

Pontificating on Perl Profiling

Lisa Hagemann
VP Engineering, Dyn Inc.
twitter.com/lhagemann
twitter.com/dyninc

What is Profiling?

- A way to evaluate what is your program doing.
- Commonly evaluates the behavior of the program, measuring frequency and duration of different call paths including the call paths.
- Used to evaluate areas ripe for optimization.

What is Benchmarking?

- Defining a measurement for comparison
- Benchmark time, memory, database calls
- Provides data, not answers

TMTOWTDI

- There's More Than One Way To Do It
- What's the Best way to do it?

use Benchmark;

use Benchmark;

- perl doc Benchmark
- Built in module which encapsulates a number of routines to help you figure out how long it takes to execute some code.

```

1 #!/usr/bin/env perl
2
3 #List::Util::first() is slower than a for loop:
4
5 use Benchmark qw(:all :hireswallclock);
6 use List::Util qw(first);
7
8 my @list = 1..100;
9
10 my $results = timethese(1_000_000, {
11   'first' => sub {
12     my $f;
13     $f = first { $_ == 5 } @list;
14     return $f;
15   },
16   'loop' => sub {
17     my $f;
18     for (@list) {
19       if ($_ == 5) {
20         $f = $_;
21         last;
22       }
23     }
24     return $f;
25   },
26 });
27
28 cmpthese($results);

```

use Benchmark;

Built in module encapsulates a number of routines to help you figure out how long it takes to execute some code.

timethese (COUNT, CODEHASHREF, [STYLE])

Time COUNT iterations of CODEHASHREF.

cmpthese (COUNT, CODEHASHREF, [STYLE])

or

cmpthese (RESULTSHASHREF, [STYLE])

Uses timethese (or the results of a timethese() call and outputs in a comparison table

```
$ perl simpleloop2.pl
Benchmark: timing 1000000 iterations of first, loop...
  first: 1.59767 wallclock secs ( 1.48 usr +  0.01 sys =  1.49 CPU) @
671140.94/s (n=1000000)
  loop: 1.08002 wallclock secs ( 0.92 usr +  0.01 sys =  0.93 CPU) @
1075268.82/s (n=1000000)
```

	Rate	first	loop
first	671141/s	--	-38%
loop	1075269/s	60%	--

output from `timethese()` is the default style 'auto' key of coderef followed by the times for 'wallclock' time, user time, and system time followed by the rate

output from `cmpthese()` gives us a comparison chart sorted from slowest to fastest, and shows the percent speed difference between each pair of tests.

Things to consider

- Focus on code that will be executed the most (think loops)
 - Are there expensive comparisons/computations that can be cached sensibly?
 - Are there chains of comparisons that aren't optimized statistically?
- Unnecessary sorting?
- Are you reinventing the wheel?

A simple text parsing script

- Uses a package for creating objects
- Simple parsing of a zone file into DNS records: hash 'em if we know how
- 20K lines to parse

```

1 #!/usr/bin/env perl -l
2 use strict;
3 use warnings;
4 use RR;
5 use Net::DNS::RR;
6 use Benchmark qw(:hireswallclock);
7
8 my $t0 = new Benchmark;
9 while (my $line = <>) {
10     chomp($line);
11
12     # Ignore blank lines
13     next unless $line;
14
15     my $obj = RR->new($line);
16
17     # Generate Net::DNS::RRs
18     my $rr;
19     if ($obj->as_hash()) {
20         $rr = Net::DNS::RR->new($obj->as_hash());
21     } else {
22         $rr = Net::DNS::RR->new($obj->as_string());
23     }
24 }
25 my $t1 = new Benchmark;
26 my $runtime = timestr(timediff($t1,$t0));
27
28 print "Zone parse time: $runtime";

```

use Benchmark;

Built in module encapsulates a number of routines to help you figure out how long it takes to execute some code.

new():

Returns the current time as an object the Benchmark methods use

timediff(T1 , T2):

A Benchmark object representing the difference between two Benchmark times, suitable for passing to timestr();

timestr (TIMEDIFF, [STYLE, [FORMAT]]):

returns a string in the requested format suitable for printing. Format defaults to '%5.2f'.

Benchmark




```
$ perl zoneparse.pl zone.com.txt
```

```
Zone parse time: 4.61972 wallclock secs ( 4.55 usr + 0.02 sys = 4.57 CPU)
```



- 📌 4.61 seconds to process the file
- 📌 Good? Bad? Can it be better?

Profiling Packages

Devel::DProf

-  Built in, produces an output file, utility to format that
-  watches subroutine calls noting elapsed time
-  totals each run into a total time spent in the subroutine

Devel::SmallProf

-  Install from CPAN
-  Human readable output file, clunky for programs with imported libraries

Devel::NYTProf

Devel::NYTProf

<http://search.cpan.org/~timb/Devel-NYTProf-4.06/>

- Devel::NYTProf from CPAN is a powerful, fast, feature-rich perl source code profiler*
- Statement and Subroutine profiling showing Inclusive and Exclusive Time
 - Inclusive includes time spent in subroutines called from within another subroutine
- Handy report HTML generator

Run the script with Devel::NYTProf

```
$ perl -d:NYTProf zoneparse.pl zone.com.txt
```

- `-d` flag starts the debug mode which is shorthand for `-MDevel::`
 - loads the module `Devel::NYTProf` before running the provided script
- produces `nytprof.out` file
- Adds a little overhead

```
$ nytprofhtml -o ./nytprof_run1 -f ./nytprof_run1.out --open
Reading ./nytprof_run1.out
Processing ./nytprof_run1.out data
Writing sub reports to ./nytprof_run1 directory
100% ...
Writing block reports to ./nytprof_run1 directory
100% ...
Writing line reports to ./nytprof_run1 directory
100% ...
```

- 📌 nytprofhtml generates HTML report

- 📌 Useful flags for keeping multiple runs

- 📌 -f --file: file name to use; defaults to ./nytprof.out

- 📌 -o --out: the output directory to place all the html files

Performance Profile Index

For zoneparse.pl

Run on Wed Jan 19 13:51:21 2011
Reported on Wed Jan 19 14:42:23 2011

Profile of zoneparse.pl for 13.0s (of 19.3s), executing 8071444 statements and 2414346 subroutine calls in 48 source files and 8 string evals.

/Library/Perl/5.10.0/IO/Socket/INET.pm

Top 15 Subroutines

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
90041	1	1	2.31s	2.69s	Net::DNS::presentation2wire
60037	1	1	1.11s	1.23s	Net::DNS::wire2presentation
40019	2	2	592ms	3.28s	Net::DNS::name2labels
90018	9	1	505ms	615ms	Net::DNS::Header::AUTOLOAD
10006	1	1	467ms	2.76s	Net::DNS::RR::new from string
20004	2	2	429ms	1.55s	Net::DNS::Packet::dn comp
10002	1	1	359ms	3.38s	Net::DNS::Packet::data
10002	1	1	350ms	1.99s	Net::DNS::RR::data
624759	2	1	348ms	348ms	Net::DNS::CORE:unpack (opcode)
10002	1	1	333ms	6.38s	Net::DNS::RR::normalize rdata
10002	1	1	313ms	8.66s	Net::DNS::RR::new from hash
20004	2	2	289ms	527ms	Net::DNS::Packet::push
10002	1	1	279ms	1.04s	Net::DNS::Packet::parse
20015	6	4	241ms	3.65s	Net::DNS::stripdot
10002	1	1	212ms	841ms	Net::DNS::RR::parse

See all 585 subroutines

Source Code Files — ordered by exclusive time then name

Stmts	Exclusive Time	Reports	Source File
4106502	4.96s	line • block • sub	Net/DNS.pm
1220447	2.42s	line • block • sub	Net/DNS/RR.pm (including 1 string eval)
1030219	1.54s	line • block • sub	Net/DNS/Packet.pm
790182	1.05s	line • block • sub	Net/DNS/Header.pm
290131	534ms	line • block • sub	/Users/lhagemann/Documents/Presentations


```
1 #!/usr/bin/perl -l
2
3 use strict;
4 use warnings;
5
6 use RR;
7 use Net::DNS::RR;
8
9 use Benchmark qw(:hireswallclock);
10
11 my $t0 = new Benchmark;
12
13 while (my $line = <>) {
14     chomp($line);
15
16     # Ignore blank lines
17     next unless $line;
18
19     my $obj = RR->new($line);
20
21     # Generate Net::DNS::RRs
22     my $rr = Net::DNS::RR->new($obj->as_string());
23 }
24
25 my $t1 = new Benchmark;
26 my $runtime = timestr(timediff($t1,$t0));
27
28 print "Zone parse time: $runtime";
```

Benchmark

```
$ perl zoneparse2.pl zone.com.txt  
Zone parse time: 2.03943 wallclock secs ( 2.01 usr + 0.01 sys = 2.02 CPU)
```

- 📌 4.61 seconds down to 2.03 seconds
- 📌 > 50% speed up! Any others?

```
$ perl -d:NYTProf zoneparse2.pl zone.com.txt  
Zone parse time: 10 wallclock secs ( 9.43 usr + 0.03 sys = 9.46 CPU)
```

```
$ nytprofhtml -o ./nytprof_run2 -f ./nytprof_run2.out --open  
Reading ./nytprof_run2.out  
Processing ./nytprof_run2.out data  
Writing sub reports to ./nytprof_run2 directory  
100% ...  
Writing block reports to ./nytprof_run2 directory  
100% ...  
Writing line reports to ./nytprof_run2 directory  
100% ...
```

Performance Profile Index

For zoneparse2.pl

Run on Tue Feb 15 10:25:47 2011
Reported on Tue Feb 15 10:27:39 2011

Profile of zoneparse2.pl for 6.22s (of 9.61s), executing 4236315 statements and 1326375 subroutine calls in 48 source files and 8 string evals.

/Library/Perl/5.10.0/IO/Socket/INET6.pm

Top 15 Subroutines

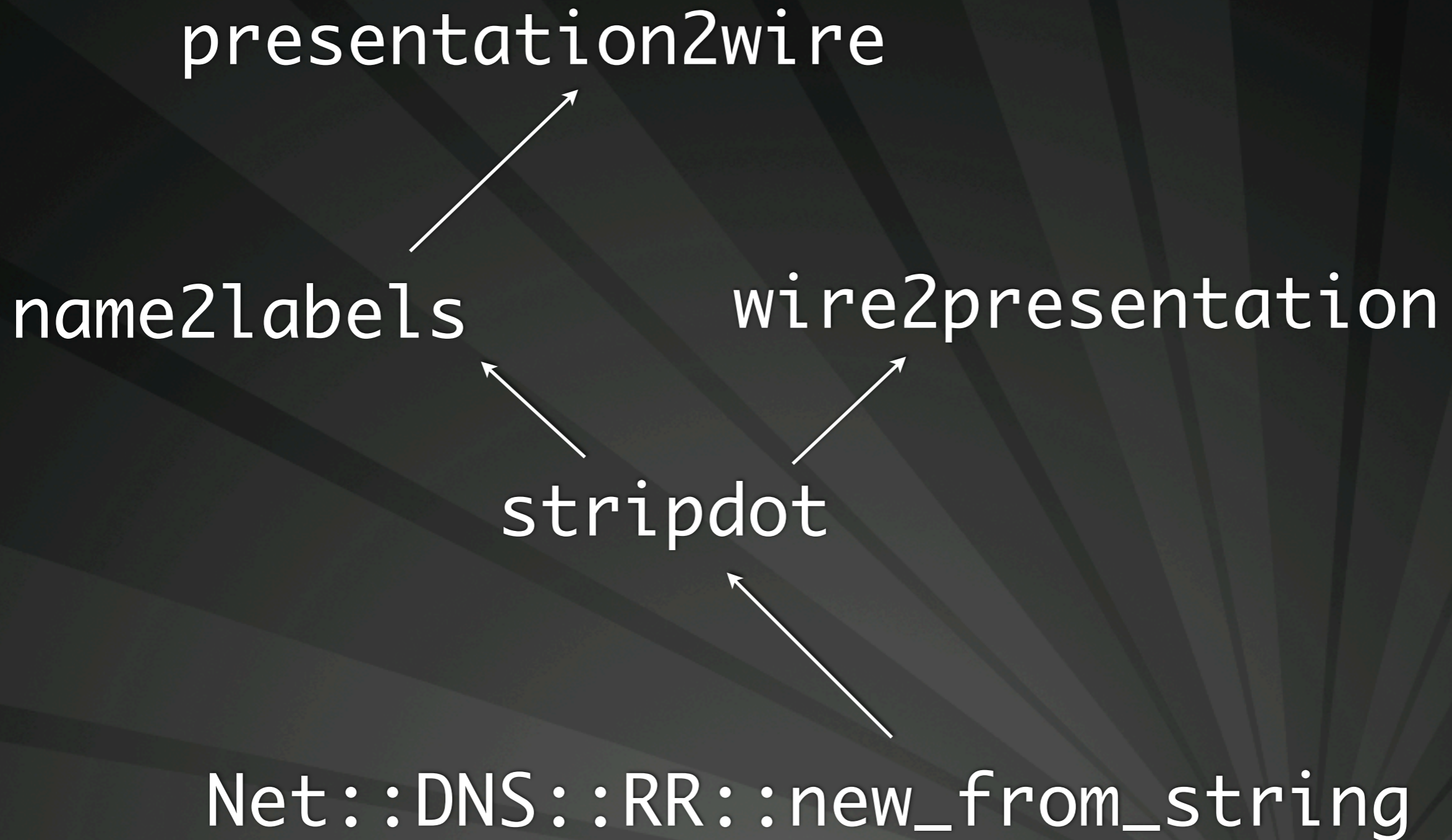
Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
60037	1	1	1.55s	1.81s	Net::DNS::presentation2wire
60037	1	1	1.14s	1.26s	Net::DNS::wire2presentation
20008	1	1	921ms	5.27s	Net::DNS::RR::new from string
20015	1	1	371ms	2.18s	Net::DNS::name2labels
495853	2	1	273ms	273ms	Net::DNS::CORE:unpack (opcode)
20015	5	4	240ms	3.67s	Net::DNS::stripdot
10000	1	1	209ms	240ms	Net::DNS::RR::AAAA::normalize AAAA
20008	1	1	192ms	362ms	RR::new
10002	1	1	111ms	127ms	RR::A::set
80032	4	1	107ms	107ms	Net::DNS::RR::CORE:match (opcode)
20008	1	1	102ms	103ms	Net::DNS::RR::get subclass
217909	1	1	92.0ms	92.0ms	Net::DNS::CORE:pack (opcode)
10000	1	1	81.9ms	322ms	Net::DNS::RR::AAAA::new from string
140056	6	1	73.8ms	73.8ms	Net::DNS::RR::CORE:subst (opcode)
20008	1	1	72.0ms	5.34s	Net::DNS::RR::new

See [all 571 subroutines](#)

Source Code Files — ordered by exclusive time then name

Stmts	Exclusive Time	Reports	Source File
2981884	3.30s	line • block • sub	Net/DNS.pm
720347	1.18s	line • block • sub	Net/DNS/RR.pm (including 1 string eval)
250115	413ms	line • block • sub	/Users/lhagemann/Documents/Presentations/PerlProfiling/RR.pm
80063	310ms	line • block • sub	/Users/lhagemann/Documents/Presentations/PerlProfiling/zoneparse2.pl

Net::DNS



```
1 #!/usr/env/bin perl -l
2
3 use strict;
4 use warnings;
5
6 use RR;
7 use Net::DNS;
8 use Net::DNS::RR;
9 use Benchmark qw(:hireswallclock);
10
11 sub stripdot {
12     my ($str) = @_ ;
13     #Replace any period at the end of a label that is not escaped by '\'
14     $str =~ s{(?<!\)\)\.\s*$}{};
15     return $str;
16 }
17
18 #override the Net::DNS stripdot
19 *Net::DNS::RR::stripdot = \&stripdot;
20
21 my $t0 = new Benchmark;
22
23 while (my $line = <>) {
24     ...
25 }
26
27 my $t1 = new Benchmark;
28 my $runtime = timestr(timediff($t1,$t0));
29 print "Zone parse time: $runtime";
```

Benchmark

```
$ perl zoneparse3.pl zone.com.txt
Subroutine Net::DNS::RR::stripdot redefined at zoneparse3.pl line 19.
Zone parse time: 1.10183 wallclock secs ( 1.10 usr + 0.00 sys = 1.10 CPU)
```

- 2.03 seconds to 1.10 seconds
- Another ~50% speed up!

```
$ perl -d:NYTProf zoneparse3.pl zone.com.txt
Subroutine Net::DNS::RR::stripdot redefined at zoneparse3.pl line 19.
Zone parse time: 3 wallclock secs ( 3.51 usr + 0.02 sys = 3.53 CPU)
```

```
$ nytprofhtml -o ./nytprof_run3 -f ./nytprof_run3.out --open
Reading ./nytprof_run3.out
Processing ./nytprof_run3.out data
Writing sub reports to ./nytprof_run3 directory
100% ...
Writing block reports to ./nytprof_run3 directory
100% ...
Writing line reports to ./nytprof_run3 directory
100% ...
```

Performance Profile Index

For zoneparse3.pl

Run on Tue Feb 15 11:56:46 2011
Reported on Tue Feb 15 11:57:45 2011

Profile of zoneparse3.pl for 2.73s (of 3.70s), executing 1315465 statements and 492803 subroutine calls in 48 source files and 8 string evals.

/Library/Perl/5.10.0/IO/Socket/INET6.pm

Top 15 Subroutines

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
20008	1	1	925ms	1.78s	Net::DNS::RR::new from string
10000	1	1	211ms	240ms	Net::DNS::RR::AAAA::normalize AAAA
20008	1	1	197ms	363ms	RR::new
20008	1	1	115ms	181ms	main::stripdot
10002	1	1	109ms	124ms	RR::A::set
80032	4	1	104ms	104ms	Net::DNS::RR::CORE:match (opcode)
20008	1	1	103ms	104ms	Net::DNS::RR::get subclass
10000	1	1	82.5ms	322ms	Net::DNS::RR::AAAA::new from string
140056	6	1	74.7ms	74.7ms	Net::DNS::RR::CORE:subst (opcode)
20008	1	1	71.7ms	1.85s	Net::DNS::RR::new
20008	1	1	66.0ms	66.0ms	main::CORE:subst (opcode)
10002	1	1	63.3ms	65.3ms	Net::DNS::RR::A::new from string
20008	1	1	55.8ms	55.8ms	RR::as string
10006	1	1	42.4ms	42.4ms	RR::set
20013	1	1	32.7ms	32.7ms	main::CORE:readline (opcode)

See all 579 subroutines


41	6.59ms	line • block • sub	Data/Dumper.pm
1021	5.86ms	line • block • sub	Net/DNS.pm
95	4.12ms	line • block • sub	Benchmark.pm (including 1 string eval)
31	3.69ms	line • block • sub	IO/Handle.pm (including 1 string eval)

77% speed up

- Total run time: 4.61 seconds down to 1.10 seconds
- Time in external module reduced from nearly 5 secs to 6ms.
 - This includes the overhead of calling the profiling module

References

CPAN

 <http://search.cpan.org/~timb/Devel-NYTProf-4.06/>

 <http://search.cpan.org/~salva/Devel-SmallProf-2.02/>

PerlMonks.org

 [Perl Best Practices](#) by [Damian Conway](#)

 [Modern Perl](#) by chromatic