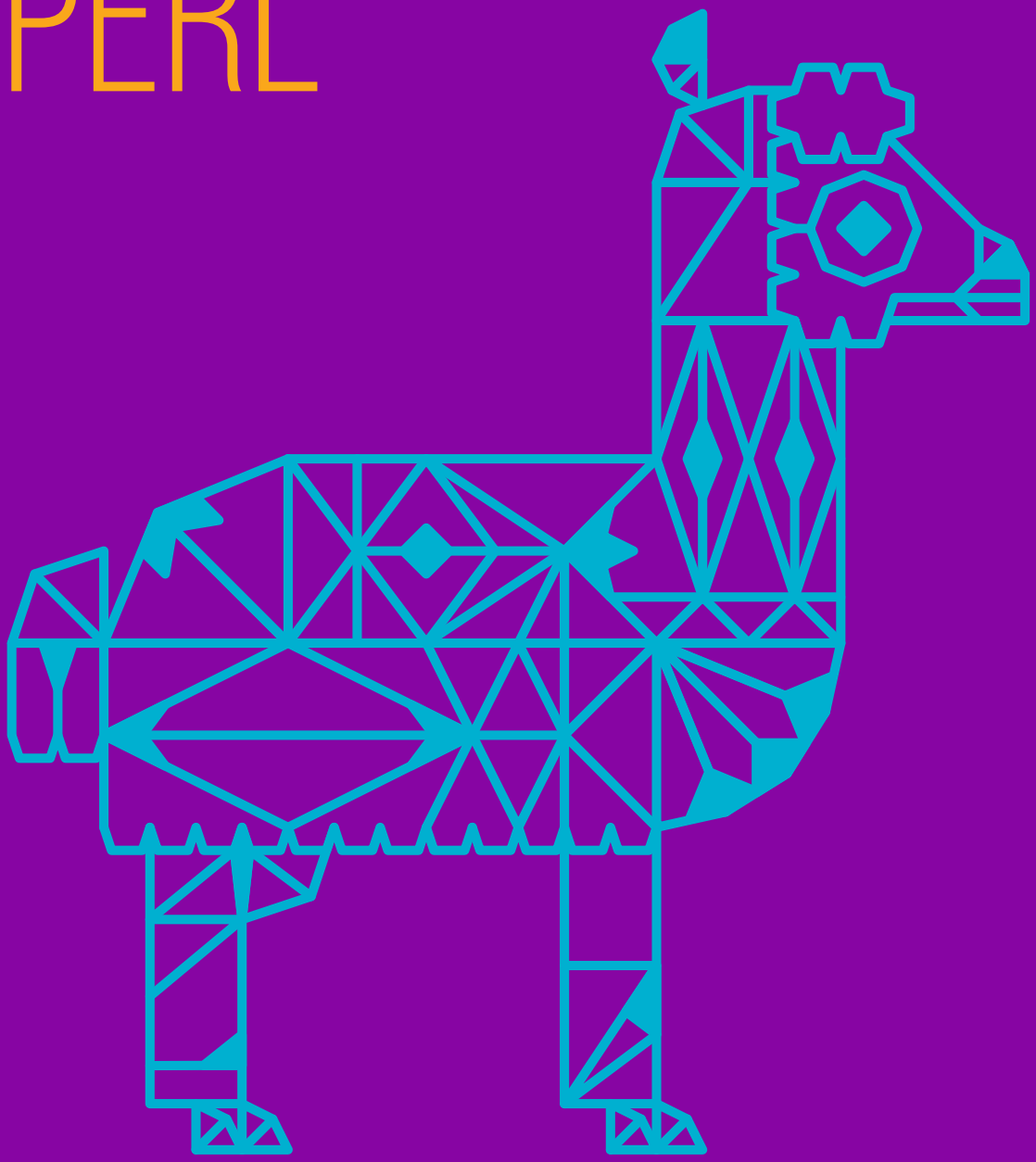
PRAGMATIC PERL

33



11/2015

pragmaticperl.com

Pragmatic Perl 33

pragmaticperl.com

Выпуск 33. Ноябрь 2015

Другие выпуски и форматы журнала всегда можно загрузить с pragmaticperl.com. С вопросами и предложениями пишите на почту editor@pragmaticperl.com.

Комментарии к каждой статье есть в html-версии. Подписаться на новые выпуски можно по ссылке pragmaticperl.com/subscribe.

Авторы статей: Тигран Оганесян, Андрей Шитов, Владимир Леттиев

Обложка: Марко Иваныk

Корректор: Андрей Шитов

Выпускающий редактор: Вячеслав Тихановский

Ревизия: 2015-11-30 19:53

Оглавление

1 От редактора

Друзья, журнал ищет новых авторов. Не упускайте такой возможности! Если у вас есть идеи или желание помочь, пожалуйста, свяжитесь с нами.

Приятного чтения.

■ *Вячеслав Тихановский*

Saint Perl — 7

В декабре в Санкт-Петербурге в седьмой раз пройдет конференция Saint Perl. Мероприятие всегда проходит примерно в день рождения перла, в этом году 19 декабря.

Желающие принять участие приглашаются к регистрации на сайте event.yapcrussia.org/s Там же можно подать заявку на доклад или блиц-доклад.

YAPC::Europe 2016

Стали известны даты проведения следующей европейской конференции

YAPC::Europe. Она пройдет 24–26 августа 2016 в румынском городе Клуж-Напока.

Страница конференции на Фейсбуке: www.facebook.com/events/1931915517032927.

Организаторы — группа Cluj.pm и компания Evozon — за последние три года сумели с нуля построить и развить местное Perl-сообщество, и судя по их нынешней активности, прикладывают большие усилия, поэтому скорее всего конференция будет организована очень хорошо. В любом случае туда стоит поехать, чтобы пообщаться с людьми.

Попасть в Клуж можно через Бухарест, откуда летают несколько самолетов в день. В Румынию пускают с любой действующей шенгенской визой. Более подробная информация о визах и правилах доступна

на сайте консульства.

■ *Андрей Шитов*

3 Сравнение веб-клиентов

Практическое сравнение веб-клиентов на реальной задаче

Недавно я подумывал о смене съемной квартиры, и в связи с этим первый вопрос, который возник у меня, был: «Какова средняя цена квартиры в моем районе?». И поскольку сам пишу на перле, то я подумал написать скрипт для сканирования объявлений в Авито.

Первое, что мне пришло в голову для решения этой бытовой задачи, — использовать `Mojo::UserAgent`, поскольку с ним можно скачать и сразу распарсить страничку, не применяя других модулей. (По правде сказать, он не один такой, есть, например, еще `WWW::Scripter`, но последний медленнее

работает). Получился маленький скриптик:

```
#!/usr/bin/env perl

use strict;
use warnings;
use Mojo::UserAgent;

my $ua = Mojo::UserAgent->new();

my $url_template =
  'https://www.avito.ru/moskva/
    kvartiry/sdam/
    na_dlitelnyy_srok/1-
    komnatnye?p=%s&i=1&metro=85&
    view=list';

my $data = {};

# Скачиваем первую странницу
my $dom = $ua->get( sprintf(
   $url_template, 1 ) )->res->dom
   ;
```

```
16
17 # Определяем последнюю
18 my $last_page = ( $dom->find('div
       .pagination__pages a')->[-1]->
       attr('href') =~ /\?p=(\d+)/ )
       [0];
19
20 # Парсим первую страничку
21 parse_dom( $dom, $data );
22
23 # Качаем и парсим последующие
       странички
24 if ( $last_page > 1 ) {
25     for my $page ( 2 ..
          $last_page ) {
26         my $dom = $ua->get(
              sprintf( $url_template
              , $page ) )->res->dom;
27         parse_dom( $dom, $data );
28     }
29 }
30
31 my ( $count, $area, $price, %min,
```

```
     %max );
my $simple = shift @{ [ keys %
    $data ] };

$min{price} = $max{price} = $data
    ->{$simple}->{price};
$min{ref}   = $max{ref}   =
    $simple;

# Определяем среднюю цену,
    среднюю за квадратный метр,
    минимум, максимум
for my $link ( keys %$data ) {
    $count++;
    $price += $data->{$link}{
        price};
    $area  += $data->{$link}{area
        };
    if ( $data->{$link}{price} <
        $min{price} ) {
        $min{price} = $data->{
            $link}{price};
        $min{ref}   = $link;
```

```
45      }
46      if ( $data->{$link}{price} >
           $max{price} ) {
47          $max{price} = $data->{
               $link}{price};
48          $max{ref}   = $link;
49      }
50  }
51
52  print "count = $count, price = "
       . $price / $count . "
       area_price = " . $price /
       $area . "\n";
53  print "min => $min{price} $min{
       ref}\n";
54  print "max => $max{price} $max{
       ref}\n";
55
56  sub parse_dom {
57      my ( $dom, $data ) = @_;
58
59      $dom->find('div.catalog-list
           div.item')->each(
```

```
60          sub {
61              my $a   = $_->find('
                   div.title a')
                   ->[0];
62              my $ref = $a->attr('
                   href');
63              ( $data->{$ref}{rooms
                   }, $data->{$ref}{
                   area} ) = split
                   /,/, $a->attr('
                   title');
64              $data->{$ref}{price}
                   = $_->find('div.
                   price p')->[0]->
                   text;
65
66              for my $field (qw(
                   price rooms area))
                    {
67                  $data->{$ref}{
                       $field} =~ s/\
                       D//g;
68              }
```

```
69
70          }
71      );
72  }
```

Конечно, для какого-либо проекта сюда надо добавить всякие проверки статуса запроса, валидацию и т.п., но для меня и так вполне сошло. И отработал скрипт довольно быстро.

```
1  $ time ./mojo.pl
2  count =142, price =
       29180.9788732394 area_price =
       755.322457163689
3  min => 5000 /moskva/kvartiry/1-
       k_kvartira_42_m_1117_et.
       _672470440
4  max => 49000 /moskva/kvartiry/1-
       k_kvartira_44_m_618_et.
       _668652789
5
6  real 0m1.599s
```

```
user 0m0.640s
sys 0m0.044s
```

Всего 1,5 секунды, совсем не плохо. Однако он скачал и пропарсил всего 2-3 страницы, а если мне понадобится узнать среднюю не по моему району, а по всему городу/области/стране, то там уже сотни страниц (для Москвы где-то 200 было), и, следовательно, надо это дело оптимизировать. Для начала посмотрим, что у нас медленнее выполняется: парсинг или скачивание данных? Для этой цели я решил использовать модуль `Time::HR`, который работает с наносекундами. Добавив замеры, получил такой скриптик.

```
#!/usr/bin/env perl

use strict;
use warnings;
use Mojo::UserAgent;
```

```
use Data::Dumper;
use Time::HR;

my $ua = Mojo::UserAgent->new();

my $url_template =
  'https://www.avito.ru/moskva/
     kvartiry/sdam/
     na_dlitelnyy_srok/1-
     komnatnye?p=%s&i=1&metro=85&
     view=list';

my %time;
my $data = {};

# Засекаем начало скачивания
$time{1} = gethrtime();

my $dom = $ua->get( sprintf(
    $url_template, 1 ) )->res->dom
    ;

# Конец скачивания и начало
```

```
    парсинга
$time{2} = gethrtime();

parse_dom( $dom, $data );

# Конец парсинга
$time{3} = gethrtime();

my $last_page = ( $dom->find('div
    .pagination__pages a')->[-1]->
    attr('href') =~ /\?p=(\d+)/ )
    [0];

print "Fetching: @{[ $time{2} -
    $time{1} ]}\nparsing: @{[
    $time{3} - $time{2} ]}
\ndelta: @{[ ( $time{3} - $time
    {2} ) * 100 / ( ( $time{2} -
    $time{1} ) - ( $time{3} -
    $time{2} ) ) ]} %\n";
exit;

...
```

Прогоняем

```
./mojo.pl
Fetching: 733339136
parsing: 92294656
delta: 14.3975432094821 %
```

То есть парсинг всего на 14% быстрее, чем скачивание данных. Это конечно несущественно, но начнем оптимизацию со скачивания. Для начала я сравнил известные мне веб-клиенты:

```
LWP::UserAgent
Mojo::UserAgent
WWW::Curl
IO::Socket::SSL (SSL потому как
    Авито на https)
```

И для сравнения я решил теребить не сайт Авито (чтоб ненароком не забанили), а сайт `https://example.com`. Сравнение произвожу просто: делаю запрос, HTML-

код ответа пишу в переменную. Получился такой скрипт. Для сравнения скорости буду использовать модуль Benchmark, а для памяти Memchmark.

```
#!/usr/bin/env perl

use strict;
use warnings;
use Benchmark;
use Memchmark;
use Mojo::UserAgent;
use LWP::UserAgent;
use WWW::Curl::Easy;
use IO::Socket::SSL;

my $curl = WWW::Curl::Easy->new;
my $m    = Mojo::UserAgent->new;
my $l    = LWP::UserAgent->new;
my $sock = IO::Socket::SSL->new('
   example.com:443') or die "
   Cannot construct socket - $@";
my $url  = "https://example.com";
```

```
17
18 my %cmp_hash = (
19     'lwp' => sub { my $res = $l->
          get($url); my $html = $res
          ->decoded_content; },
20     'mojo' => sub { my $html = $m
          ->get($url)->res->body; },
21     'curl' => sub {
22         $curl->setopt(
              CURLOPT_URL, $url );
23         my $html;
24         $curl->setopt(
              CURLOPT_WRITEDATA, \
              $html );
25         $curl->perform;
26     },
27     'sock' => sub {
28         print $sock "GET / HTTP
              /1.1\nHost: example.
              com\nConnection: keep-
              alive\n\n";
29         my $html = "";
30         while (<$sock>) {
```

```
            next if /\r\n$/;
            $html .= $_;
            last if /<\/body>/;
        }
    }
);

Memchmark::cmpthese(%cmp_hash);
Benchmark::cmpthese( 30000, \%
    cmp_hash );
```

Прогон бенчмарка занял огромное время (поначалу ставил 100 000 итераций, но скрипт не отработал за ночь, так что опустил до 30 000):

```
$ time ./benchmark.pl
test: curl, memory used: 2875392
    bytes
test: lwp, memory used: 8388608
    bytes
test: mojo, memory used: 188416
    bytes
```

```
test: sock, memory used: 0 bytes
Rate lwp mojo curl sock
lwp 34.8/s -- -85% -97% -100%
mojo 226/s 550% -- -79% -99%
curl 1096/s 3053% 385% -- -96%
sock 29703/s 85300% 13042% 2609%
    --

real 451m26.547s
user 16m11.740s
sys 0m54.373s
```

Данные моего ноута:

```
Processor: Intel® Core™ i5-2450M
    CPU @ 2.50GHz × 4
Memory: 3.8 GiB
OS: Ubuntu, Release 12.04 (
    precise) 64-bit, Kernel Linux
    3.13.0-67-generic
```

Победителем оказался `IO::Socket::SSL`,

и, по-моему, это просто потому, что он держит соединение (заголовок Connection: keep-alive), а остальные на каждый запрос открывают новый сокет. Я так и не понял, почему `Memchamrk` не зафиксировал потребление памяти для `IO::Socket::SSL`, но для остальных меньше всего памяти потребляет `Mojo::UserAgent`.

Но это пока синхронный вариант скачивания. Посмотрим, что выйдет, если скачивать асинхронно. Для реализации асинхронного скачивания мне были известны два модуля, и еще два (`AnyEvent::HTTP`, `WWW::Curl::Multi`) мне подсказали коллеги:

```
1 Mojo::UserAgent
2 YADA
3 AnyEvent::HTTP
4 WWW::Curl::Multi
```

Чтобы сравнить их производительность, я просто 1000 раз качаю `https://example.com` и складываю результат в массив. Но для начала проделаю это с `IO::Socket::SSL`, чтобы было с чем сравнить. Скрипт для него:

```
#!/usr/bin/env perl

use strict;
use warnings;
use IO::Socket::SSL;

my $sock = IO::Socket::SSL->new('example.com:443') or die "ERROR::$@";
my @html;
for ( ( 1 .. 1000 ) ) {
    get();
}

sub get {
    print $sock "GET / HTTP/1.1\
```

```
        nHost: example.com\
        nConnection: keep-alive\n\
        n";
    my $html = "";
    while (<$sock>) {
        +next if /\r\n$/;
        $html .= $_;
        last if /<\/body>/;
    }
    push @html, $html;
}
```

Прогоняем:

```
$ time ./socket.pl

real 2m22.407s
user 0m2.635s
sys 0m0.306s
```

Далее пишем для осталных модулей.

Для YADA:

```
#!/usr/bin/env perl

use strict;
use warnings;
use YADA;

my @url;
$url[$_] = "https://example.com/$_" for ( ( 0 .. 999 ) );  # урлы для YADA должны быть разные

my @html;

YADA->new->append( [@url] => sub { push @html, ${ $_[0]->data }; } )->wait;
```

Прогоняем:

```
$ time ./yada.pl

```

```
real 0m37.593s
user 0m8.100s
sys 0m0.405s
```

О! Уже гораздо лучше, вместо двух минут всего 37 секунд, и код гораздо компактней.

Для AnyEvent::HTTP:

```
#!/usr/bin/env perl

use strict;
use warnings;
use AnyEvent;
use AnyEvent::HTTP;

my @url;
$url[$_] = "https://example.com"
    for ( ( 0 .. 999 ) );

my @html;
my $cv = AnyEvent->condvar;

```

```
14 for my $url (@url) {
15     $cv->begin;
16     http_get(
17         $url,
18         sub {
19             push @html, $_[0];
20             $cv->end;
21         }
22     );
23 }
24
25 $cv->wait;
```

Результат:

```
1 $ time ./anyevent.pl
2
3 real 0m35.149s
4 user 0m1.215s
5 sys 0m0.179s
```

Почти так же, но кода больше.

Для WWW::Curl:

```
#!/usr/bin/env perl

use strict;
use warnings;
use WWW::Curl::Easy;
use WWW::Curl::Multi;

my $url = 'https://example.com';

my $curlm = WWW::Curl::Multi->new
    ;
my %html;
my %easy;
my $active_handles = 1000;

for my $i ( ( 1 .. 1000 ) ) {

    my $curl = WWW::Curl::Easy->
       new;
    $easy{$i} = $curl;

```

```
20      $curl->setopt(
           CURLOPT_PRIVATE,   $i );
21      $curl->setopt( CURLOPT_URL,
                $url );
22      $curl->setopt(
           CURLOPT_WRITEDATA, \$html{
           $i} );
23      $curlm->add_handle($curl);
24  }
25  while ($active_handles) {
26      my $active_transfers = $curlm
           ->perform;
27      if ( $active_transfers !=
           $active_handles ) {
28          while ( my ( $id,
                $return_value ) =
                $curlm->info_read ) {
29              if ($id) {
30                  $active_handles
                       --;
31                  delete $easy{$id
                       };
32              }
```

```
        }
    }
}
```

Результат

```
$ time ./curl.pl

real 1m48.848s
user 0m53.507s
sys 0m2.206s
```

Может я как-то криво написал (я писал по примеру на CPAN), но получилось втрое медленнее, однако это все равно почти в два раза быстрее синхронного варианта с `IO::Socket::SSL`.

Для `Mojo::UserAgent`:

```
#!/usr/bin/env perl

use strict;
```

```
use warnings;
use Mojo::UserAgent;

my $ua = Mojo::UserAgent->new();
my @url;
$url[$_] = "https://example.com/
    $_" for ( ( 0 .. 999 ) );

my @html;
Mojo::IOLoop->delay(
    sub {
        my $delay = shift;
        for my $url (@url) {
            $ua->get( $url =>
                $delay->begin );
        }
    },
    sub {
        my ($delay) = shift;
        for my $r (@_) {
            push @html, $r->res->
                body;
        }
```

```
    }
)->wait;
```

Прогоняем:

```
$ time ./mojo_async.pl

real 0m11.177s
user 0m3.198s
sys 0m0.727s
```

Ого! В три раза быстрее, чем с AnyEvent::HTTP и YADA (я все примеры прогонял по 5 раз, разброс результатов был в пределах 1-2 секунд). Теперь посмотрим, что с бенчмарком и потреблением памяти.

```
#!/usr/bin/env perl

use strict;
use warnings;
use Memchmark;
use Benchmark;
```

```
use AnyEvent;
use AnyEvent::HTTP;
use Mojo::UserAgent;
use YADA;
use WWW::Curl::Easy;
use WWW::Curl::Multi;

my @url;
$url[$_] = "https://example.com/
    $_" for ( ( 0 .. 999 ) );

my $html;

my $url = 'https://example.com';

my %cmp_hash = (
    'mojo' => sub {
        my $ua = Mojo::UserAgent
            ->new();
        Mojo::IOLoop->delay(
            sub {
                my $delay = shift
                    ;
```

```
27                  for my $url (@url
                        ) {
28                      $ua->get(
                            $url =>
                            $delay->
                            begin );
29                  }
30              },
31              sub {
32                  my ($delay) =
                        shift;
33                  for my $r (@_) {
34                      $html = $r->
                            res->body;
35                  }
36              }
37          )->wait;
38      },
39      'anyevent' => sub {
40          my $cv = AnyEvent->
                condvar;
41
42          for my $url (@url) {
```

```
            $cv->begin;
            http_get(
                $url,
                sub {
                    $html = $_
                        [0];
                    $cv->end;
                }
            );
        }
        $cv->wait;
    },
    'curl' => sub {
        my $curlm = WWW::Curl::
            Multi->new;
        my $html;
        my %easy;
        my $active_handles =
            1000;

        for my $i ( ( 1 ..
            $active_handles ) ) {
             my $curl = WWW::Curl
```

```
            ::Easy->new;
62          $easy{$i} = $curl;
63
64          $curl->setopt(
               CURLOPT_PRIVATE,
                 $i );
65          $curl->setopt(
               CURLOPT_URL,
                     $url );
66          $curl->setopt(
               CURLOPT_WRITEDATA,
                \$html );
67          $curlm->add_handle(
               $curl);
68      }
69      while ($active_handles) {
70          my $active_transfers
               = $curlm->perform;
71          if (
               $active_transfers
               != $active_handles
                ) {
72              while ( my ( $id,
```

```
                        $return_value
                        ) = $curlm->
                       info_read ) {
                        if ($id) {
                            $active_hand
                               --;
                            delete
                               $easy{
                               $id};
                        }
                    }
                }
            }
        },
        'yada' => sub {
            YADA->new->append( [@url]
                => sub { $html = ${
               $_[0]->data }; } )->
               wait;
        },
);

Memchmark::cmpthese(%cmp_hash);
```

```
87 Benchmark::cmpthese( 100, \%
       cmp_hash );
```

Результат:

```
1  $ time ./benchmark_async.pl
2  test: anyevent, memory used:
       8114176 bytes
3  test: curl, memory used:
       2878337024 bytes
4  test: mojo, memory used: 17453056
       bytes
5  test: yada, memory used: 18440192
       bytes
6              s/iter     curl     yada
                      mojo anyevent
7  curl          62.2       --      -86%
           -97%     -97%
8  yada          8.48      634%       --
           -77%     -80%
9  mojo          1.99     3030%      327%
            --     -13%
10 anyevent      1.73     3488%      389%
```

```
                15%          ——

real    301m51.671s
user    121m48.920s
sys     4m14.271s
```

Здесь я ограничился всего 100 итерациями, потому как 1000 выполнялись более суток, и я не дождался результата. В итоге 400 000 запросов пролетели за 301 минуту, и как видно, больше всех потребляет память вариант с curl, а меньше всех AnyEvent, причем вариант с Mojo в 2 раза больше чем AnyEvent. По скорости же Mojo и AnyEvent выполняются почти одинаково, Mojo даже чуток уступает AnyEvent. Однако я запустил скрипты для Mojo и AnyEvent на 100 000 урлов вместо 1000, и вот что получилось:

```
$ time ./anyevent.pl

```

```
real    59m3.158s
user    6m8.883s
sys     0m28.461s

$ time ./mojo_async.pl

real    1m20.387s
user    1m11.533s
sys     0m2.759s
```

Разница огромная. Отсюда предполагаю что у Mojo медленно создается событийная петля, но запросы идут гораздо быстрей, хотя и потребляют гораздо больше памяти.

В следующей статье попробую разобраться с парсерами.

■ *Тигран Оганесян*

4 Perl 6 для веба

О том, как на Perl 6 уже сегодня поднять простой работающий веб-сервер

Новости

Пара новостей, прежде чем перейти к основной теме.

Во-первых, открылся замечательный сайт perl6intro.com с хорошо структурированным описанием Perl 6. Заодно напомню о существовании репозитория perl6-examples на гитхабе.

Во-вторых, только что вышла новая версия Rakudo: 2015.11.

Чтобы заработали примеры из этой статьи, надо обязательно обновиться, потому что с предыдущей версией ничего работать не будет: можно довести до ума, правя входящие в поставку файлы и обновив вручную пару модулей, но проще взять и поставить 2015.11:

```
perl Configure.pl --backend=moar
    --gen-moar
make
make install
```

Panda

В дистрибутиве Rakudo Star идет довольно большое число модулей и программа для их установки Panda. На сегодня вместо CPAN6 используется понятие «экосистемы», и Panda умеет правильно

устанавливать модули, если они были оформлены по соответствующим правилам экосистемы.

После сборки Ракудо исполнимый файл `panda` окажется в каталоге `install/bin`.

Для установки модуля теперь достаточно выполнить команду `panda install Module::X`. Модули, идущие с дистрибутивом, находятся в каталоге `install/share/perl6/lib`, а установленные пандой — в `install/share/perl6/site`.

HTTP-сервер

В дистрибутиве Ракудо и на сайте modules.perl6 есть несколько модулей, на основе которых можно построить свой веб-сервер (в том

числе с веб-сокетами) или веб-клиент. Интереса заслуживают, прежде всего, `HTTP::Easy`, `HTTP::Easy::PSGI`, серия модулей `Web::App` и фреймворк `Crust`. Еще обратите внимание на мини-фреймворк `[wee6](https://github.com/vti/wee6)`. Я же сейчас коснусь только фреймворка `Bailador`.

Bailador

Это реализация с интерфейсом, максимально приближенным к `Dancer` из Perl 5. Даже название такое же — *танцор* по-испански. Файл с приложением Bailador запускает PSGI-сервер, который сразу готов принимать запросы на дефолтном порту.

Минимальный «Привет, мир!» выглядит

так:

```
use Bailador;

get '/' => sub {
    'Hello, world!'
}

baile;
```

Запуск:

```
$ perl6 1.pl
Entering the development dance
    floor: http://0.0.0.0:3000
[2015-11-29T23:26:13Z] Started
    HTTP server.
```

Сервер заработал, можно посмотреть, что он отвечает, зайдя по адресу `http://0.0.0.0:3000`.

Параметры запросов можно брать сразу из

урла:

```
get '/:name' => sub ($name) {
    "Hello, $name!"
}
```

Обратите внимание, что между sub и скобкой нужен пробел. Если его не поставить, программа не скомпилируется:

```
===SORRY!=== Error while
    compiling /Users/ash/test/
    perl6/bailador/2.pl
Variable '$name' is not declared
at /Users/ash/test/perl6/bailador
    /2.pl:3
——————>     get '/:name' => sub□
    $name) {
```

Bailador с удовольствием принимает регулярные выражения:

```
get / 'square-of/' (<digit>+) /
    => sub ($n) {
```

```
    $n * $n
}
```

Этот блок будет обрабатывать адреса типа `http://0.0.0.0:3000/square-of/5`.

Для чтения переменных окружения и параметров GET- или POST-запроса можно обратиться к методу `request`:

```
get '/ua' => sub {
    request.env<HTTP_USER_AGENT>
    ~ '<br />' ~
    request.env<QUERY_STRING>
}
```

Наконец, простейший пример с выдачей результата на основе шаблона:

```
use Bailador;

get '/form' => sub {
    template 'test.tt';
```

```
}

baile;
```

Файл шаблона нужно поместить в подкаталог `views`.

Упс, не работает:

```
Use of uninitialized value
    $_location of type Any in
    string context
Any of .^name, .perl, .gist, or .
    say can stringify undefined
    things, if needed.
in method template at /Users/ash/
    perl6/rakudo-star-2015.11/
    install/share/perl6/lib/
    Bailador/App.pm:14
Failed to open file /views/t.tt:
    no such file or directory
```

Хотя в этом сообщении об ошибке и указан файл, в котором что-то не так, изменить его не так легко. При сборке Ракудо модули дополнительно компилируются в файлы `.moarvm` и как-то хитро связываются между собой. Поэтому проще полностью удалить танцорские файлы и каталоги `rm –rf install/share/perl6/lib/Bailador*` и установить его заново:

```
panda install Bailador
```

После чего исправить ошибку в файле `install/share/perl6/site/lib/Bailador/App.pm`:

```
– my $_location;
+ my $_location = '.';
```

Теперь запрос `http://0.0.0.0:3000/form` выводит содержимое файла `views/test.tt`.

Шаблон может принимать хеш с параметрами:

```
get '/form/:name' => sub ($name) {
    template 'name.tt', {name => $name}
}
```

Сам шаблон:

```
% my ($params) = @_;

Hi, <%= $params<name> %>!
```

DBIish

Для работы с базой данных удобно воспользоваться модулем `DBIish`, который просто берет и работает (если на компьютере есть `libmysqlclient`).

```
use DBIish;

my $dbh = DBIish.connect(
    'mysql',
    :host<example.com>,
    :port(3306),
    :database<dbname>,
    :user<username>,
    :password<password>
);

my $sth = $dbh.prepare("select
   now()");
$sth.execute();

my $arr = $sth.fetchall_arrayref
   ();
for $arr -> $x {
    say $x;
}

$sth.finish;
$dbh.disconnect;
```

Логика работы с `DBIish` аналогична стандартному `DBI` с поправкой на синтаксис Perl 6.

■ *Андрей Шитов*

5 Обзор CPAN за октябрь 2015 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц.

Статистика

- Новых дистрибутивов — 182
- Новых выпусков — 791

Новые модули

PEF::CacheLRU

`PEF::CacheLRU` — это реализация алгоритма кеширования LRU (вытеснение давно

неиспользуемых ключей) на чистом Perl. В описании модуля приводится сравнение производительности с реализацией `Cache::LRU`, в котором `PEF::CacheLRU` оказывается существенно более быстрым.

Term::Choose_HAE

С помощью модуля `Term::Choose_HAE` можно создавать консольные интерактивные меню для выбора одного или нескольких значений из списка. В отличие от `Term::Choose`, данный модуль не удаляет экранирующие последовательности ANSI, что позволяет использовать цвета для элементов меню. Также появилась опция `fill_up`, которая задаёт стиль закраски курсора.

Exporter::Attributes

`Exporter::Attributes` позволяет указывать какие символы модуля могут быть экспортированы или экспортируются по умолчанию с помощью атрибутов. Например, сравнение с Exporter:

```
package My::Module;                    |    package My::Module;
use Exporter;                          |    use Exporter::Attributes qw(
                                       |    import);
                                       |
our @EXPORT = qw(foo bar);             |    sub foo : Exported;
                                       |
                                            sub bar :
```

```
                                    Expo
                                    ;
6                                   |
7 our @EXPORT_OK = qw(baz);         |
     sub baz : Exportable(qux);
8 our %EXPORT_TAGS =                |
9      qw(qux => [qw(baz)]);        |
```

Модуль является аналогом `Exporter::Simple`, который уже давно заброшен и не работает на современных версиях Perl.

Devel::Unstate

Модуль `Devel::Unstate` позволяет превратить state-переменные в my. Это может быть полезно при тестировании, когда требуется, чтобы данные в кеше обновлялись при каждой итерации теста. Эффект действия `Devel::Unstate` глобальный, но

затронуты будут только те переменные, которые были объявлены после загрузки модуля.

IPC::Lockfile

`IPC::Lockfile` — это реализация классического способа обеспечения запуска только одного экземпляра текущей программы путём установки лока на исходный код программы:

```
1 open SELF, "< $0" or die ...;
2 flock SELF, LOCK_EX | LOCK_NB  or
      exit;
```

Вероятно, до сих пор ради двух строчек кода никто больше не рисковал создать целый модуль.

Perlito5

Впервые на CPAN выложен компилятор Perl 5, написанный на Perl 5. Компилятор позволяет также транслировать Perl 5 код в любой из поддерживаемых бекендов: js, perl5, perl6, xs, java. Таким образом, существует возможность для бутстрапа самого Perlito5, например в js-код, тем самым позволив запускать Perl 5 программы на node.js и, вероятно, и в браузере:

```
$ perlito5 --bootstrapping -Cjs \
    `which perlito5` > perlito5.
       js

$ node perlito5.js -e ' print "
   hello, world!\n" '
hello, world!
```

Acme::Test::VW

Скандал вокруг дизельных двигателей Volkswagen оставил свой след и в фольклоре программистов. В октябре стали появляться реализации модулей, которые при детектировании выполнения тестов внутри автоматизированных систем CI/QA делают все падающие тесты успешными. Для Perl был создан модуль `Acme::Test::VW`, который при запуске тестов в CPAN Testers, Jenkins, Travis CI и других системах всегда будет давать успешный результат.

```
# export PERL5OPT=-MAcme::Test::VW

use Test::More;
ok 1 == 2;
done_testing;
```

Acme::Excuse

В отличие от `Acme::Test::VW`, `Acme::Excuse` не пытается скрыть наличие ошибок в вашем коде, но зато пытается найти оправдание им. В случае, если происходит фатальная ошибка, `Acme::Excuse` загружает и выводит сообщение с сайта www.programmerexcuses.com:

```
$ perl -MAcme::Excuse -e 'use Perl or die'
Well done, you found my easter egg!
```

App::Mimic

Если же так получилось, что в вашем коде нет багов, то их можно добавить, причём так, что найти их будет совсем непросто.

Утилита `mimic` может внести случайные изменения в исходный код программы, выполняя замену ASCII-символов на схожие Юникод омоглифы, например:

- ; U+003B SEMICOLON
- ; U+037E GREEK QUESTION MARK
- ﬔ U+FE54 SMALL SEMICOLON
- ； U+FF1B FULLWIDTH SEMICOLON
- ﹔ U+FE14 PRESENTATION FORM FOR VERTICAL SEMICOLON

Выводимые сообщения об ошибках могут поставить в тупик любого перловика.

Обновлённые модули

Test::Stream 1.302021

Преемник `Test::More` и `Test::Builder` модуль `Test::Steam` теперь больше не имеет статуса экспериментального, что вполне можно интерпретировать, как сигнал к началу использования. Также если заявленный грант для Test::Stream будет принят, то это в скором времени приведёт к созданию подробного мануала по использованию модуля.

Gazelle 0.36

В новой версии высокопроизводительного веб-сервера Gazelle появилась экспери-

ментальная поддержка FreeBSD, а также исправлена проверка наличия системного вызова `accept4`.

EV::ADNS 3.0

Вышел новый мажорный релиз модуля `EV::ADNS` для выполнения асинхронных DNS-запросов с помощью библиотеки adns и EV. Появилась поддержка ipv6, произошёл переход на libev 4 API, реализован вызов `EV::ADNS::reinit` для сброса всех выполняющихся запросов и реинициализации библиотеки adns.

Data::Alias 1.20

Обновлён модуль `Data::Alias`, предоставляющий набор подпрограмм для выполнения операций с алиасами (псевдонимами). Новая версия теперь собирается и работает на Perl 5.22, но в документации теперь указывается, что появившиеся в Perl 5.22 средства по созданию алиасов работают гораздо надёжнее и лучше сопровождаются, хотя их синтаксис отличается, и функционал не так богат. Поэтому если вы используете последние версии Perl в разработке — есть смысл отказаться от использования `Data::Alias`.

BSD::Resource 1.2908

В новой версии BSD::Resource добавлено множество новых значений `RLIMIT` для различных систем и особенно linux 2.6: `RLIMIT_PTHREAD`, `RLIMIT_RTPRIO`, `RLIMIT_RTTIME` и другие. Поддерживается расширение `PRIO_THREAD`, если оно доступно в системе.

DBD::mysql 4.033

В новой версии драйвера СУБД MySQL `DBD::mysql` исправлено несколько ошибок, включая утечку памяти в `$sth->{ ParamValues}`, патч для которой два с половиной года ждал своего часа в RT. Также внесены изменения для совместимости с последним релизом MySQL 5.7.9.

Minilla v3.0.0

Вышел новый мажорный релиз утилиты Minilla для подготовки дистрибутивов для CPAN. В новой версии совсем немного изменений, поэтому релиз скорее всего просто фиксирует стабильное состояние.

Search::Elasticsearch 2.00

Выпущен новый мажорный релиз официального клиента Elasticsearch. В данной версии по умолчанию используется API `2_0::Direct`, но по-прежнему поддерживаются предыдущие версии 1.0 и 0.90.

Devel::MAT 0.21

Обновлён модуль `Devel::MAT` для анализа использования памяти Perl-программой. В новой версии обеспечена совместимость с Perl 5.22, а также обновлён формат хранения дампов.

■ *Владимир Леттиев*

6 Интервью с Рикардо Мартинесом

Рикардо Мартинес — Perl-программист, работает над devops-сервисами для крупных клиентов из бизнес-сектора

Как и когда научился программировать?

Мое первое знакомство с программированием было в доме моего друга в 1982 году, когда мне было 12 лет. Его старший брат купил подержанный Sinclair ZX-81 и, подключив его к телевизору, я был поражен той “мощью”, которой обладали несколько строк на BASIC. Помню, мы провели все выходные программируя дурацкие алгоритмы по типу напечатать все нечетные числа от 1 до 1000 и т.п. Единственной документацией к BASIC был стостраничный мануал к этой чертовой машине без

клавиш.

После этого я ходил по дому и выпрашивал купить мне одну из этих машин. Через несколько месяцев Санта принес мне Sinclair ZX-Spectrum. И у него были клавиши!!! Резиновые клавиши, но все же клавиши. С этим компьютером я проводил часы пытаясь понять секрет этих таинственных команд PEEK и POKE, которые я увидел в примерах, пока у меня не получилось нарисовать иконку на экране и перемещать ее нажатием определенных клавиш. Круто!!!!!

Через несколько лет мой отец купил первый PC, это был Digital XXXX с 5,25“-дискетами и операционной системой CPM, и я там нашел BASIC-интерпретатор!! и продолжил обучение. Позже у меня появился СВОЙ PC 8088 и я научился про-

граммировать bat-файлы и затем у меня появились Windows 1.0 и Access 1.0 и было круто работать со структурированными данными в базе данных. Затем я поступил в университет и познакомился со многими новыми друзьями: Fortran, Pascal, Modula-2, Cobol, C, ADA, LISP, Assembler для Motorola-68000, Informix SQL… ух!

Моей первой работой было программирование на Cobol для MVS, с IMS/DB и IMS/DC и позже DB2 и CICS.

После этого я пробовал много технологий и языков программирования (Cobol/400, RPG/400, Java, VB, VBScript, Javascript, CGI Perl, …). Последние 15 лет я в основном использовал Java, Perl и Javascript.

Какой редактор используешь?

Все зависело от платформы и языка. Я начал использовать vi в университете, но на первой работе были в основном ISPF и SPF/400. Когда я начал работать с Java и Perl, я использовал eclipse с Epic. Когда мы оставили Java позади, я некоторое время пользовался Notepad++, но вскоре вернулся на vim. Четыре года назад я наткнулся на SublimeText, и он стал основным моим редактором для программирования.

Когда и как познакомился с Perl?

В конце 90-х я написал несколько CGI-программ на Perl для листинга содержимого директорий и других базовых штук. Когда мы начали разрабатывать систему автоматизации развертывания приложений в начале 2000-х мы решили, что Perl идеально подходит под наши задачи: парсинг файлов, закачка файлов по фтп на

серверы, выполнение системных команд, и т.д. и все это мультиплатформенно. Так мы разработали множество perl-скриптов в виде User Defined Processes (UDPs) из CA Harvest. С одного проекта к другому мы все больше учили Perl и находили ему все больше применений в своих имплементациях. Одним из ключевых моментов было решение отказаться от Java servlet для пакетного запуска процессов и переписать все на Perl, так чтобы отвязать наше приложение от java-сервера для выполнения фоновых задач. Время выполнения заметно улучшилось, у нас появилась гибкость и простота в сопровождении. Позже мы решили переписать все как MVC-приложение на Perl (Catalyst) и ExtJS для интерфейса.

С какими другими языками интересно работать?

Мне было интересно работать с каждым языком программирования с которым пришлось столкнуться, но в течение лет я обнаружил, что мне больше нравится работать с динамическими языками, чем с другими. Мне нравятся PHP и Ruby, также привлекает новая платформа Node.js и мне бы хотелось углубить свои знания в будущем.

Что, по-твоему, является самым большим преимуществом Perl?

Я изучал Perl в течение многих лет, но не считаю себя экспертом, больше средним программистом. У меня всегда получалось находить простой путь для реализации любого фунционала используя конкретные CPAN-модули или их комбинации. Любую проблему с которой я сталкивался в Perl у меня получалось решать поиском на

perlmonks или stackoverflow. И если у меня не получалось найти описанное решение (очень редко) я находил ответ довольно быстро. По-моему, Perl самый мощный язык программирования из-за огромного сообщества его поддерживающего и огромного числа хорошо документированных расширений, которые можно легко установить и использовать.

Что, по-твоему, является самой важной особенностью языков будущего?

Языки будущего должны быть адаптированы к новым особенностям железа в мире виртуализации и контейнеров, когда инсталляция новой машины занимает несколько кликов. Очень мало функционала реализовано в языках программирования для параллельных вычислений для разных платформ. Мне хочется еще

быть живым, чтобы увидеть как первый CICS будет успешно заменен какой-нибудь платформой для параллельной обработки.

Где сейчас работаешь? Сколько времени проводишь за написанием Perl-кода?

Clarive Software Inc.. Большинство моей работы за последние 15 лет было посвящено реализации функционала для devops и простой доставки приложений для компаний, которые хотят контролировать свой цикл разработки от генерации RFC до развертывания в производстве, сохраняя всю активность выполненную всеми ролями в IT-компании и автоматизируя все повторяющиеся и моделируемые задачи.

Что такое на самом деле devops?

Большинство людей, когда говорят о

devops, имеют в виду множество утилит (или наборт инструментов) для автоматизации действий, которые выполняются обычно командами операторов, самими разработчиками. Но devops идет намного дальше. Devops это слияние и эволюция таких концепций как Непрерывная интеграция (Continuous Integration), Непрерывное внедрение (Continuous Deployment), Непрерывная поставка (Continuous Delivery) или Бережливая поставка приложений (Lean Application Delivery). Компании, которые разрабатывают свой софт, обычно используют архаичные методологии разработки такие как “водопад” (“waterfall”) с монолитными длинными релизами. В последнее время задержка выхода на рынок стала наиболее важным индикатором в бизнесе, и его очень сложно улучшить используя эти старые методологии. Внедрение devops в первую очередь означает изменение в

IT-культуре, что представляет собой изменение процесса разработки, от требований до запуска. Devops означает сотрудничество между командами (Управление релизами, Разработка, Тестирование, Операции и т.п.), автоматизацию технических задач (коммуникация между утилитами, внедрение софта и конфигурации и т.п.), прослеживаемость цикла и мониторинг процесса. Компания, которая полностью внедрила devops, может сократить цикл релизов для соответствия рынку и кардинально улучшить время выхода на рынок, и в то же время контролируя, чтобы все требования качества соответствовали заявленным. Возможно, описание всех плюсов реализации devops для компании в современном миру могут занять несколько страниц, но их можно обобщить в один рекламный слоган: “скорость под контролем”.

Важной частью devops реализации является понимание, что есть софт, который остается со старыми методологиями, в то время как новые технологии могут перейти на новые методологии. Сосуществование разных методологий и скоростей известно как бимодальные ИТ (BI-Modal IT) и в большинстве случаев является самой большой проблемой компании на пути внедрения devops.

Если бы смог начать свой проект с нуля, снова бы выбрал Perl?

Определенно я выбрал бы Perl. Я бы спасся от той головной боли, которую испытывал, когда java-сервер был единственным/лучшим кросс-платформенным решением для веб-интерфейсов для наших клиентов. Возможно, мы бы приняли несколько другие решения в архитектуре и выбрали

другие фреймворки и пользовательский интерфейс, но в основе был бы Perl.

Заботятся ли бизнес-клиенты о технологии, на которой реализованы их сервисы?

В большинстве случаев они спрашивают, но чисто из любопытства. Им свойственно чувствовать себя комфортнее с технологиями, которые они уже используют (в основном Java), но на самом деле им все равно, если поддержка и сопровождение корректны и им не приходится покупать какой-то другой софт, чтобы запустить наш.

На текущий момент мы в процессе добавления больше “стандартных” языков программирования для расширения нашей платформы (как, например, PHP или Ruby) так как тем, кто внедряет наш

продукт, проще найти программистов невысокого уровня в этих языках.

Что думаешь о тестировании приложений, где баланс между быстрой реализацией нового функционала и стабильной поддержкой?

К сожалению, я не слышал о TDD до недавнего времени и бы очень хотел начать программировать с тестированием. Сейчас (и я надеюсь, что не поздно) я начинаю понимать, что тестирование это очень важно для выпуска приложения с желаемым качеством. Отношение между скоростью реализации и усилиями при сопровождении меняется с экспоненциальной без тестирования (очень быстрая реализация в начале, но бесконечное сопровождение в будущем) на логарифмическую с тестированием (не так быстро понача-

лу в реализации, но очень стабильно в сопровождении).

Стоит ли советовать молодым людям учить Perl сейчас?

ДА!!! Perl отличный выбор для изучения, с ним очень легко начать, он очень хорошо документирован, очень мощный для дальнейшего программирования и безусловно останется в качестве стандарта для системного программирования и является отличным выбором для людей, которым нужна гибкость, современность и полнота в языке программирования.

■ *Вячеслав Тихановский*