0005	59108	a( )	../index.php:6
3	0.0005	59156	a( )	../index.php:9

Coletando mais informações

Para que o Xdebug possa coletar mais informações sobre os erros é possível configurá-lo para capturar as variáveis globais como POST, GET, FILES, etc. Para isso é só adicionar as linhas abaixo no php.ini:

```
xdebug.dump.GET=*
xdebug.dump.POST=*
xdebug.dump.COOKIE=*
xdebug.dump.ENV=*
xdebug.dump.FILES=*
xdebug.dump.REQUEST=*
xdebug.dump.SERVER=*
xdebug.dump.SESSION=*
```

O * pode ser substituído pelo nome da variável que se deseja capturar. Exemplo:

```
xdebug.dump.POST=username, password
```

Quantas funções foram invocadas no script?

```
<?php
echo xdebug_get_function_count();
?>
```

Quanto tempo demorou a execução?

```
<?php
for($i=0;$i<1000000;$i++) {
	$j = $i;
}
echo "Tempo de execução=". xdebug_time_index();
?>
```

Quanta memória utilizou?

```
<?php
echo "Memória: ", xdebug_memory_usage(), " bytes\n\n";
for($i=0;$i<1000000;$i++) {
	$j = $i;
}
echo "Máximo de Memória usada: ", xdebug_peak_memory_usage(), " bytes\n";
?>
```

Trace da aplicação

Para acompanhar o fluxo da aplicação é preciso adicionar as linhas abaixo no php.ini:

```
xdebug.auto_trace=1
xdebug.trace_output_dir=/tmp
xdebug.trace_options=1
xdebug.trace_format = 0
```

Desta forma será criado um arquivo no /tmp chamado trace.??? para cada script PHP que for executado. Neste arquivo estará descrito cada comando executado, variáveis criadas, etc. Assim facilita a visualização do fluxo do script e eventuais erros.
Outra forma de se fazer isso é não adicionar as linhas acima no php.ini e colocar explicitamente no código do script as funções xdebug_start_trace() e xdebug_stop_trace() indicando onde deve ser iniciado e finalizado o trace. Exemplo:

```
<?php
echo "Memória: ", xdebug_memory_usage(), " bytes\n\n";
xdebug_start_trace('BUG_TRACE_APPEND'); //fazer append do trace
for($i=0;$i<1000000;$i++) {
	$j = $i;
}
for($j=0;$j<100;$j++) {
	echo $j;
}
xdebug_stop_trace();
echo "Máximo de Memória usada: ", xdebug_peak_memory_usage(), " bytes\n";
?>
```

Desta forma, a função xdebug_peak_memory_usage() não será apresentada no trace. Será criado um arquivo no diretório temporário do servidor com o nome trace.??.xt

Profiling

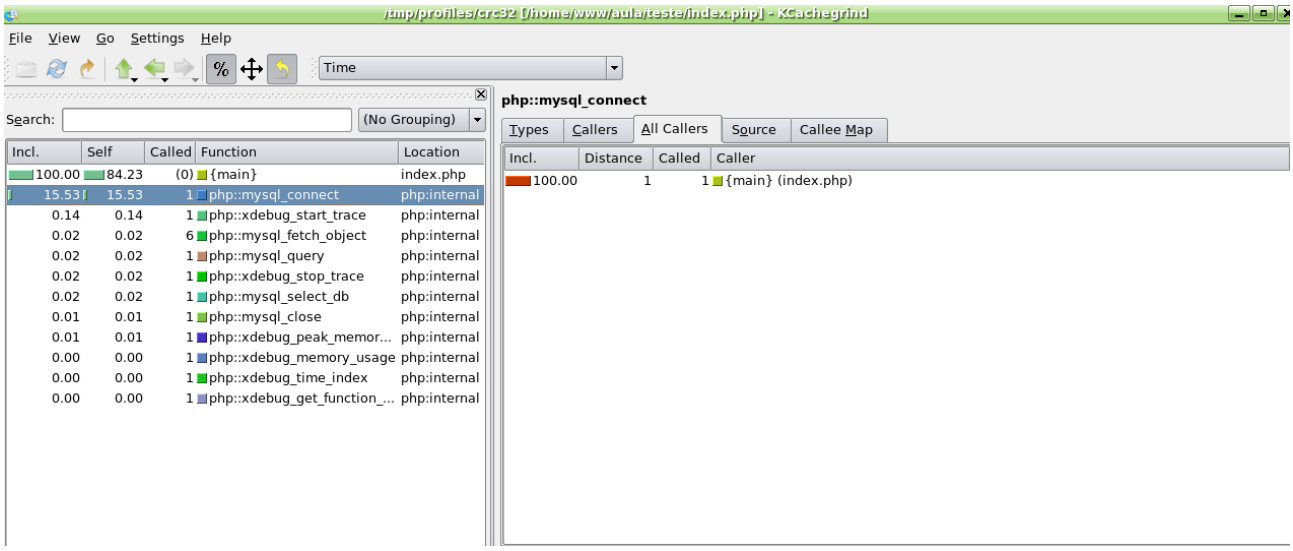
Profiling é uma maneira de visualizar quais partes do código do script estão demorando mais para executar. O Xdebug precisa ser configurado para poder gerar estas estatísticas. Para isso deve-se adicionar as linhas no php.ini:

```
xdebug.profiler_enable = 1
xdebug.extended_info = 0
xdebug.remote_enable = 0
xdebug.profiler_output_dir=/tmp
```

Após a execução de um script é gerado um arquivo no /tmp com as estatísticas de execução. Para analisarmos estas estatísticas pode-se usar uma ferramenta gráfica para Linux chamada kcachegrind, que pode ser instalada no Ubuntu com o comando

```
sudo apt-get install kcachegrind
```

Executando-se o kcachegrind e abrindo o arquivo gerado podemos visualizar qual parte do script demorou mais para ser executada, conforma a figura abaixo demonstra:



Existe uma ferramenta para Windows que pode ser encontrada em
http://sourceforge.net/projects/wincachegrind/
que tem a mesma função.

Debug de Aplicações

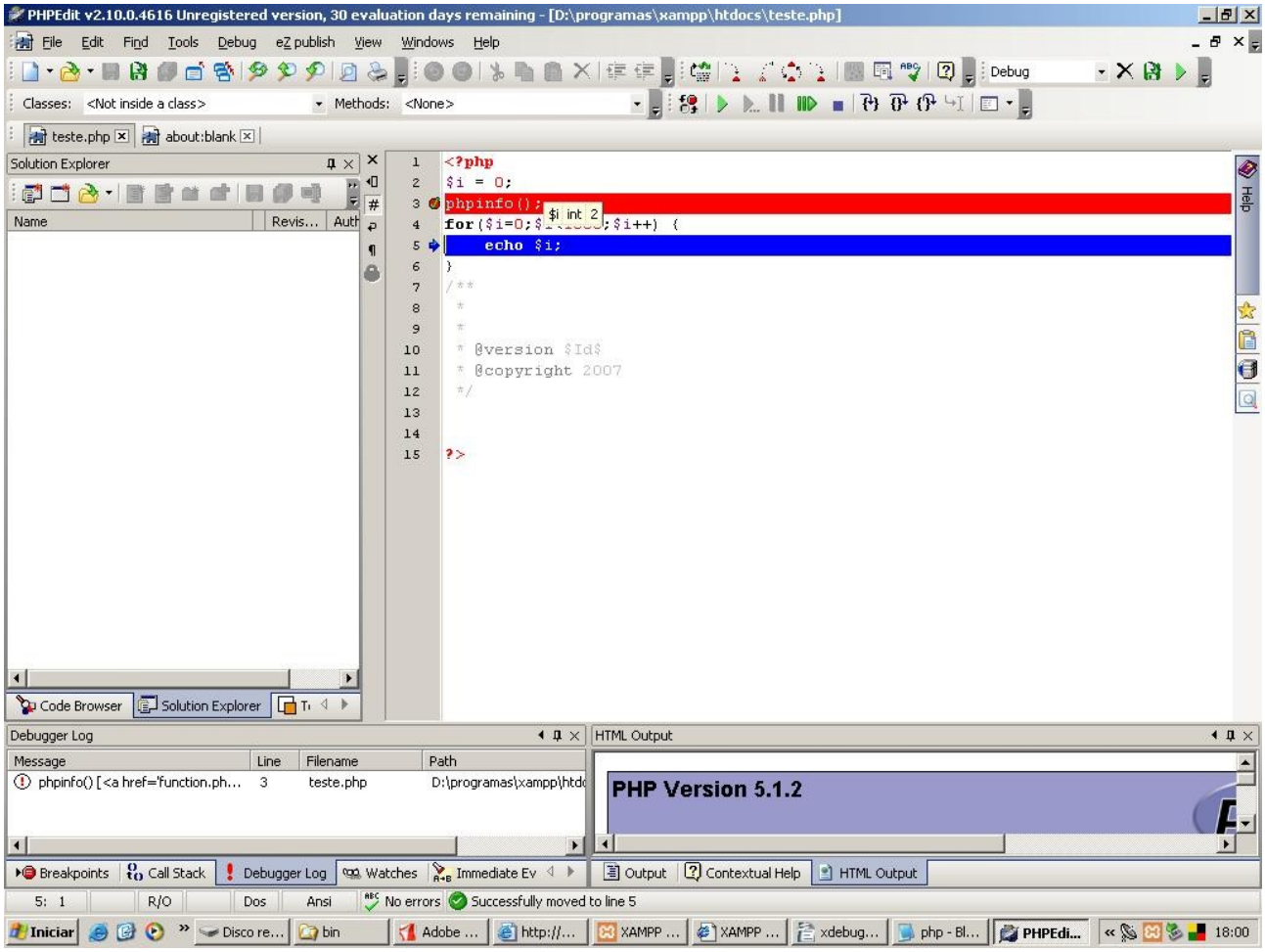
Outra característica que é importante para auxiliar o processo de desenvolvimento é o acesso a um debug da aplicação, com inspeção de variáveis, pontos de parada, etc. O Xdebug fornece acesso a este tipo de característica. Para isso é preciso ter alguma IDE que possua suporte ao Xdebug. Dentre as IDEs que se encaixam nesta categoria podem-se citar o Eclipse, vim, e o PHPEdit versão 2.10 ou superior.

Para que o Xdebug forneça acesso ao debug das aplicações é preciso adicionar as linhas abaixo no php.ini:

```
xdebug.remote_enable=1
xdebug.remote_handler=dbgp
xdebug.remote_mode=req
xdebug.remote_host=localhost
xdebug.remote_port=9000
xdebug.extended_info=1
```

A imagem abaixo demonstra o uso do debug no PHPEdit. A linha em vermelho indica um ponto de

parada (break point) do programa. A linha em azul é a linha em execução no momento.
Desta forma é possível acompanhar passo-a-passo a execução de uma aplicação, ver o conteúdo das variáveis e faciliar a descoberta de quaisquer problemas no script.



No site do Xdebug é possível encontrar outras IDEs que podem ser usadas para fazer o debug das aplicações usando a extensão.

Referências

http://files.derickrethans.nl/**xdebug-brasil6.pdf**.
http://www.derickrethans.nl/files/phparch-xdebug-qa.pdf
http://www.xdebug.org