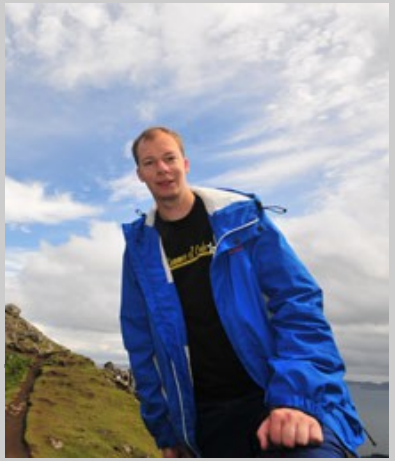
Profiling PHP Applications

PHPCon Poland - Masłów k/Kielc, Poland - Oct 22th, 2011

Derick Rethans - derick@derickrethans.nl - twitter: @derickr

About Me



Derick Rethans

- Dutchman living in London
- PHP development
- Freelancer doing PHP internals develop
  ie.: writing extensions for a living
- Anderskor's Camouflage
- Author of Xdebug
- Author of the mcrypt, input_filter, dbus, translit and date/time extensions
- Contributor to the Apache Zeta Components Incubator project (formerly eZ Components)

The Internet is Full of Nonsense

A small selection of PHP optimisation tips I found on the internet

echo is faster than print

Use sprintf instead of variables contained in double quotes, it’s about 10x faster.

Use <?php … ?> tags when declaring PHP as all other styles are depreciated, including short tags.

The Internet is Full of Nonsense

I can only say one thing:

Do I need to optimise my code?

Before you can optimise anything:

- Find out if things are running slow
- Find out whether it is the code
- Understand your code, application and execution paths
- Find out which parts of the code are slow
- Find out what you can optimise

How does the application perform?

Benchmark the application: siege

- check ~/.siegerc and set the logfile setting
- Create a file with urls:http://derickrethans.nl/ http://derickrethans.nl/spatial-indexes-data-sqlite.html http://derickrethans.nl/who.html
- run against your code: siege -c 4 -r 10 -f /tmp/urls.txt

```
Transactions:                     40 hits
Availability:                 100.00 %
Elapsed time:                   9.76 secs
Data transferred:               1.34 MB
Response time:                  0.18 secs
Transaction rate:               4.10 trans/sec
Throughput:                   0.14 MB/sec
Concurrency:                   0.73
Successful transactions:             40
Failed transactions:                  0
Longest transaction:               0.39
Shortest transaction:              0.08
```

Is it my code that is slow?

Multiple possible reasons:

- The database is slow
- There is lots of IO
- Your code is slow
- The system is busy with other things

$ vmstat 1

```
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs *|dd1111|us sy|* id wa
5  0 121248 367520 767824 3080668    0    0 11272     0 1593 5540 *|dd1111|74 22|*  3  0
5  0 121248 290784 767960 3082980    0    0   268     0 1555 5381 *|dd1111|77 20|*  3  0
5  0 121248 238340 768132 3084336    0    0  1364 21160 2263 6815 *|dd1111|70 21|*  3  7
6  0 121248 170772 768300 3087100    0    0  1652     0 1802 8540 *|dd1111|71 25|*  4  0
```

CPU is (close to) fully in use: 74 + 22 ➔ your code is slow.

$ vmstat 1

```
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id *|dd1111|wa|*
0  1 121248  50776 314796 4156000    0    0 245800     0 3018 5394  1 11 71 *|dd1111|17|*
1  0 121248  52112 315108 4236692    0    0 243672     0 3107 5552  1 10 71 *|dd1111|18|*
1  1 121248  50148 315256 4318268    0    0 243096    44 3146 5463  1 11 72 *|dd1111|17|*
1  1 121248  52120 315484 4396356    0    0 243784     0 3006 5419  1 10 72 *|dd1111|18|*
```

wait is > 10 ➔ IO is the bottleneck.

Understand your code, application and execution paths

In-code tools:

- Add timing points

External tools:

- Basic overview: inclued
- Deep details: xdebug

Timing Points

Code with analysis in mind:

- Add timing points around specific events
- Check changes over time

eZ Publish:

Symfony:

Zeta Components:

```
<?php
// Get the one and only instance of the ezcDebug.
$debug = ezcDebug::getInstance();
// Start the accumulator.
$debug->startTimer( "Program runtime", "Accumulators" );
$debug->switchTimer( "Start", "Program runtime" );
// The name of the timer is: "Hello world" and it will be
// placed in the group: "output".
$debug->startTimer ("Hello world", "output" );
echo "Hello world<br/>";
$debug->stopTimer( "Hello world" );
// Replace the "Start" timer for "Half the way".
$debug->switchTimer( "Half the way", "Start" );
// Measure the time of writing "cruel world".
$debug->startTimer( "Goodbye cruel world", "output" );
echo "Goodbye cruel world<br/>";
$debug->stopTimer( "Goodbye cruel world" );
// Stop the last timer.
$debug->switchTimer( "Stop", "Half the way" );
$debug->stopTimer( "Stop" );
// Get HTML output.
$output = $debug->generateOutput();
?>
```

Time accumulators:

Accumulator	Elapsed	Percent	Count	Average
ini_load				
Load cache	0.0181 sec	7.6849%	10	0.0018 sec
FindInputFiles	0.0148 sec	6.2859%	10	0.0015 sec
Mysql Total				
Mysql_queries	0.0605 sec	25.7125%	5	0.0121 sec
Looping result	0.0007 sec	0.3089%	4	0.0002 sec
Template Total	0.0958 sec	40.8%	3	0.0319 sec
Template load	0.0178 sec	7.5636%	3	0.0059 sec
Template processing	0.0774 sec	32.9262%	3	0.0258 sec
override				
Cache load	0.0145 sec	6.1490%	3	0.0048 sec
Matching rules	0.0003 sec	0.1127%	1	0.0003 sec
Debug-Accumulator				
test	0.0295 sec	12.5591%	1	0.0295 sec

Total script time:0.2351 sec

Timers Sf 1.0.0 | vars & config | logs & msgs | 9 | 609 ms

type	calls	time (ms)
Configuration	14	60.42
Action "question/frontpage"	1	132.43
Database	9	7.56
View "Success" for "question/frontpage"	1	243.24
Partial "question/_question_list"	1	151.49
Partial "question/_question_block"	2	119.74
Partial "question/_interested_user"	2	12.99
Partial "moderator/_question_options"	2	2.80
Component "sidebar/default"	1	0.02
Partial "sidebar/_default"	1	25.62
Partial "tag/_tag_cloud"	1	2.09
Partial "question/_search"	1	0.97
Partial "sidebar/_rss_links"	1	3.08
Partial "sidebar/_moderation"	1	1.32
Partial "sidebar/_administration"	1	1.85

Timings	Elapsed	Percent	Count	Average
output				
Hello world	0.00003	59.39 %	1	0.00003
Goodbye cruel world	0.00002	40.61 %	1	0.00002
Total:	**0.00005**	**100.00 %**	**2**	**0.00002**
Accumulators				
Program runtime	0.00051	100.00 %	1	0.00051
Start	*0.00025*	*49.41 %*		
Half the way	*0.00038*	*75.11 %*		
Stop	*0.00049*	*96.85 %*		

Inclued

Dumps includes/classes hiearchies

Install:

```
pecl install inclued
```

Add to php.ini:

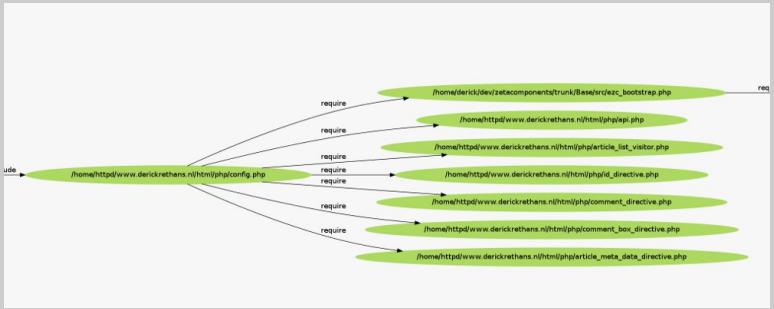
```
extension=inclued.so
inclued.enabled=1
inclued.dumpdir=/tmp
```

Generate graphs:

```
php -dinclued.enable=0 gengraph.php -t includes -i /tmp/inclued.22439.1
dot -Tpng -o inclued-includes.png inclued.out.dot
php -dinclued.enable=0 gengraph.php -t classes -i /tmp/inclued.22439.1
dot -Tpng -o inclued-classes.png inclued.out.dot
```

Include overview: inclued-includes.png:

Class hiearchy: inclued-classes.png:



I Do Need a Debugger

- Xdebug: An Open Source debugging tool
- About 8 years old
- Works on "every" operating system
- Version 2.1 released earlier this year
- PHP 5.1, 5.2, 5.3, 5.4 and trunk

Function trace

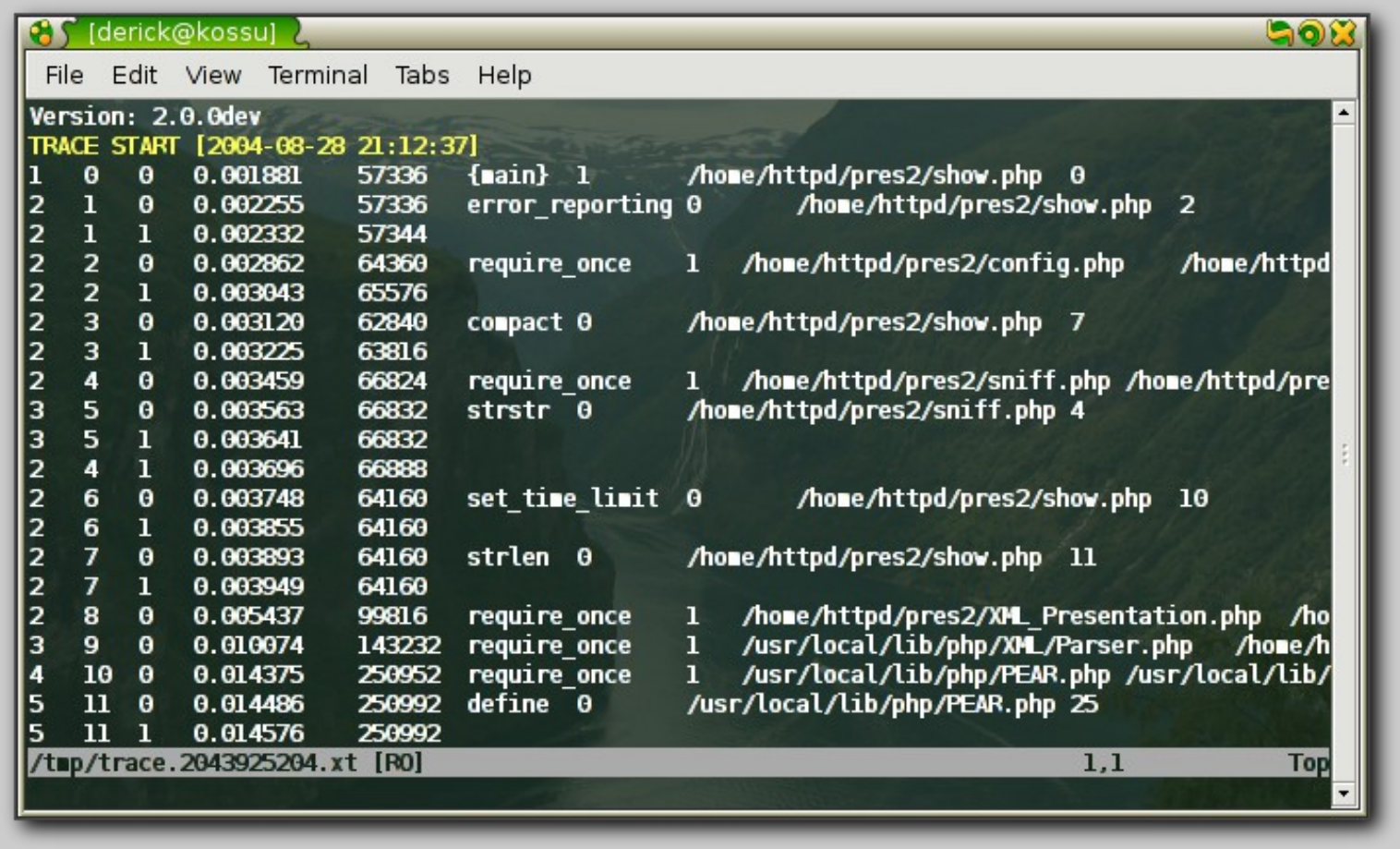
```
TRACE START [2010-03-24 11:03:12]
    0.0022     787776   -> {main}() /home/httpd/pres2/show2.php:0
    0.0028     803160     -> require(/home/derick/dev/ezcomponents/trunk/Base/src/
ezc_bootstrap.php)/home/httpd/pres2/show2.php:2
    0.0029     803552       -> dirname('/home/derick/dev/ezcomponents/trunk/Base/src/
ezc_bootstrap.php')/home/derick/dev/ezcomponents/trunk/Base/src/ezc_bootstrap.php:12
    0.0030     803968       -> explode('/', '/home/derick/dev/ezcomponents/trunk/Base/src')/home/
derick/dev/ezcomponents/trunk/Base/src/ezc_bootstrap.php:13
    0.0031     807352       -> count(array (0 => '', 1 => 'home', 2 => 'derick', 3 => 'dev', 4 =>
'ezcomponents', 5 => 'trunk', 6 => 'Base', 7 => 'src'))/home/derick/dev/ezcomponents/trunk/Base/src/
ezc_bootstrap.php:15
    0.0032     807800       -> array_slice(array (0 => '', 1 => 'home', 2 => 'derick', 3 => 'dev', 4
```

Some settings:

```
xdebug.auto_trace=1
xdebug.trace_output_dir=/tmp
xdebug.collect_params=1
xdebug.collect_return=1
xdebug.collect_includes=1
xdebug.collect_assignments=1
```

Function trace to file

Automatic readable format




```
xdebug.auto_trace=1      ; enable tracing
xdebug.trace_format=1    ; selects computerized format
xdebug.trace_options=0   ; sets extra option (1 = append)
```

Function trace

Other functionality

- HTML traces
- Tracing only parts of an application with xdebug_start_trace() and xdebug_stop_trace().
- Fetching the trace file name that is being used with xdebug_get_tracefile_name().
- Changing how much data is shown with xdebug.var_display_max_children, xdebug.var_display_max_data and xdebug.var_display_max_depth.

Trace file analysers

One bundled with Xdebug:

```
php ~/dev/php/xdebug/trunk/contrib/tracefile-analyser.php time-own 5
...
Showing the 5 most costly calls sorted by 'time-own'.

                                                                 Inclusive        Own
function                                                  #calls  time     memory  time     memory
-------------------------------------------------------------------------------------------------
array_pop                                                    715  1.0252  -139880  1.0252  -139880
preg_match                                                  2986  0.3718  1016336  0.3718  1016336
{main}                                                         1  6.4562  7335704  0.3476 -15198832
next                                                         432  0.2386        0  0.2386        0
count                                                       3302  0.2132        0  0.2123        0
```

ValaXdebugTools (http://tinyurl.com/vxdebugtools):

- trace4func: Give a function name, get a stack trace for each time it is called.
- evaled: Prints an ordered list of evals, together with what file is responsible for this evil.
- whocalls: Find out what files contain calls to the given function.
- whatisincluded: Find out what files are included and where they are included from.

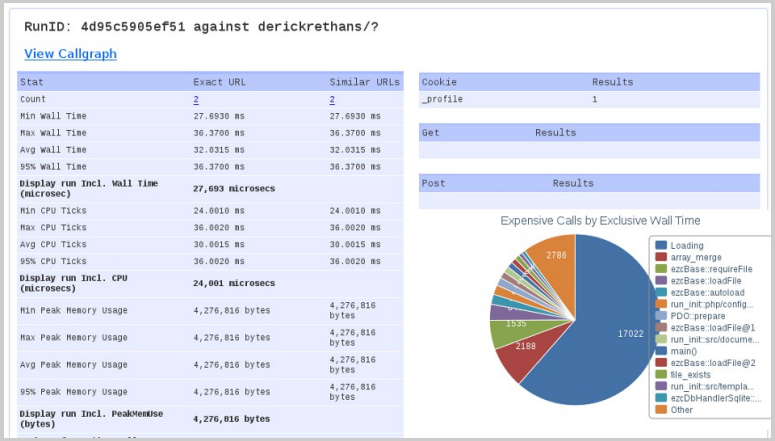
demo

Profiling

XHProf

Install:

```
pecl download xhprof-beta
tar -xvzf xhprof-0.9.2.tgz
cd xhprof-0.9.2/extension
phpize && ./configure && make install
```



In php.ini:

```
extension=xhprof.so
xhprof.output_dir=/tmp
```

Download XHGui:

```
$ cd /home/httpd
$ git clone https://github.com/preinheimer/xhprof
```

Config XHGui (xhprof/xhprof_lib/config.php:

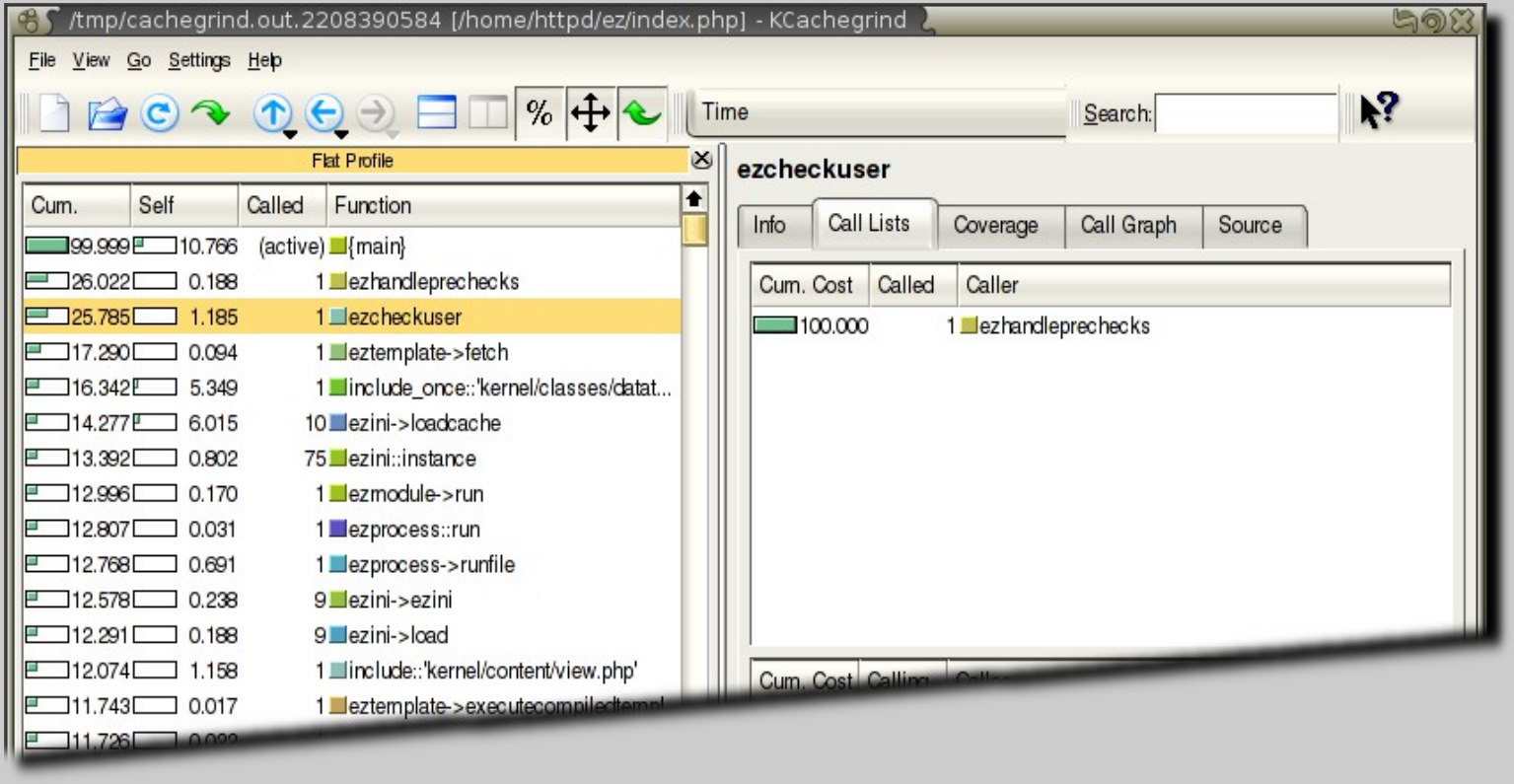
```
$_xhprof['dbhost'] = 'localhost';
$_xhprof['dbuser'] = 'root';
$_xhprof['dbpass'] = 'root';
$_xhprof['dbname'] = 'xhprof';
$_xhprof['servername'] = 'derickrethans';
$_xhprof['namespace'] = 'MySite';
$_xhprof['url'] = 'http://xhprof/';
```

Setup XHGui database:

```
mysqladmin -u root -p create xhprof
mysql -u root -p xhprof
CREATE TABLE `details` (
  `id` char(17) NOT NULL,
  `url` varchar(255) default NULL,
```

Profiling

KCacheGrind's Flat Profile and Call List

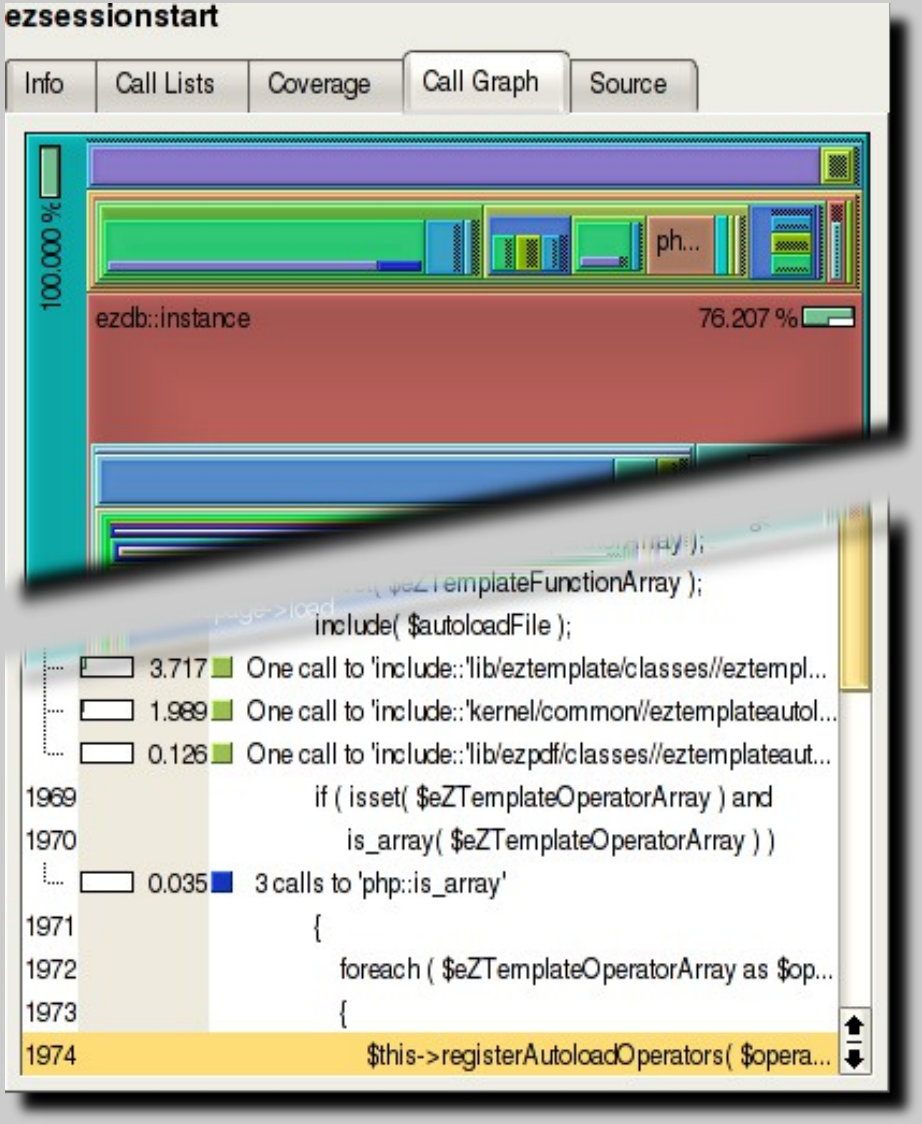



```
xdebug.profiler_enable=1          ; enable profiler
xdebug.profiler_output_dir=/tmp    ; output directory
xdebug.profiler_output_name=cachegrind.out.%p
```

Profiling

KCacheGrind's Call Graph and Source Annotations

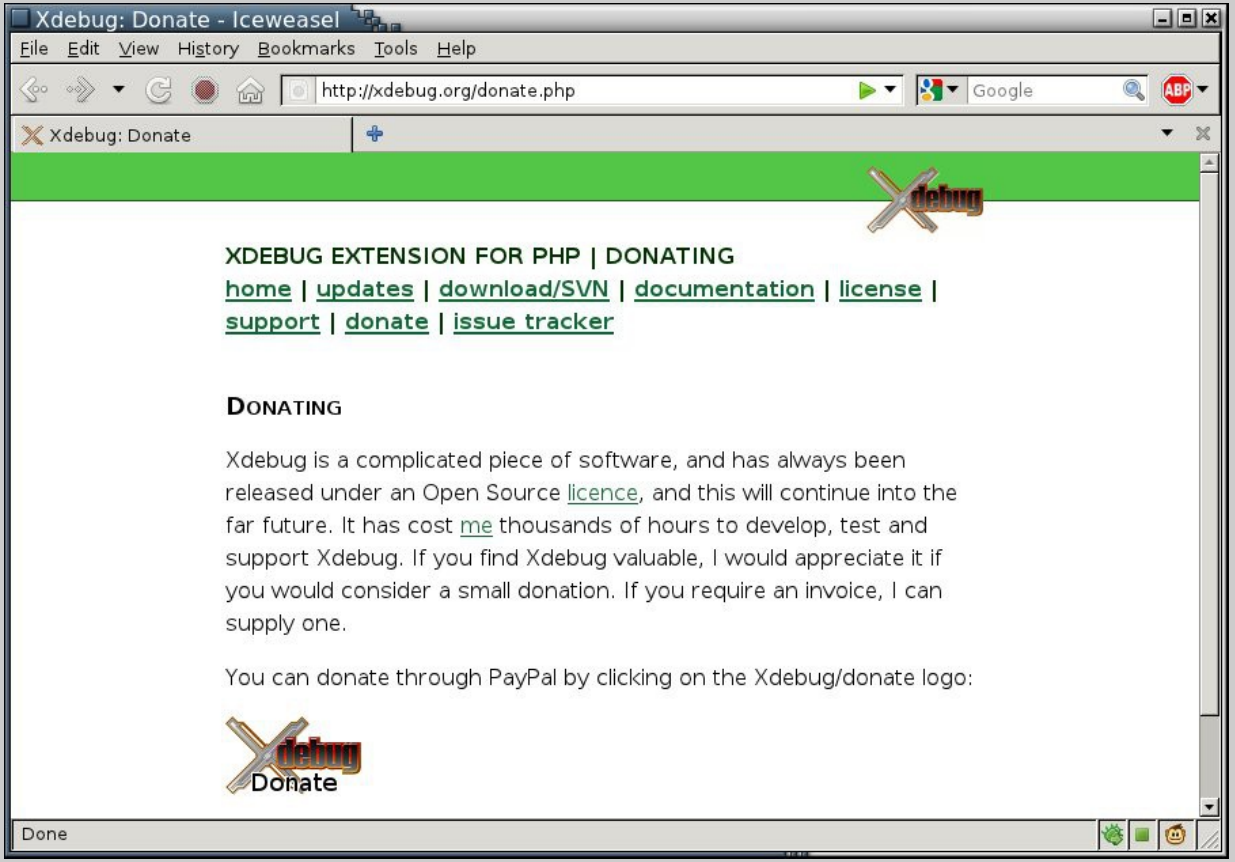
- Call graph
- Area shows time spend
- Stacked to show callees
- Source annotations
- Number of calls
- Total time per function



demo

Xdebug

- It's Open Source and free (as in "free beer")
- Working on Xdebug takes up a lot of spare time
- I don't have a lot of spare time



Resources

- Siege: http://www.joedog.org/index/siege-home
- Apache Zeta Components: http://incubator.apache.org/zetacomponents/
- Inclued: http://uk2.php.net/inclued
- ValaXdebugTools: http://tinyurl.com/vxdebugtools
- XHProf: http://mirror.facebook.net/facebook/xhprof/
- XHGui: https://github.com/preinheimer/xhprof
- Xdebug: http://xdebug.org
- If you like Xdebug: http://xdebug.org/donate.php
- These slides: :-:joindin:-: