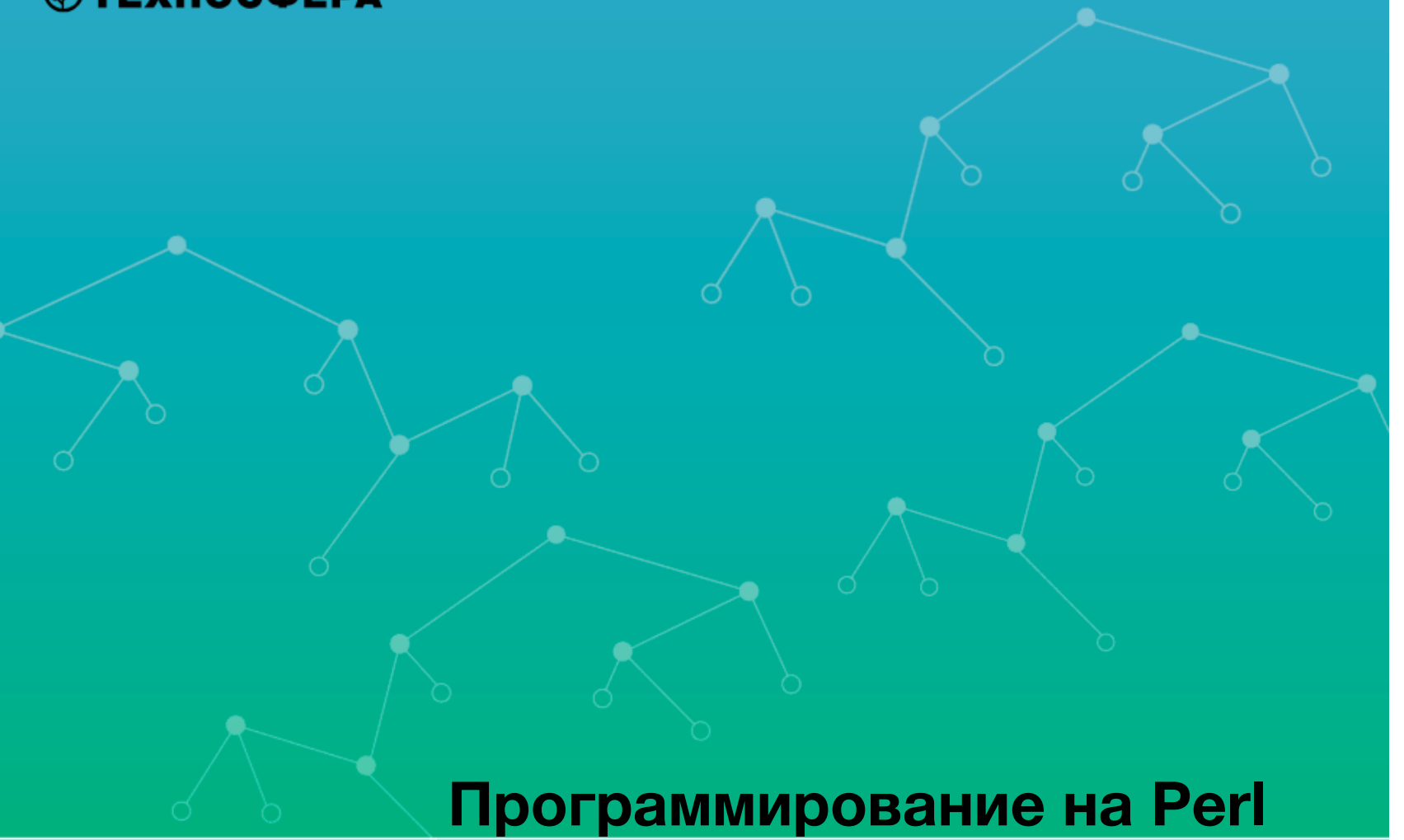
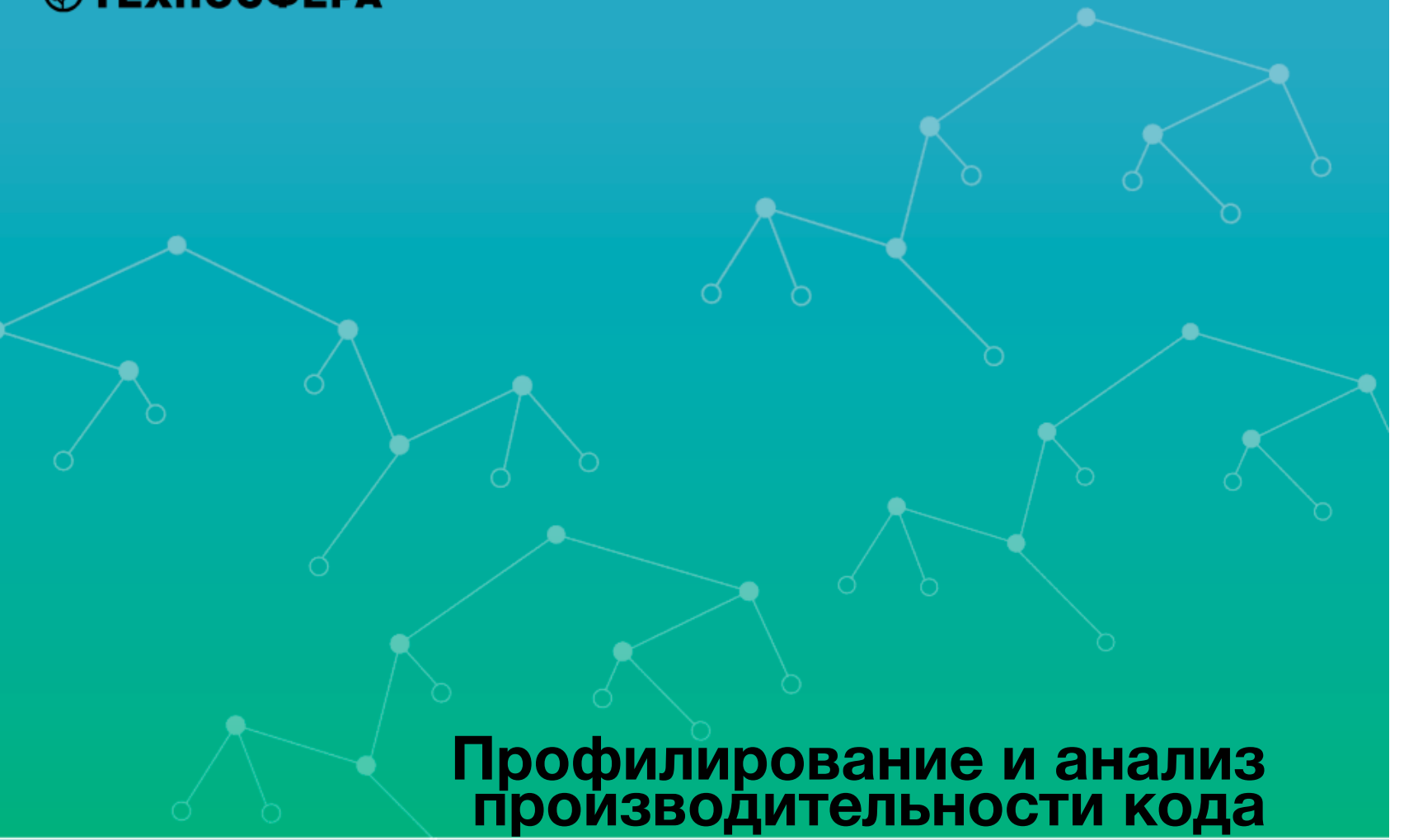
ТЕХНОСФЕРА



Программирование на Perl



Профилирование и анализ производительности кода

Отметьтесь на портале!

Последняя лекция! :)

Содержание

Профилирование — сбор характеристик работы программы, таких как время выполнения отдельных фрагментов или потребления иных ресурсов

Ресурс №1: CPU

CPU time vs real time (wallclock)

cpu time

- низкое разрешение (0.01s)
- нет влияния от загрузки системы
- не включает время io

real time

- высокое разрешение (µs, ns)
- влияние от загрузки системы
- включает время io

time (unix tool)

```
$ time perl -MLWP::UserAgent -E \
'LWP::UserAgent->new->get("https://mail.ru/");'
```

```
real    0m0.056s
user    0m0.043s
sys     0m0.008s
```

0.056s / request?

time (unix tool)

```
$ time perl -MLWP::UserAgent -E \
'LWP::UserAgent->new->get("https://mail.ru/")
 for 1..100;'
```

```
real    0m0.164s
user    0m0.104s
sys     0m0.022s
```

0.164s/100 = 0.016s/request?

Devel::NYTProf

New York Times Profiler

```
$ perl -d:NYTProf -MLWP::UserAgent -E \
'LWP::UserAgent->new->get("https://mail.ru/")
 for 1..100;'

$ ls -la *out
-rw-r--r-- 1   mons staff 314206   nytprof.out

$ nytprofhtml

$ ls -lad nytprof
drwxr-xr-x 200 mons staff 6800     nytprof
```

Devel::NYTProf

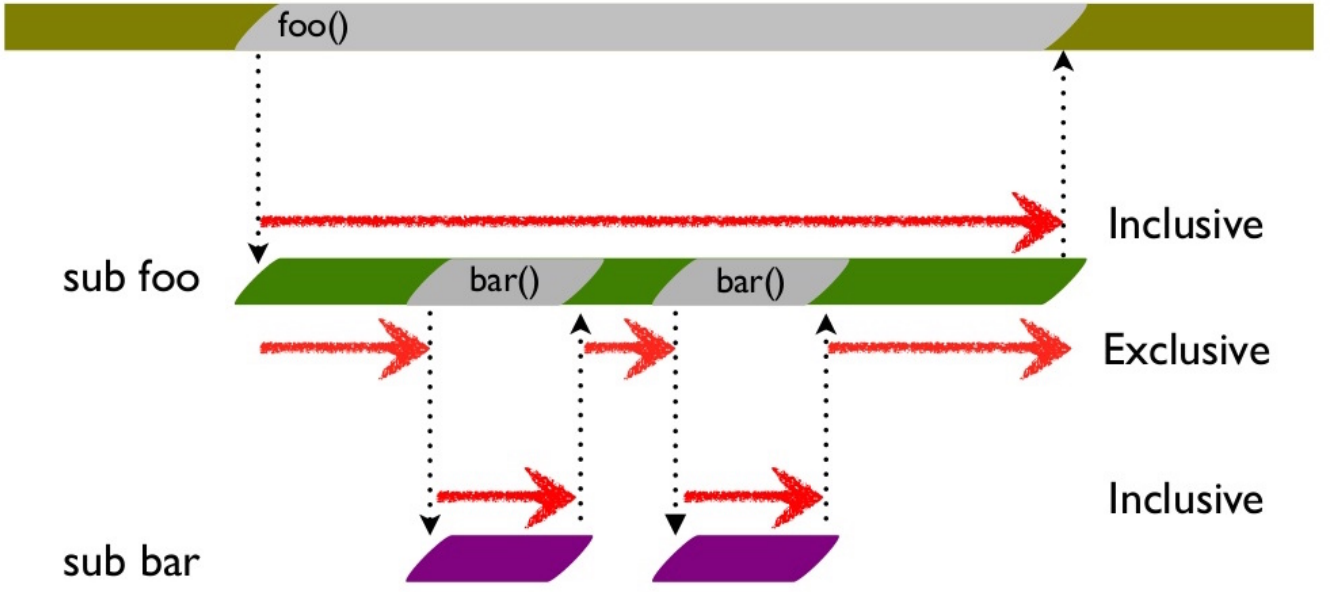
Top 15 Subroutines

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
100	1	1	17.0ms	75.3ms	LWP::UserAgent::send_request
1	1	1	15.9ms	50.6ms	LWP::UserAgent::BEGIN@10
700	2	1	7.64ms	8.53ms	HTTP::Headers::_header
100	1	1	7.17ms	23.5ms	URI::new
100	1	1	6.90ms	16.2ms	Carp::caller_info
100	1	1	6.78ms	7.02ms	LWP::Protocol::implementor
1	1	1	5.90ms	75.8ms	main::BEGIN@0 (xsub)
200	2	1	5.32ms	6.47ms	URI::scheme
100	1	1	5.31ms	5.66ms	URI::implementor
1	1	1	4.70ms	4.81ms	Config::AUTOLOAD
3	3	3	4.59ms	4.59ms	XSLoader::load
1	1	1	4.14ms	6.53ms	LWP::UserAgent::BEGIN@12
600	3	2	3.63ms	11.2ms	HTTP::Headers::header
600	7	2	3.30ms	14.3ms	HTTP::Message::__ANON__[/opt/local/lib
100	1	1	3.24ms	126ms	LWP::UserAgent::get

See all 723 subroutines

Devel::NYTProf

Inclusive time vs Exclusive time



Devel::NYTProf

```
$ NYTPROF=start=init \
  perl -d:NYTProf -MLWP::UserAgent -E \
 'LWP::UserAgent->new->get("https://mail.ru/")
 for 1..100;'

$ nytprofhtml
```

Devel::NYTProf

Top 15 Subroutines

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
100	1	1	9.66ms	14.5ms	URI::_server::_host_escape
700	2	1	8.42ms	9.39ms	HTTP::Headers::_header
100	1	1	7.86ms	8.14ms	LWP::Protocol::implementor
100	1	1	6.59ms	8.87ms	LWP::UserAgent::default_header
100	1	1	4.97ms	68.5ms	LWP::UserAgent::send_request
100	1	1	4.88ms	5.03ms	LWP::UserAgent::protocols_forbidden
600	3	2	4.10ms	12.4ms	HTTP::Headers::header
100	1	1	4.02ms	4.35ms	URI::implementor
100	1	1	3.95ms	4.08ms	HTTP::Headers::_sorted_field_names
600	7	2	3.43ms	15.8ms	HTTP::Message::__ANON__[/opt/local/lib/perl5/s
100	1	1	3.16ms	11.3ms	Carp::caller_info
100	1	1	3.13ms	18.2ms	LWP::UserAgent::new
100	1	1	2.73ms	4.10ms	overload::AddrRef
1100	5	2	2.49ms	2.49ms	HTTP::Message::_elem
100	1	1	2.45ms	14.1ms	LWP::UserAgent::prepare_request

See all 431 subroutines

Devel::NYTProf

```
$ NYTPROF=start=no \
  perl -d:NYTProf -MLWP::UserAgent -E \
 'my $ua = LWP::UserAgent->new;
  DB::enable_profile();
  $ua->get("https://mail.ru/") for 1..100;
  DB::disable_profile();'

$ nytprofhtml
```

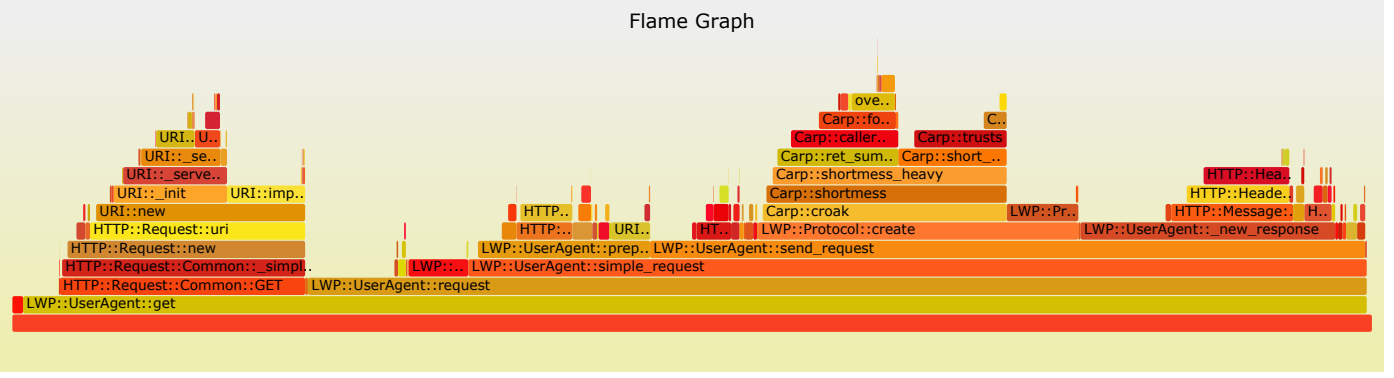
Devel::NYTProf

Top 15 Subroutines

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
600	2	1	10.9ms	11.7ms	HTTP::Headers::_header
100	1	1	8.99ms	110ms	LWP::UserAgent::request
100	1	1	8.84ms	26.5ms	LWP::UserAgent::_new_response
100	1	1	7.66ms	8.12ms	URI::implementor
500	5	1	7.45ms	8.48ms	LWP::UserAgent::run_handlers
200	2	1	7.22ms	9.60ms	Carp::trusts
100	1	1	7.20ms	7.44ms	LWP::Protocol::implementor
100	1	1	5.27ms	5.40ms	HTTP::Headers::_sorted_field_names
100	1	1	4.35ms	74.3ms	LWP::UserAgent::send_request
200	2	1	3.76ms	4.90ms	URI::scheme
100	1	1	3.68ms	140ms	LWP::UserAgent::get
1	1	1	3.27ms	3.99ms	URI::_idna::BEGIN@7
600	7	2	3.19ms	18.3ms	HTTP::Message::__ANON__[/opt/local/lib
500	2	2	3.02ms	13.8ms	HTTP::Headers::header
100	1	1	2.90ms	11.1ms	Carp::caller_info

See all 401 subroutines

Devel::NYTProf



Devel::NYTProf

```
                                  # spent 11.7ms (10.9+865µs) within HTTP::Headers::_header which was called 600 times, avg 20µs/call:
                                  # 500 times (9.88ms+849µs) by HTTP::Headers::header at line 84, avg 21µs/call
                                  # 100 times (986µs+16µs) by HTTP::Headers::init_header at line 111, avg 10µs/call
                                  {
600    269µs                          my($self, $field, $val, $op) = @_;

600    252µs                          Carp::croak("Illegal field name '$field'")
                                          if rindex($field, ':') > 1 || !length($field);

600    965µs  600    130µs           unless ($field =~ /^:/) {
                                      # spent   130µs making 600 calls to HTTP::Headers::CORE:match, avg 216ns/call
600    200µs                              $field =~ tr/_/-/ if $TRANSLATE_UNDERSCORE;
600    68µs                               my $old = $field;
600    100µs                              $field = lc $field;
600    433µs                              unless($standard_case{$field} || $self->{'::std_case'}{$field}) {
                                              # generate a %std_case entry for this field
200    5.75ms 800    736µs                  $old =~ s/\b(\w)/\u$1/g;
                                              # spent   449µs making 600 calls to HTTP::Headers::CORE:substcont, avg 748ns/call
                                              # spent   287µs making 200 calls to HTTP::Headers::CORE:subst, avg 1µs/call
200    207µs                                  $self->{'::std_case'}{$field} = $old;
                                          }
                                      }

600    29µs                           $op ||= defined($val) ? 'SET' : 'GET';
600    80µs                           if ($op eq 'PUSH_H') {
                                          # Like PUSH but where we don't care about the return value
                                          if (exists $self->{$field}) {
                                              my $h = $self->{$field};
                                              if (ref($h) eq 'ARRAY') {
                                                  push(@$h, ref($val) eq "ARRAY" ? @$val : $val);
                                              }
                                              else {
                                                  $self->{$field} = [$h, ref($val) eq "ARRAY" ? @$val : $val]
                                              }
                                              return;
```

Devel::NYTProf

```
spent 11.7ms (10.9+865µs) within
    HTTP::Headers::_header which was called
    600 times, avg 20µs/call:

    500 times (9.88ms+849µs)
        by HTTP::Headers::header at line 84,
        avg 21µs/call
    100 times (986µs+16µs)
        by HTTP::Headers::init_header at line 111,
        avg 10µs/call
```

Devel::NYTProf

```
unless ($field =~ /^:/) {
    # spent   130µs making 600 calls to
    HTTP::Headers::CORE:match, avg 216ns/call
```

```
$old =~ s/\b(\w)/\u$1/g;
    # spent   449µs making 600 calls to
        HTTP::Headers::CORE:substcont,
        avg 748ns/call
    # spent   287µs making 200 calls to
        HTTP::Headers::CORE:subst,
        avg 1µs/call
```

Первое правило оптимизации программ:

Не делайте этого

Второе правило оптимизации программ (только для экспертов):

Не делайте этого пока

Michael A. Jackson

Как профилировать правильно

1. Даём репрезентативную нагрузку
2. Смотрим общее время функций
3. Время выглядит адекватно?
4. Исследуем самые медленные
5. Исправляем простые проблемы
6. Профилируем заново
7. Производительность хорошая? СТОП!
8. Повторите 1-2 раза
9. Переходите к другим улучшениям

Tim Bunce

```
my $str = "x"x1024;

sub func1 {
    my $s = $str;
    substr($s,0,10,"");
}

sub func2 {
    my $s = $str;
    $s = substr($s,10);
}
```

use Benchmark;

```
use Benchmark qw(:all);

timethis(1e6,\&func1);
timethis(-1,\&func1);
```

```
timethis 1000000:  0 wallclock secs
    ( 0.20 usr +  0.01 sys =  0.21 CPU)
        @ 4761904.76/s (n=1000000)
(warning: too few iterations for a reliable count)

timethis for 1:  1 wallclock secs
    ( 1.02 usr +  0.00 sys =  1.02 CPU)
        @ 4665608.82/s (n=4758921)
```

```
use Benchmark qw(:all);

timethese -1, {
    inplace => \&func1,
    copy    => \&func2,
};
```

```
Benchmark: running copy, inplace for
    at least 1 CPU seconds...
copy:  2 wallclock secs
    ( 1.00 usr +  0.02 sys =  1.02 CPU)
        @ 4151600.98/s (n=4234633)
inplace:  1 wallclock secs
    ( 1.09 usr +  0.01 sys =  1.10 CPU)
        @ 4549606.36/s (n=5004567)
```

```
cmpthese timethese -1, {
    inplace => \&func1,
    copy    => \&func2,
};
```

```
Benchmark: running copy, inplace for
    at least 1 CPU seconds...
copy:  2 wallclock secs
    ( 1.07 usr +  0.00 sys =  1.07 CPU)
        @ 3957600.93/s (n=4234633)
inplace:  1 wallclock secs
    ( 1.07 usr +  0.00 sys =  1.07 CPU)
        @ 4777389.72/s (n=5111807)
             Rate    copy inplace
copy    3957601/s      --    -17%
inplace 4777390/s     21%      --
```

```
use Dumbbench;
my $bench = Dumbbench->new(
    target_rel_precision => 0.005,
    initial_runs         => 50,
);
$bench->add_instances(
    Dumbbench::Instance::PerlSub
      ->new(name => 'inplace', code => \&func1),
    Dumbbench::Instance::PerlSub
      ->new(name => 'copy', code => \&func2),
);
$bench->run();
$bench->report();
```

```
inplace: Ran 93 iterations (26 outliers).
inplace: Rounded run time per iteration:
    1.5462e-07 +/- 3.4e-10 (0.2%)
copy: Ran 56 iterations (6 outliers).
copy: Rounded run time per iteration:
    4.0881e-07 +/- 6.0e-10 (0.1%)
```

/dev/hands

```
use List::Util qw(sum); my $N = 4e6;
my $cpu = sum((times)[0,1]);

func1() for 1..$N;

my $cpu1 = sum((times)[0,1]);
printf "%0.3fs CPU, %0.1fns/call\n",
$cpu1-$cpu, (1e9/$N)*($cpu1-$cpu);

func2() for 1..$N;

my $cpu2 = sum((times)[0,1]);
printf "%0.3fs CPU, %0.1fns/call\n",
$cpu2-$cpu1, (1e9/$N)*($cpu2-$cpu1);
```

```
1.320s CPU, 330.0ns/call
1.460s CPU, 365.0ns/call
```

Оптимизация:
Простые локальные изменения

Вынести постоянные выражения за пределы цикла

```
for (...) {
    my $a = 123;
    call_sub($a);
}
```

```
my $a = 123;
for (...) {
    call_sub($a);
}
```

Избегать повторения цепочек аксессоров

```
Avoid->repeated()->chains()->of->accessors();
Avoid->repeated()->chains()->of->accessors();
Avoid->repeated()->chains()->of->accessors();
```

```
my $one = Avoid->repeated()->chains()->of;
$one->accessors();
$one->accessors();
$one->accessors();
```

Не вызывайте лишнего

```
use constant DEBUG => 0;
my $unused = $self->get_something;
my $do_log = $logger->logging;

for (...) {
    $logger->log(...) if $do_log;
    do_debug(...) if DEBUG;
    ...
}
```

Выходите раньше, инициализируйте позже

```
sub {
    return unless @_;
    return 1 if @_ == 1;
    for (@_) { ... }
    ...
}

sub {
    if () { ... }
    elsif () { ... }
    else {
        return;
    }
}
```

Избегайте ненужных проверок

```
- return exists $hash{$something}{$key}
-             ? $hash{$something}{$key}
-             : undef

+ return $hash{$something}{$key}
```

Используйте аргументы без распаковки в очень "горячих" функциях

```
sub add ($$) {
    return $_[0] + $_[1]
}
```

Внести цикл под вызов

```
$object->walk($_) for @dogs;

$object->walk_that(\@dogs);
```

Оптимизация: значительные изменения

Выбрать более быстрый модуль

http://neilb.org/reviews/

Обновить perl

- Сборка без тредов дает до +30%
- Сборка без отладки дает до +5%
- 5.10 -> 5.16 может дать до +15%
- 5.16 -> 5.20 может дать до +20%

Оптимизация алгоритмов:

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

Переписывание "горячих" участков на XS

Ресурс №2: память

Классическая утечка: циклическая ссылка

```
my $x;$x = \$x; # << selfref

my $h = {};
$h->{key} = $h; # << cyclic ref

my $s;$s = sub {
    $s->(); # << ref by closure
};
```

use Devel::Leak;

```
use Devel::Leak;
Devel::Leak::NoteSV($handle);

my $x;$x = \$x;
my $h = {};$h->{key} = $h;
my $s;$s = sub {$s->();};

Devel::Leak::CheckSV($handle);
```

use Devel::Leak;

```
new 0x7fc84b021b28 :
new 0x7fc84b021b40 :
new 0x7fc84b021b58 :
new 0x7fc84b021b70 :
new 0x7fc84b021c00 :
new 0x7fc84b021c18 :
new 0x7fc84b004ce8 :
new 0x7fc84b004ec8 :
```

perl built with -DDEBUGGING

```
new 0x2410de8 : SV = PVHV(0x2416e40) at 0x2410de8
  REFCNT = 2
  FLAGS = (SHAREKEYS)
  ARRAY = 0x24aa640  (0:7, 1:1)
  hash quality = 100.0%
  KEYS = 1
  FILL = 1
  MAX = 7
  RITER = -1
  EITER = 0x0
new 0x2410f68 : SV = IV(0x2410f58) at 0x2410f68
  REFCNT = 1
  FLAGS = (ROK)
  RV = 0x2410de8
    SV = PVHV(0x2416e40) at 0x2410de8
      REFCNT = 2
      FLAGS = (SHAREKEYS)
      ARRAY = 0x24aa640  (0:7, 1:1)
```

XS version

```
int checkpoint = PL_sv_count;

/* ...some leaky code... */

printf("leaked by %d SV's",
    PL_sv_count - checkpoint);
```

/dev/hands

```
/proc/[pid]/statm
  Provides information about memory usage.

  size   total program size
         (same as VmSize in /proc/[pid]/status)
  resident   resident set size
         (same as VmRSS in /proc/[pid]/status)
  share  shared pages (from shared mappings)
  text   text (code)
  lib    library (unused in Linux 2.6)
  data   data + stack
  dt     dirty pages (unused in Linux 2.6)
```

/dev/hands

```
use POSIX qw( _SC_PAGESIZE sysconf );
my $PG = sysconf(_SC_PAGESIZE);
sub mem () {
    open my $f,'<:raw','/proc/self/statm';
    my ($all,$rss,$shr,$txt,$lib,$data) =
    map $_*$PG, split /\s+/, scalar <$f>;
    return $data;
}

my $before = mem();
for (1..1e5) {
    my %h;$h{key} = \%h;
}
my $after = mem();
printf "Lost %0.1fM\n",
    ($after-$before)/1024/1024;
```

```
Lost 17.7M
```

Manual catch in XS

```
void test(SV *var)
PPCODE:
    SV *leaky = newSVpvs("dummy");
    XSRETURN_UNDEF;
```

```
use Devel::Leak;
Devel::Leak::NoteSV(my $chk);
LeakTest::test($smth);
Devel::Leak::CheckSV($chk);
```

```
new 0x1496a68 :
```

Manual catch in XS

```
void test(SV *var)
PPCODE:
    SV *leaky = newSVpvs("dummy");
    printf("var = %p\n",var);
    SV * ghost = (SV *) ( (char *)var + 0 );
    sv_dump( ghost );
    XSRETURN_UNDEF;
```

```
var = 0x2075c38
SV = NULL(0x0) at 0x2075c38
  REFCNT = 1
  FLAGS = (PADMY)
new 0x2058a68 :
```

```
0x2058a68 - 0x2075c38 = -119248
```

Manual catch in XS

```
void test(SV *var)
PPCODE:
    SV *leaky = newSVpvs("dummy");
    printf("var = %p\n",var);
    SV * ghost = (SV *) ( (char *)var - 119248 );
    sv_dump( ghost );
    XSRETURN_UNDEF;
```

```
var = 0x20f2c38
SV = PV(0x20d3cf0) at 0x20d5a68
  REFCNT = 1
  FLAGS = (POK,pPOK)
  PV = 0x21ab7a0 "dummy"\0
  CUR = 5
  LEN = 16
new 0x20d5a68 :
```

Manual catch in XS

```
void test(SV *var)
PPCODE:
    SV *leaky = sv_2mortal(newSVpvs("dummy"));
    printf("var = %p\n",var);
    SV * ghost = (SV *) ( (char *)var - 119248 );
    sv_dump( ghost );
    XSRETURN_UNDEF;
```

```
var = 0x20f2c38
SV = PV(0x20d3cf0) at 0x20d5a68
  REFCNT = 1
  FLAGS = (POK,pPOK)
  PV = 0x21ab7a0 "dummy"\0
  CUR = 5
  LEN = 16
```

Self-referenced closure

```
my $cb; $cb = sub {
    something_async sub {
        $cb->();
    };
};
$cb->();
```

Not very self-referenced closure

```
use Scalar::Util qw(weaken);
my $cb; $cb = sub {
    my $cb = $cb or return;
    something_async sub {
        $cb->();
    };
};
$cb->(); # initial call
# or $cb->() for 1..$N
weaken($cb);
```

Destruction tracking

```
use Guard;
{
    my %struct;
    # ...
    $struct{__tmp} = guard {
        warn "Object destroyed";
    }
}
```

```
Object destroyed at - line 6.
```

Destruction tracking

```
use Guard;
{
    my %struct;
    # ...
    $struct{__} = \%struct;
    $struct{__tmp} = guard {
        warn "Object destroyed";
    }
}
```

```
Object destroyed at - line 7 during global destruction.
```

Debugging-Perl-Memory-Usage

```
$ PERL_DEBUG_MSTATS=2 perl -e "require Carp"
Memory allocation statistics after compilation: (
   14216 free:   130   117    28     7     9   0
      437    61    36     0     5
   60924 used:   125   137   161    55     7   8
       74   109   304    84    20
Total sbrk(): 77824/21:119. Odd ends: pad+heads+c
Memory allocation statistics after execution:   (
   30888 free:   245    78    85    13     6   2
      315   162    39    42    11
  175816 used:   265   176  1112   111    26  22
      196   178  1066   798    39
Total sbrk(): 215040/47:145. Odd ends: pad+heads+
```

```
use Devel::Gladiator qw(arena_table);
say arena_table();
```

```
ARENA COUNTS:
 1471 SCALAR
  237 GLOB
  184 ARRAY
  114 CODE
   75 HASH
   27 REF
   21 REF-ARRAY
   15 REGEXP
    5 LVALUE
    4 IO::File
    4 REF-version
    4 version
    2 FORMAT
    2 REF-HASH
```

Ресурс №3: дескрипторы

/dev/hands

```
open my $f,'<:raw', '/dev/null';
my $before = fileno($f);
close ($f);

# ...leaky code ...

open my $f,'<:raw', '/dev/null';
my $after = fileno($f);
close ($f);

printf "Leaked by %d descriptors\n",
    $after - $before;
```

```
# perlvar: $SYSTEM_FD_MAX
my $before = $^F;
# ...leaky code ...
my $after = $^F;
```

Resources

- Devel::NYTProf
- Benchmark, Bench
- Dumbbench
- Devel::Leak, Devel::Leak::Object
- Devel::Size, Devel::SizeMe
- DashProfiler
- Dash::Leak
- Devel::Gladiator, Devel::Arena
- Guard

__END__

Оставьте отзыв на портале

Спасибо за внимание!