Pragmatic Perl 8

pragmaticperl.com

Выпуск 8. Октябрь 2013

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

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

Авторы статей: Владимир Леттиев, Денис Федосеев

Корректор: Андрей Шитов

Выпускающий редактор: Вячеслав Тихановский (vti)

Ревизия: 2014-11-29 16:12

Оглавление

1. От редактора

В этом номере вместо Perl Quiz попробуем Perl Golf. Если вам это больше понравится, пожалуйста, скажите нам об этом в комментариях, по почте или через социальные сети.

Читайте также в номере интервью с chromatic, автором популярной и свободно доступной книги о современном Perl — Modern Perl.

Мы продолжаем искать авторов для следующих номеров. Если у вас есть идеи или желание помочь, пожалуйста, свяжитесь с нами.

Приятного чтения.

■ *Вячеслав Тихановский*

2. Разворачивание PSGI/Plack приложения

С появлением Plack у программистов пропала необходимость затачивать свое приложение под каждый из возможных вариантов развертывания приложений. Однако у администраторов никуда не пропала необходимость настроек сервера под это самое приложение. И если для опытного программиста/администратора этот вопрос не вызвает затруднений, то у новичков в вебе трудности вылезают вполне ощутимые.

Итак, вниманию благодарной публики представляется сборник рецептов по развертыванию минимального PSGI/Plack приложения на всех популярных видах хостингов.

В качестве тестового приложения мы будем использовать немножко измененное приложение, автоматически созданное фреймворком с названием, начинающимся на `M`. Тем, кому не нравится фреймворк на `M`, могут использовать фреймворк на `D`. или любой другой по своему вкусу с поддержкой `Plack`.

`Plack` предствляет собой стандартизированный интерфейс между сервером и приложением. В обычном случае приложению даже нет необходимости знать, в каком режиме и с каким сервером оно работает, что очень упрощает жизнь простому программисту.

Итак, поехали. Иходник приложения:

```
#!/usr/bin/env perl
use Mojolicious::Lite;

get '/' => sub {
  my $self = shift;
  $self->render('index');
};

get '/test' => sub {
    my $self = shift;
    $self->render('test');
};

app->start;
__DATA__
```

```
16
17 @@ index.html.ep
18 % layout 'default';
19 % title 'Welcome';
20 Welcome to the Super—Puper app!
21
22 @@ test.html.ep
23 % layout 'default';
24 % title 'Test';
25 Route test page!
26
27 @@ layouts/default.html.ep
28 <!DOCTYPE html>
29 <html>
30   <head><title><%= title %></title></head>
31   <body><%= content %></body>
32 </html>
```

Начнем от простого к сложному.

Внимание! Все конфигурации для серверов приведены в минимальном варианте. Без ненужных в данном случае настроек типа виртуальных хостов, защит от ботов, ддос, нападения инопланетян и пролития кофе на клавиатуру. В общем, за выкаченные без изменений на боевой сервер конфиги автор ответственности не несет.

Nginx reverse proxy

Здесь все просто, у вас есть собственный сервер/VDS, есть ваше приложение, запущенное любым удобным способом, и nginx просто прокидывает на него запрос пользователя.

```
1 $ ./test daemon
2 [Wed Oct  2 06:16:37 2013] [info] Listening at "http
    ://*:3000".
3 Server available at http://127.0.0.1:3000.
```

Лучше, конечно, использовать prefork-сервер типа `Starman`, но в нашем случае это не принципиально.

Запущенное `M.`-приложение по умолчанию запускается на 3000-м порту. Его-то мы и будем использовать по умолчанию везде, где

нам нужен порт в конфигурации сервера.

Создаем конфиг nginx:

```
server {
    listen *;

    location / {
        proxy_pass http://127.0.0.1:3000/;
        proxy_set_header   X-Real-IP $remote_addr;
        proxy_set_header   Host $http_host;
        proxy_set_header   X-Forwarded-For
           $proxy_add_x_forwarded_for;
    }

    # Передаем директорию для static файлов на обработку
       nginx
    location ~* ^.+\.(jpg|jpeg|gif|png|ico|css|pdf|ppt|
       txt|bmp|rtf|js)$ {
        root /path/to/your/app/public/; # Путь к папке
           для public файлов
        expires 3d; # кешируем у клиента на 3 дня
    }
}
```

Способ хорош своей простотой и масштабируемостью. Чтобы увеличить количество обрабатываемых коннектов, достаточно запустить приложение под `Starman` или любым другим сервером и получить практически линейный рост числа обрабатываемых клиентов (пока не упремся в мощьность сервера, конечно).

Минус этого способа — за приложением придется следить самому и писать дополнительную обвязку на случай перезагрузки сервера/-приложения.

Но здесь нам поможет `Ubic`. Устанавливаем из CPAN:

```
# cpan -i Ubic
# ubic-admin setup
```

Создаем минимальный файл конфигурации:

```
# vim /etc/ubic/service/site.ini

[options]
bin = /path/to/your/app/super_app.pl daemon
```

Стартуем сервис:

```
# ubic start site
```

И получаем работающий за nginx бекэнд. Если приложение внезапно упадет, `Ubic` запустит его обратно. Естественно, вместо *test daemon* лучше использовать какой-либо из специализированных серверов, например `Starman`.

Рассматривать способ деплоя через Nginx/FastCGI не имеет особого смысла т.к. отличия в конфигурации буквально в двух строчках. Но Nginx, в отличии от Apache, не умеет автоматически запускать запрошенное приложение. Так что здесь не обойтись без внешнего менеджера, и пропадает вообще какой-либо смысл использования FastCGI.

Перейдем к устаревшему варианту. Допустим, что выделенного сервера у нас нет, а есть shared-хостинг с Apache и поддержкой Perl.

Apache/CGI

Вариант, отличающийся простотой настройки, работоспособностью везде, где есть Apache и Perl, а так же самой высокой тормознутостью из всех возможных к использованию вариантов.

В Cookbook предлагается для этого случая прописать *ScriptAlias*

```
ScriptAlias / /home/sri/myapp/script/myapp/
```

Проблема этого метода в том, что эта директива не работает в *.htaccess*, а править файл конфигурации виртуального хоста вам никто не даст. С другой стороны, если у вас есть доступ к редактированию конфигурации виртуальных хостов, зачем настраивать CGI?

В общем мы, как настоящие герои, пойдем в обход и все настройки будем делать через `.htaccess`. В директории, где лежит сайт, создаем файл `.htaccess` со следующим содержимым:

```
Options +ExecCGI
```

```
AddHandler cgi-script .pl
DirectoryIndex index.pl
```

`index.pl` — это приложение, которое должно обрабатывать запрос. Внимательно проверьте атрибуты файла, по умолчанию `M.` создает файл с атрибутами 744, так что Apache не сможет его запустить и выдаст ошибку 500. Для нормальной работы надо сменить атрибуты на 755.

Заходим на сайт, проверяем — маршруты не работают. Для этого воспользуемся возможностями модуля `mod_rewrite`, таким образом Apache обучается делать то, что нам нужно.

Полный `.htaccess`:

```
CharsetDisable On

# Принудительно выставляем кодировку UTF-8 ибо 21 век на дворе!
AddDefaultCharset utf-8

# Разрешаем CGI
AddHandler cgi-script .pl
Options +ExecCGI

# Включаем mod_rewrite
RewriteEngine on

# Проверяем что запрошенный ресурс не реальный файл или директория
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l
RewriteCond %{REQUEST_FILENAME} !-d

# и передаем путь нашему обработчику
RewriteRule ^(.*)$ index.pl/$1 [L]
```

Все, на этом настройку Apache/CGI в целом можно считать законченной.

Apache/mod_perl

Более прогрессивный способ по сравнению с CGI, позволяет запускать приложение автоматически и выдает достаточно неплохую скорость. В минусах — доступен не везде, и есть небольшие заморочки с написанием кода под **mod_perl**. Так что, возможно, понадобится небольшая доработка приложения, если не повезет. Как минимум, для этого способа понадобится установить *Plack* из CPAN, т.к. нам понадобится *Plack::Handler::Apache2*.

Настройка `.htaccess`:

```
<Perl>
    $ENV{PLACK_ENV} = 'production';
    $ENV{MOJO_HOME} = '/var/www/index.pl';
    $ENV{MOJO_TEMPLATE_CACHE} = 0;
</Perl>
SetHandler perl-script
PerlHandler Plack::Handler::Apache2
PerlSetVar psgi_app /var/www/index.pl

# Включаем mod_rewrite
RewriteEngine on

# Проверяем, что запрошенный ресурс не реальный файл или директория
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l
RewriteCond %{REQUEST_FILENAME} !-d

# Передаем путь обработчику
RewriteRule ^(.*)$ index.pl/$1 [L]
```

Разворачиваем наше приложение и радуемся жизни.

Apache/FastCGI

В целом все аналогично предыдущим пунктам, за исключением необходимости небольшой доработки приложения. Строку

```
app->start;
```

необходимо заменить на:

```
app->start('fastcgi');
```

И все это сопроводить следующим `.htaccess` файлом:

```
# Объявляем хэндлер для FastCGI приложения
SetHandler fcgid-script
Options +ExecCGI

# Включаем mod_rewrite
RewriteEngine on

# Проверяем, что запрошенный ресурс не реальный файл или
   директория
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l
RewriteCond %{REQUEST_FILENAME} !-d

# Передаем путь обработчику
RewriteRule ^(.*)$ index.pl/$1 [L]

# Запрос без указания пути передаем обработчику
# принудительно, иначе получим 403
RewriteRule ^$ index.pl [L]
```

И напоследок рассмотрим способ развертывания через *Apache/mod_proxy*.

Apache/mod_proxy

Да, как это ни странно, Apache тоже умеет работать в режиме реверс-прокси. Этот режим примечателен тем, что позволяет использовать всю мощь модулей Apache и при этом обработку непосредственно приложения возложить на отдельный сервер. У меня работает конфигурация, в которой Apache осуществляет SSO-авторизацию пользователей через mod_kerberos и после этого направляет их на приложение, которое крутится под `Starman`. Позволяет убить двух зайцев сразу — в браузерах с поддержкой SSO пользователю нет необходимости вводить свой доменный логин/пароль, с другой стороны — мне нет необходимости заниматься вопросами аутентификации пользователей, если авторизация через Kerberos не пройдет — то Apache просто не пропустит пользователя до приложения. Сплош-

ная экономия, хотя для сайтов в интернете технология практически не применимая.

Итак, приступим к настройке, `.htaccess` нам здесь уже не помощник, будем править файл виртуального хоста:

```
<VirtualHost *:80>
    DocumentRoot /var/www/

    <Proxy *>
        Order deny,allow
        Allow from all
    </Proxy>

    ProxyVia On

    # Превращаем переменные Apache в переменные %ENV
    ProxyPassInterpolateEnv On
    ProxyRequests Off
    ProxyPreserveHost On
    ProxyPass / http://localhost:3000/ keepalive=On
    ProxyPassReverse / http://localhost:3000/
    RequestHeader set X-Forwarded-HTTPS "0"
</VirtualHost>
```

Конфиг не отличается затейливостью и есть в *Cookbook*, за одним исключением:

```
ProxyPassInterpolateEnv On
```

Эта опция заставляет Apache прокинуть все свои переменные в переменные **%ENV**. Сильно помогает, когда вы используете авторизацию через Apache, например.

Но просто так вся эта конструкция не заработает, нужно включить дополнительные модули:

```
# a2enmod proxy
# a2enmod headers
# a2enmod proxy_http
# service apache2 restart
```

На этом краткий практикум можно считать законченным. Он покрывает 95% процентов случаев развертывания приложений на всевозможных серверах, с которыми мне только приходилось сталки-

ваться за всю свою трудовую деятельность. Оставшиеся 5% обычно представляли собой особо экзотичные настройки окружения shared-хостингов и побеждались допиливанием напильником вышепреведенных хостингов или техподдержки хостинг-провайдеров.

■ *Денис Федосеев*

3. Консольные приложения на Curses

Приложения с текстовым интерфейсом, работающие в терминале, по-прежнему очень популярны и отлично конкурируют с приложениями с графическим интерфейсом. `Mutt`, `irssi`, `vim`, `tmux` *и многие другие являются незаменимыми в повседневной работе. На CPAN есть модули, позволяющие создавать приложения с текстовым интерфейсом, в том числе модуль* `Curses`*, являющийся обвязкой к распространённой C-библиотеке* `ncurses`*.*

История

Для самых ранних ЭВМ в качестве ввода/вывода информации использовались электромеханические телетайпы (*Teletype*, или сокращённо *TTY*), которые позволяли вводить текст с клавиатуры и печатать на бумаге символ за символом, полученные от ЭВМ. Позже телетайп был заменён терминалом, где принтер заменил ЭЛТ-экран, на который на фиксированные позиции можно было выводить символы. Самым первым компьютерным терминалом стал *DataPoint 3300*, который имел экран, позволяющий выводить 25 рядов по 72 символа. Примечательно, что этот терминал был создан на основе обычных TTL-микросхем, а дальнейшие попытки упростить логику и уменьшить размер внутренней начинки терминала стали одной из побуждающих причин разработки микропроцессоров.

Как правило, к одному мейнфрейму под управлением какой-либо из UNIX-систем подключалось множество терминалов. ЭВМ запускала программу командной оболочки, ввод и вывод которой был связан с устройством терминала. Оболочка обрабатывала вводимые команды и выводила результаты их работы на терминал.

Стандартом де-факто на долгие времена стал терминал *VT100*, созданный компанией DEC в 1978 г. Терминал подключался к вычислительной машине через последовательный интерфейс, позволял выводить 80 или 132 символа в ряд и использовал для кодировки символов стандарт *ASCII*, а также набор управляющих последовательностей (escape-последовательностей), которые были стандарти-

зованы *ANSI* в виде стандарта *ECMA-48*. Escape-последовательности получили своё название от кода клавиши *Escape*, который предварял набор кодов, определяющих управляющую последовательность. Управляющие последовательности позволяли передвигать курсор, очищать экран, задавать толщину символов или даже делать мигающую строку, на цветных терминалах появилась возможность задавать цвет фона и символа. Например, последовательность кодов `ESC[1m` задаёт жирный шрифт для последующих выводимых символов, а `ESC[0m` сбрасывает все установленные атрибуты.

Возможность манипулировать расположением символов и их атрибутами на экране открыло путь к созданию программ с текстовым пользовательским интерфейсом. В отличие от командного интерфейса, в текстовом интерфейсе с помощью `ASCII`-символов можно было отображать границы окон, диалоговые окна, меню или таблицы, а также кнопки и поля ввода.

С ростом числа терминалов стала возникать проблема с поддержкой всех их видов внутри программ. В 1978 г. Билл Джой, при работе над текстовым радактором *vi* для *Berkeley Unix*, выделил код для поддержки разных типов терминалов в отдельную библиотеку под названием *termcap*, которая стала базой данных описаний существующих терминалов и позволяла реализовывать программы, независимые от типа терминала. Termcap давал возможность программе узнать ширину терминала, правильную последовательность escape-кодов для перемещения курсора и т.д. Вслед за termcap появилась библиотека *terminfo*, которая являлась улучшенной реализацией termcap с более быстрым доступом к описанию терминала. Именно terminfo получила максимальное распространение в UNIX-системах.

В 1980 г. большую популярность получила игра Rogue, которая дала старт целому жанру rogue-подобных игр, в которой игровой персонаж исследует подземелья, сражается с монстрами и ищет сокровища. Игра для вывода использовала текстовую консоль, монстры обозначались заглавными буквами, коридоры и границы подземелий прорисовывались ASCII-символами `|` и `–`. Один из разработчиков, Кен Арнольд, специально для игры создал библиотеку, которая абстрагировалась от работы с конкретным типом терминала, вводила абстрактные понятия окон как матриц символов, забирая на себя

все заботы по выводу и обновлению экрана. Библиотека получила название *curses* («проклятие»), что является каламбуром на словосочетание «cursor optimization». Первоначально curses была основана на termcap. В отличие от termcap, которая фактически являлась текстовой базой данных о свойствах терминалов, curses предоставляла довольно внятное C API.

Позже появились библиотеки *pcurses* и *PDcurses*, которые являлись свободными альтернативами *BSD curses*. В 1991 г. работа над pcurses была продолжена, а в 1993 г. она была переименована в *ncurses*, что было сокращением от *new curses* (новая curses), и стала развиваться в рамках проекта GNU. Со временем к ncurses появились обертки для других языков программирования, в том числе и Perl-модуль Curses, который позволяет использовать библиотеку в Perl-программах.

Широкое использование аппаратных терминалов прекратилось после появления видеодисплеев, но тем не менее простота интерфейса и количество существующих для него программ привело к тому, что были созданы эмуляторы терминала — программы, которые эмулировали видеотерминал внутри другой видео системы, например, *X Window*. Это позволяло продолжать использовать программы, заточенные для работы в терминале, и в других системах. Наиболее популярные графические эмуляторы терминала — *xterm*, *rxvt*, а также современные *gnome-terminal* и *konsole*.

Использование консольных приложений с текстовым интерфейсом актуально и на сегодняшний день. Такие приложения как *top*, *vim*, *emacs*, *mutt*, *irssi*, *moc*, *midnight commander* используют многие и просто не представляют себе другой альтернативы. Терминальное приложение может работать поверх последовательных линий и поверх сети на очень низких скоростях, которые недоступны *VNC* или *RDP*. А в отличие от веб-интерфейсов, не требуют от клиента наличия гигабайтов оперативной памяти и многоядерных процессоров для того, чтобы удовлетворить системным требованиям современных браузеров.

Устройство терминала

Поскольку исторически терминалы являлись внешними устройствами с посимвольным обменом информации по последовательным линиям, то в UNIX-системах каждому терминалу соответствовал файл устройства вида `/dev/tty*`. Системные консоли были доступны на `/dev/tty[0..NN]`, последовательные порты на `/dev/ttyS[0..NN]` и т.д. (вариация названия зависела от конкретной реализации UNIX-системы). Для каждого такого устройства существует возможность установки скорости обмена, управляющих последовательностей и других настроек. Текущие настройки любого такого устройства можно увидеть с помощью утилиты `stty`:

```
# stty -a --file=/dev/tty0
speed 38400 baud; rows 64; columns 160; line = 0;
...
```

Для процессов, которым необходима работа с терминалом, не связанным с каким-то физическим устройством, создаются так называемые псевдотерминалы. В современных UNIX-системах для эмулятора терминала по запросу создаётся файл устройства псевдотерминала в файловой системе *devpts*, например `/dev/pts/0`, с которым ассоциируются стандартные файловые дескрипторы ввода/вывода `STDIN`, `STDOUT` и `STDERR` запускаемых эмулятором дочерних процессов. Эмулятор терминала определяет, каким будет тип терминала, каков будет размер экрана (количество строк и символов в строке) и другие параметры. Для запущеного же в псевдотерминале приложения нет никакого отличия работает ли он в псевдотерминале или на любом другом реальном терминале.

Perl Curses

Perl-модуль `Curses` является низкоуровневой обвязкой к C-библиотеке ncurses. Из существующей POD-документации практически невозможно понять, как программировать приложения на `Curses`, поскольку подразумевается, что вы будете использовать документацию по самой C-библиотеке ncurses.

Помимо `Curses`, на CPAN существуют ещё несколько дистрибутивов, которые позволяют разрабатывать приложения с текстовым интерфейсом. Часть их используют `Curses`, но имеют более высокоуровневый интерфейс, удобный для разработки. Это такие модули как `Curses::UI` и `Curses::Toolkit`. Есть также модули, которые вообще не привязаны к ncurses, например, `Tickit`. Если вы планируете создание сложных консольных приложений, где будут активно использоваться примитивы окон, диалогов, меню, форм, функции обратного вызова на события, то эти модули, вероятно, наиболее лучший выбор для использования. Если же стоит задача создания тривиального позиционного вывода информации, как, например, в приложении `top`, то использование `Curses` может быть целесообразнее.

Библиотека `Curses` работает с экраном, который представляется в виде прямоугольной матрицы, каждый элемент которой связан с позицией на экране для вывода символа. Начало координат, позиция *(0,0)*, задаёт левый верхний угол экрана. На экране могут быть заданы одно или несколько окон и субокон — прямоугольных областей, для которых определены свои свойства вывода.

Старт с Curses

Несколько функций библиотеки инициализируют работу приложения в текстовом режиме:

- `initscr` — функция инициирует режим curses, создаёт окно по умолчанию, производит очистку экрана и устанавливает текущую позицию курсора в левый верхний угол (0,0);
- `cbreak` и `raw` — функции отключают буферизованный ввод, и приложение сразу получает информацию о нажатых клавишах, при этом `raw` также перехватывает и специальные сочетания клавиш, такие как `Ctrl+C` и `Ctrl+Z`;
- `echo` и `noecho` — включают и отключают режим «эха», т.е. вывод на экран вводимых с клавиатуры символов.

Библиотека `Curses` имеет ОО-интерфейс, в этом случае вместо инициализации окна по умолчанию через `initscr` можно использовать вызов `new`:

```
my $win = Curses->new();
```

Многие функции C-библиотеки ncurses имеют различные вариации одной и той же функции с префиксами *w*, *mv*, и *wmv*. Префикс подразумевает, что появляется какой-либо дополнительный параметр: *w* — объект окна, *mv* — координаты *y*, *x* позиции курсора. В Perl-библиотеке `Curses` такие вариации функций были объединены в одну, но с опциональными параметрами окна и координат:

```
function( [win], [y, x], args );
```

Такие функции получили название *унифицированных*, и их можно использовать как методы при программировании с использованием ОО-интерфейса. В документации `Curses` приведён полный список функций и указано, какие из них являются унифицированными, а какие — нет. Во всех последующих примерах будет использоваться ОО-интерфейс, поэтому если используется прямой вызов функций, то это явно подразумевает, что функция не является унифицированной.

Рассмотрим пример приложения, которое выводит сообщение в центр экрана, ожидает ввод пользователя и завершает работу:

```
use strict;
use warnings;
use Curses;

my $win = Curses->new();

raw();
noecho();

$win->keypad(1);
$win->getmaxyx(my $row, my $col);

my $str = "Your terminal size: ${row}x$col";

$win->addstr( $row/2, ( $col - length $str )/2 , $str );
$win->refresh();

my $ch = $win->getch();
```

```
19
20 endwin();
```

Приложение создаёт окно. Метод keypad со значением истины в параметре включает обработку специальных клавиш, таких как F1, F2 и т.д. Метод getmaxyx позволяет определить максимальные значения координат на экране, т.е. размер текущего терминала. Метод addstr позволяет вывести строку по заданным начальным координатам. Метод refresh отображает всё то, что мы вывели на экран. Метод getch ожидает нажатия клавиши и возвращает введённый символ. Функция endwin завершает режим curses, возвращая предыдущее содержимое экрана.

Окна и субокна

Окна и субокна используются для того, чтобы установить ограничение для контента, который будет выводиться внутри окна, а также для того, чтобы управлять атрибутами внутри окна.

Для создания нового окна используется функция newwin:

```
1 my $win = newwin($rows, $cols, $y, $x);
```

где $rows, $cols — определяют высоту и ширину окна, а $y, $x — положение окна на экране.

Окна могут перекрываться, при этом в месте пересечения будет отображаться контент того окна, которое было обновлено последним. Если требуется создать окно внутри другого окна, то можно использовать функцию создания субокна subwin или derwin:

```
1 my $subwin = $win->subwin($rows, $cols, $y, $x)
2
3 my $subwin = $win->derwin($rows, $cols, $dy, $dx)
```

Отличие subwin и derwin в том, что в derwin координаты субокна задаются относительно верхнего левого угла родительского окна.

Для удаления окна или субокна используется функция delwin

```
1 $win->delwin();
```

Окно можно перемещать по экрану с помощью функции `mvwin` (`mvderwin` — для субкона):

```
$win->mvwin($newY, $newX)

$subwin->mvderwin($newDX, $newDY);
```

Очистить содержимое окна можно с помощью функции `clear`:

```
$win->clear();
```

Существует возможность отобразить границы окна с помощью функции `box`:

```
$win->box("|","-");
$win->box(0,0);
```

где первый параметр определяет символ для вертикальных линий, а второй — горизонтальных. Если используется значение `0` (или `undef`), то используются символы границ по умолчанию.

Если требуется задать различные символы для всех сторон и углов окна, то можно воспользоваться функцией `border`:

```
$win->border(
    $left, $right, $top, $bottom,
    $top_left, $top_right, $bottom_left, $bottom_right
);
```

Атрибуты

Для каждого выводимого символа может быть установлены атрибуты, в том числе:

- `A_BOLD` — режим повышенной яркости;
- `A_NORMAL` — нормальный режим яркости;
- `A_DIM` — режим пониженной яркости;
- `A_UNDERLINE` — подчёркнутый текст;
- `A_REVERSE` — инверсный текст;
- `A_BLINK` — мерцающий текст;

- A_INVIS — невидимый текст.

Не все терминалы поддерживают указанные атрибуты (как правило, A_NORMAL совпадает с A_DIM, а некоторые атрибуты могут заменяться цветовым выделением).

Установить конкретный атрибут или набор атрибутов можно функцией attron:

```
$win->attron(A_BOLD | A_UNDERLINE);
```

После чего при выводе текста в данное окно начинают применяться указанные атрибуты. Также для установки атрибутов можно использовать функцию attrset, но данная функция сбросит все другие атрибуты, которые были до этого установлены:

```
$win->attron(A_BOLD);       # attributes now A_BOLD
$win->attron(A_REVERSE);    # attributes now A_BOLD +
    A_REVERSE

$win->attrset(A_UNDERLINE); # attributes set to
    A_UNDERLINE
```

Для отключения определённых атрибутов используется функция attroff:

```
$win->attroff(A_BOLD);
```

Цвета

Многие терминалы имеют поддержку цвета. Для того, чтобы определить, поддерживает ли терминал цвет, можно использовать функцию has_colors. Если поддержка цветов в терминале присутствует, то перед работой с цветовой палитрой требуется инициализировать цвета вызовом функции start_color:

```
die "Your terminal does not support color\n" unless
    has_colors;
start_color();
```

Для создания палитры из двух цветов (цвет фона и цвет текста) используется функция init_pair:

```
init_pair(1, COLOR_BLUE, COLOR_BLACK);
```

Установить указанную палитру цветов для вывода можно всё той же функцией attron:

```
$win->attron(COLOR_PAIR(1));
```

Цветовая палитра ограничена 16 цветами, причём базовых цветов только 8, а дополнительные цвета получаются за счёт использования повышенной яркости (A_BOLD):

```
COLOR_BLACK   0
COLOR_RED     1
COLOR_GREEN   2
COLOR_YELLOW  3
COLOR_BLUE    4
COLOR_MAGENTA 5
COLOR_CYAN    6
COLOR_WHITE   7
```

Существует возможность переопределить любой базовый цвет с помощью функции init_color, которая для заданного базового цвета устанавливает значения интенсивности соответственно красного, зелёного и синего цветов в диапазоне от 0 до 1000:

```
init_color(COLOR_BLACK, 100,100,100); # black -> dark
    gray
```

Не все терминалы поддерживают такую возможность, поэтому перед изменением цветов палитры следует выяснить, присутствует ли такая возможность в текущем терминале, с помощью функции can_change_color.

Обработка ввода

Для получения значения введённого символа используется метод getch. В случае если была активирована обработка специальных клавиш (F1, PgUp, PgDown и т.д.) с помощью keypad(1), то getch возвращает числовой код специальной клавиши, например, значение 338 для PgDown. Если же обработка специальных клавиш отключена, то getch() вернёт лишь первый код из escape-последовательности

клавиши.

Если на вводе оказывается символ Unicode, то getch воспринимает лишь первый байт последовательности. Соответственно для обработки Unicode-символа, имеющего внутреннее представление из двух байт, потребуется два последовательных вызова getch:

```
$win->keypad(1);
$ch = $win->getch();

if (length $ch > 1 && $ch == KEY_F(1)) {

    # looks like a F1 key
    $win->addstr(1, 1, "F1 KEY!");
}
elsif (ord($ch) < 128) {

    # looks like an ASCII
    $win->addstr(1, 1, $ch);
}
elsif (ord($ch) >> 5 == 0b110) {

    # hack: looks like a first byte of 2bytes UTF-8
       unicode code point
    $ch .= $win->getch();
    $win->addstr(1, 1, decode_utf8($ch));
}
```

В Curses определены константы для функциональных клавиш, а также функция KEY_F, возвращающая код F*-клавиш. Названия констант можно найти в документации Curses, все они имеют префикс KEY_.

В C-библиотеке ncursesw, есть функция get_wch, которая может обрабатывать ввод «широких» символов, но, к сожалению, она не перенесена в Perl-обвязку Curses.

С помощью функции halfdelay можно указать, какое время (в десятых долях секунды) getch должен ожидать ввода символа. Если после истечения таймаута клавиша не нажата, getch вернёт значение ERR (-1).

```
halfdelay(10)
$ch = $win->getch();
if ($ch == ERR) {
```

```
4
5     # Timeout
6     ...
7 }
```

Для получения строки символов до символа переноса строки (или `EOF`) может использоваться функция `getstr`:

```
1 $win->getstr(my $str);
2 $win->addstr(1,1,"You type: $str");
```

Управление курсором

Получить текущие координаты курсора можно с помощью `getyx`:

```
1 $win->getyx(my $y, my $x);
```

Если вызывается какая-либо функция вывода на экран без указания координат, то вывод производится начиная с текущей позиции курсора. Курсор можно передвигать по экрану с помощью функции `move`:

```
1 $win->move($y, $x);
```

Очистка экрана

Существует возможность очистки экрана или его части. Под очисткой понимается заполнение его пробельными символами.

Функция `clear` очищает окно целиком:

```
1 $win->clear();
```

Функции `clrtoeol` и `clrtobot` очищают экран, начиная с текущей позиции курсора до конца строки и конца окна соответственно:

```
1 $win->ctrloeol();
2 $win->ctrlobot();
```

Функции `deleteln` и `delch` удаляют строку и символ соответственно на текущей позиции курсора:

```
$win->deleteln();
$win->delch();
```

Функция insdelln вставляет указанное число пустых строк, начиная с текущей позиции курсора (или удаляет, если аргумент отрицательный).

```
$win->insdelln(10);
```

Обработка событий мыши

Curses позволяет обрабатывать события, возникающие при использовании мыши, такие как нажатие клавиш и перемещение указателя. Перед началом обработки действий мыши необходимо задать маску событий, которые будет фиксировать программа. Делается это с помощью функции mousemask:

```
mousemask(ALL_MOUSE_EVENTS, my $old_mask);
```

Фиксировать наличие действий, произведённых мышью, можно всё той же функцией getch. Функция getmouse позволяет получить структуру произошедшего события, куда входит информация об идентификаторе устройства мыши (их может быть несколько), координатах мыши в трёх измерениях и непосредственно типе события.

```
my $key = $win->getch();
if ($key == KEY_MOUSE) {
    getmouse(my $ev);

    # fix align of short in C-structure
    my ($id, $x, $y, $z, $state) = unpack("s x![i] i3 L",
       $ev);

    if ($state & BUTTON1_CLICKED) {

        # left button clicked

    }
}
```

Изменение размера экрана

Поскольку зачастую консольные приложения запускаются внутри графического окна эмулятора терминала, то во время работы приложения может происходить изменение размеров экрана, когда пользователь растягивает окно терминала или наоборот уменьшает. Для этих случаев существует сигнал `SIGWINCH`, которым эмулятор терминала оповещает запущенные приложения об изменении размера экрана.

Библиотека `Curses` устанавливает обработчик этого сигнала и пытается адаптировать существующие окна под новый размер экрана. Конечно не всегда это может быть выполнено корректно, поэтому для пользовательского приложения реализована возможность получить оповещение об произошедшем изменении размеров экрана — функция `getch` возвращает значение `KEY_RESIZE`:

```
my $key = $win->getch();
if ($key == KEY_RESIZE) {

    # get new window size
    $win->getmaxyx(my $row, my $col);
}
```

Не стоит переопределять обработчик сигнала `SIGWINCH` в коде приложения, поскольку это приведёт к завершению работы приложения после получения сигнала.

Панели, меню и формы

В рамках проекта ncurses существуют дополнительные библиотеки, расширяющие возможности ncurses:

- Панели. Представляют собой окна, которые могут перекрывать друг друга, при этом можно указать в каком порядке следуют панели, т.е. указывается третье измерение — глубина. Панели можно скрывать, отображать, перемещать наверх в стопке.

- Меню. Позволяет реализовывать внутри ваших приложений привычные меню с поддержкой вложенности.
- Формы. Позволяют создавать привычные элементы для ввода данных в виде текстовых полей, которые можно заполнять и редактировать.

Подробное описание этих расширений может стать темой для ещё одной статьи, но вполне возможно, если приложение требует подобного функционала, стоит посмотреть на более мощные альтернативы на CPAN, например, `Curses::UI` или `Tickit`.

■ *Владимир Леттиев*

4. Миграция веб-приложений с Dancer на Dancer2

Готова ли новая версия фреймворка `Dancer2` для использования в промышленной эксплуатации? Сложно ли произвести миграцию существующего приложения на новую версию фреймворка? Какие преимущества даст переход, и есть ли недостатки? Данная статья попытается дать ответы на поставленные вопросы и осветить подводные камни миграции.

Что нового есть в `Dancer2`

На момент написания статьи на CPAN доступен дистрибутив `Dancer2` версии 0.09, вышедший 2 сентября 2013 г., поэтому все факты, изложенные ниже, справедливы именно для этой версии.

`Dancer2` — это легковесный веб-фреймворк для Perl нового поколения. `Dancer2` был практически написан заново, чтобы решить некоторые серьёзные недостатки дизайна `Dancer`, такие как использование глобальных переменных и синглтонов и отсутствие хорошего и ясного объектно-ориентированного внутреннего API.

В новой версии `Dancer2` стал использоваться мощный и легковесный ООП-фреймворк `Moo`, который позволил легко строить и расширять внутренние классы фреймворка. Особое внимание было уделено обратной совместимости, чтобы необходимость изменений в коде для перехода на новую версию фреймворка требовалось как можно меньше (в идеале не надо было исправлять ничего). Но, к сожалению, некоторые шероховатости в миграции и задержка с выпуском новой версии привели к тому, что `Dancer2` был объявлен не как замещение, а как параллельный проект со своим пространством имён, в котором будут появляться новые фичи, в то время как `Dancer` продолжит своё существование только в режиме поддержки и исправления ошибок.

Кроме внутренних изменений в `Dancer2` на данный момент нет

каких-то новых убийственных фич. На данном этапе Dancer2 лишь пытается догнать Dancer по поддерживаемым возможностям.

Отличия Dancer2

При миграции веб-приложения на Dancer2 можно столкнуться с необходимостью внесения изменений как в код, так и в конфигурацию приложения. Попробуем разобрать, какие произошли изменения.

Отличия в конфигурации

В отличие от Dancer, в Dancer2 поддерживается конфигурационные файлы различного формата: yaml, json, apache, ini и прочие, которые понимает модуль Config::Any. Таким образом, если кто-то испытывает дискомфорт в работе с YAML, может переписать конфигурационный файл config.yml в удобный ему формат. Dancer2 будет пытаться найти конфигурационный файл нужного формата исходя из названия расширения файла.

Изменилось описание конфигурации шаблонизаторов и движков сессий, например:

```
template: "xslate"
engines:
    xslate:
        cache: 1
        cache_dir: "cache"
```

в Dancer2 это нужно будет изменить на:

```
template: "Xslate"
engines:
    template:
        Xslate:
            cache: 1
            cache_dir: "cache"
```

Т.е. добавляется промежуточный параметр, указывающий тип движка: `template` или `session`.

Пример изменения конфигурации для движков сессий:

```
session: "memcached"
session_expires: 604800
memcached_servers: "127.0.0.1:11211"
```

Изменяется на:

```
session: "Memcached"
engines:
    session:
        Memcached:
            session_duration: 604800
            memcached_servers: "127.0.0.1:11211"
```

Значения каталогов, таких как `confdir` и `public`, невозможно установить в файле конфигурации или через вызов `set` в приложении. Единственный пока способ изменить их значения с помощью переменных окружения:

```
DANCER_CONFDIR
DANCER_PUBLIC
```

Если вы устанавливаете эти переменные окружения в самом приложении, то это необходимо делать до вызова `use Dancer2`, т.е. в секции `BEGIN`.

Плагины

`Dancer2` использует ООП-фреймворк `Moo`, что серьёзно поменяло техническую реализацию модулей плагинов. Таким образом, использовать существующие плагины для `Dancer` стало невозможно, и требуется переписывать их под `Dancer2`, что в свою очередь может отразиться и на их API, и на опциях конфигурации, и в общем случае нельзя наверняка сказать — потребуется ли что-то менять в коде приложения, которое использовало возможности данного плагина.

Посмотреть список существующих плагинов для Dancer2 можно на странице проекта на github. Не все из них могут оказаться рабочими, так как API плагинов Dancer2 всё ещё в активной разработке.

Подключение Dancer2

В Dancer2 реализована поддержка лишь части аргументов при подключении модуля:

```
1 use Dancer2 qw( :tests :script !keyword )
```

Такой часто используемый аргумент загрузки модуля как :syntax, в Dancer2 отсутствует. Несмотря на наличие опции :script, в Dancer2 пока не реализована поддержка конфигурации через опции командной строки.

Изменения в DSL

В Dancer2 перенесены все директивы за исключением тех, которые помечены как устаревшие в Dancer. Например, мне при переносе небольшого проекта на Dancer2 в коде не пришлось менять ничего, кроме замены use Dancer на use Dancer2.

Развёртывание веб-приложений

Наиболее рациональный способ развёртывания Dancer-приложений — это запуск их в standalone-режиме с помощью plackup и проксирование к ним запросов через фронтенд-веб-сервер *Apache* или *Nginx*. Также можно использовать запуск через *FCGI*. В этом отношении развёртывание веб-приложений на Dancer2 не изменилось.

Гораздо интересней вариант развёртывания с использованием *mod_perl2* и *Apache* в режиме *prefork* или с *MPM worker*. Как известно, в Dancer невозможно в рамках одного процесса запускать два разных приложения, так как они разделяют общие данные конфигурации и маршрутов.

Что же изменилось в Dancer2? Рассмотрим конфигурацию с двумя виртуальными хостами:

```
<IfModule mpm_prefork_module>
    StartServers          1
</IfModule>

PerlSwitches -I/srv/App1/lib
PerlSwitches -I/srv/App2/lib

<VirtualHost *>
    ServerName app1.local
    DocumentRoot "/srv/App1/public"
    <Location />
        SetHandler perl-script
        PerlHandler Plack::Handler::Apache2
        PerlSetVar psgi_app /srv/App1/bin/app.pl
    </Location>
</VirtualHost>
<VirtualHost *>
    ServerName app2.local
    DocumentRoot "/srv/App2/public"
    <Location />
        SetHandler perl-script
        PerlHandler Plack::Handler::Apache2
        PerlSetVar psgi_app /srv/App2/bin/app.pl
    </Location>
</VirtualHost>
```

Проверим их работу:

```
$ curl -s http://app1.local/ | grep '<title>'
<title>App1</title>
```

```
$ curl -s http://app2.local/ | grep '<title>'
<title>App1</title>
```

Как видно, в конфигурации с одним рабочим процессом первое запрошенное приложение фиксирует конфигурацию, и второе приложение начинает использовать шаблоны и маршруты первого приложения, т.е. в данной ситуации поведение Dancer2 нисколько не отличается от Dancer.

Если обратиться к документации Dancer2, то для объединения двух приложений в одно рекомендуется использовать Plack::Builder в таком виде:

```
use App1;
use App2;
use Plack::Builder;

builder {
    mount '/app1' => App1->dance;
    mount '/app2' => App2->dance;
};
```

Такая конфигурация подразумевает работу двух приложений в рамках одного виртуального хоста, и приложения разделяются по префиксу в URI: `/app1` и `/app2`. К сожалению, на практике и такая конфигурация является нерабочей, поэтому вопрос об использовании двух и более приложений `Dancer2` в рамках одного процесса остаётся открытым.

Производительность

Тест производительности не претендует на высокую точность, но позволяет примерно оценить относительные изменения в производительности между `Dancer` и `Dancer2`. С помощью утилит, которые генерирует шаблонное приложение, создадим веб-приложения на `Dancer` и на `Dancer2`:

```
$ dancer -a Bench::Dancer
```

```
$ dancer2 -a Bench::Dancer2
```

Запустим веб-сервер на основе *Twiggy* по очереди для каждого приложения и запустим утилиту для бенчмарка `ab2`.

```
$ plackup -Ilib -E deployment -s Twiggy -a bin/app.pl -p
    5001 &
```

```
$ ab2 -n 2000 http://127.0.0.1:5001/
```

Для `Dancer` получим следующий отчёт:

```
Document Length:        5582 bytes
Time taken for tests:   10.657 seconds
Complete requests:      2000
Failed requests:        0
Requests per second:    187.66 [#/sec] (mean)
Time per request:       5.329 [ms] (mean)
```

Для Dancer2 отчёт будет такой:

```
Document Length:        5540 bytes
Time taken for tests:   12.192 seconds
Complete requests:      2000
Failed requests:        0
Requests per second:    164.04 [#/sec] (mean)
Time per request:       6.096 [ms] (mean)
```

Абсолютные величины здесь не интересны (тестовая система с весьма скромной производительностью), но можно оценить, что производительность в Dancer2 просела на ~13% при прочих равных условиях.

Объём занимаемой памяти

Объём занимаемой памяти Perl-программы зависит от версии интерпретатора и опций сборки. Для случая Perl 5.16.3, собранного в виде разделяемой библиотеки libperl.so для платформы x86_64 с поддержкой тредов, при запуске голого perl получаем такую статистику из /proc/$$/status:

```
VmSize:    19108 kB
VmRSS:      2028 kB
VmData:      520 kB
VmLib:      4524 kB
```

Т.е. общий объём занятой виртуальной памяти — 19 МБ, из них 2 МБ находятся постоянно в памяти, 4,5 МБ — загруженные разделяемые библиотеки, 0,5 МБ — это объём сегмента данных.

Попробуем теперь сравнить объём занимаемой памяти минимальных Dancer и Dancer2 приложений при запуске в standalone-режиме:

```
$ perl -Ilib bin/app.pl
```

Статистика для Dancer:

```
VmSize:    59296 kB
VmRSS:     15940 kB
VmData:    13420 kB
VmLib:      4924 kB
```

Статистика для Dancer2:

```
VmSize:    78848 kB
VmRSS:     23028 kB
VmData:    20252 kB
VmLib:      5512 kB
```

Таким образом, минимальное приложение на Dancer2 занимает уже на ~30% больше виртуальной памяти и на ~45% больше резидентной.

Зависимости

Dancer и Dancer2 имеют довольно большое дерево зависимостей. С помощью утилиты strace можно оценить количество модулей, которые требуются при загрузке минимального приложения:

```
$ strace -e open -o dancer.trace perl -Ilib bin/app.pl &
...
$ grep '\.pm' dancer.trace | wc -l
127

$ egrep 'perl5.+\.so' dancer.trace | wc -l
11
```

Таким образом, Dancer в общей сложности загружает 127 модулей и 11 разделяемых библиотек XS-модулей.

Для Dancer2 статистика существенно тяжелее: 217 модулей и 18 разделяемых библиотек XS-модулей. То есть на ~70% больше загружаемых модулей, что и объясняет увеличение занимаемой памяти.

Для сравнения — Moose загружает 109 модулей...

Выводы

Если после прочтения статьи у вас сложится впечатление, что мигрировать на Dancer2 не имеет смысла, то рекомендую вам всё же попробовать для эксперимента это сделать. Если что-то сломается,

то информация об этом может оказаться очень полезной для разработчиков, чтобы к следующему релизу это могло быть исправлено.

■ *Владимир Леттиев*

5. Обзор CPAN за сентябрь 2013 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц.

Статистика

- Новых дистрибутивов — 253
- Новых выпусков — 988

Новые модули

- Tickit::DSL Модуль реализует DSL-язык для описания консольных интерфейсов приложений, реализуемых на основе модуля `Tickit`.
- Exporter::Tiny Альтернативная реализация `Sub::Exporter` предоставляет возможность экспорта функций, включая возможность их переименования, а также поддерживаются конфигурация через переменные `@EXPORT`, как в `Exporter`. При этом модуль не имеет небазовых зависимостей.
- AnyEvent::WebSocket::Client Реализация WebSocket-клиента на основе `AnyEvent`.
- Error::Tiny Легковесная реализация обработки исключений try/catch. Исключение всегда передаётся в catch в виде объекта, что удобно для обработки. Правда, в отличие от `Try::Tiny`, содержимое `$@` не предохраняется.
- ExtUtils::MakeMaker::CPANfile Модуль включает поддержку файла `cpanfile` в скриптах установки на `ExtUtils::MakeMaker`.
- Ark Вышел первый релиз на CPAN уже давно существующего `Catalyst`-подобного веб-фреймворка `Ark`.

- Elasticsearch Вышел официальный Perl-клиент для распределённого движка поиска и анализа `Elasticsearch`. Модуль `ElasticSearch` объявлен устаревшим.
- IO::Socket::Timeout Модуль добавляет возможность установки таймаутов для чтения и записи в `IO::Socket::INET`.
- Seis Модуль для трансляции Perl 6 синтаксиса в Perl 5. Возможности модуля ещё не слишком велики и далеко не все конструкции Perl 6 переносимы, но как концепт модуль очень интересен.
- Text::Markdown::Hoedown Модуль обвязки к библиотеке Hoedown, являющейся ожившим форком библиотеки Sundown — парсера формата markdown, которая была по странным причинам объявлена устаревшей без объявления преемника.

Обновлённые модули

- File::LibMagic 1.00 Вышел мажорный релиз модуля для определения типа MIME-данных с использованием `libmagic`. В новой версии появилась возможность передавать ссылку на скаляр с данными для анализа.
- Starman 0.4008 В новом релизе веб-сервера Starman сделано несколько исправлений, в том числе обработка сигнала `EPIPE` при отправке данных клиенту.
- Cache 2.06 После продолжительного забвения вышел новый релиз модуля для управления кэшем. В новой версии произошёл переход на сборку с `Module::Build` и исправлены некоторые ошибки.
- Date::Manip 6.41 Вышел новый багфикс-релиз модуля `Date::Manip` для проведения различных вычислений с датами.
- Clone 0.35 В новом релизе `Clone` исправлена проблема с `SIGSEGV` в perl при попытке клонирования структуры, в которой есть скаляр со значением `NULL`.

- Config::Any 0.24 В новой версии наконец-то убрали ужасно раздражающее предупреждение о том, что не установлен `YAML::XS`.
- local::lib 1.008018 Новая версия `local::lib` теперь работает и при установленным флаге `-T` (taint-режим) при запуске Perl.
- autodie 2.22 Множество исправлений и улучшений в работе модуля autodie в новом релизе.
- perlsecret 1.004 Обновился замечательный документ о секретных операторах и константах в Perl. В этом выпуске раскрыт новый секретный оператор «ключ к правде» `0+!!`
- Tickit 0.40 Вышел новый релиз инструментального набора для создания программ с консольным интерфейсом. В данном выпуске исправлены ошибки и обновлена документация.

■ *Владимир Леттиев*

6. Интервью с chromatic

chromatic — автор книги Modern Perl, участвовал в разработке Perl 6 и Parrot, автор многих популярных модулей на CPAN.

Когда и как начал программировать?

В который раз?

Как и многие мои сверстники в штатах (белые дети из среднего класса, выросшие за городом), в первый раз я увидел компьютер в начальной школе. Мне было пять или шесть лет. Это был Commodore 64. В те дни компьютеры поставлялись с документацией и BASIC, и в документации было описано как писать простые програмы на BASIC практически на первых страницах. (Страница 32 — только что проверил. Первые несколько страниц рассказывают, как подключить телевизор или монитор и как работает клавиатура.)

В конце концов я уговорил родителей купить для семьи компьютер.

Когда я был в старшей школе, мои интересы изменились и я больше проводил времени, играя музыку, чем программируя, но в колледже я попытался заняться C++ (не пошло) и Java (пошло немного лучше) и на первой в своей карьере работе я написал несколько программ для отдела, занимающегося принтерами в HP, и нашел это самым увлекательным среди всего, чем я занимался.

Какой редактор используешь?

Vim.

Когда и как познакомился с Perl?

Java была новой и интересной, когда я начал работать в HP, но я работал с HP-UX и Linux (Red Hat 4 или 5, кажется). Java на Linux была сильно позади в то время, но там хотя бы была базовая поддержка Swing.

Второй серьезный проект, над которым я работал, должен был

отправлять email списку клиентов об обновлениях сетевой службы поддержки принтеров. Я набросал на коленке bash-скрипт на версии, которую поддерживал в то время HP-UX 9. Он был длиной в десять строк. Затем мне пришлось портировать его на Java, потому как отделу нужно было запускать его на Windows NT. В том время Java была еще молодой. У нее не было большого количества библиотек. Для отправки email нужно было писать много кода для подключения к SMTP-серверу и отправки правильных заголовков и прочего. Нужно было написать больше кода для инициализации Java-библиотек, чем написать всю задачу на shell.

Я начал искать другие языки. Мой брат учился и работал с телекоммуникациями в университете в то время, и многое автоматизировал с помощью Perl. Я купил второе издание Camel book, прочел ее от корки до корки, и понеслась.

С какими другими языками приятно работать?

Haskell, за его систему типов, которой можно пользоваться (а также за то, что он поощряет написание небольших кусков кода, которые разумно взаимодействуют).

Scheme, за возможность построить хорошую систему из небольших годных компонентов.

Хотелось, чтобы и Go понравился, но для меня он пока не был достаточно полезным.

Продолжаю думать, что в C++ есть хороший язык, который пытается выбраться, но мне кажется я не достаточно умный, чтобы его обнаружить.

Какое, по-твоему, самое большое преимущество Perl?

Бесцеремонность. Можно дать Perl кому-то с небольшим опытом программирования и он или она сделают что-то гораздо быстрее, чем вы можете ожидать. Он или она скорее всего напишут какую-то жуть — развитие дисциплины в програмировании требует времени и опыта — но прагматизм Perl это большое преимущество.

Какими, по-твоему, особенностями должны обладать языки будущего?

Все зависит от языка. Большинство людей считают, что скорость выполнений это самая важная часть, и в некоторых случаях так и есть. Если вы хотите получить большое количество пользоватлей, необходимо дать им возможность делать, что они хотят, не раздражая их настолько, что они уйдут куда-нибудь. Опять же, прагматизм здесь идет на пользу.

У PHP много пользователей не потому что это хороший язык. У PHP много пользователей потому что это самый простой способ сделать динамическую веб-страницу.

Ты многое вложил в экосистему и инфраструктуру Perl. Что является причиной этому?

Я хочу упростить свою жизнь. Хорошие тесты у CPAN-модулей означают, что я могу доверять CPAN-модулям больше, чем раньше. Наличие простых и хороших абстракций для таких вещей как email или OO-разработки означает, что я могу выполнить свою работу лучше, затратив меньше сил, чем раньше.

Наличие в свободном доступе хорошей книги для начинающих, какой я надеюсь является Modern Perl, означает, что больше людей начнут писать хороший Perl-код, а не плохой, если бы они учились по ужасным, устаревшим урокам, которые можно найти в сети.

В течение своей работы над Perl 6 ты занимался ролями. Являются ли они основой современного ООП? Это «серебряная пуля» или «еще один способ прострелить себе ногу», если говорить о реальных больших и сложных проектах?

Если читать современные ООП-уроки, плохие будут объяснять, что наследование, это основая ООП. Это глупо. Полиморфизм — это основа ООП.

(Полиморфизм — важная часть в программировании. Если вы не научились работать с объектами, разнящимися в мелочах, но одинаковых в главном, у вас будет много проблем. Люди часто не учат-

ся работать с массивами, потому что хотят именовать перменные `$recipe1`, `$recipe2` и `$recipe3`. И только когда они осознают, что могут работать с переменными как с элементами массива `@recipes`, они начинают понимать силу программирования.)

Во время разработки ролей я доказывал, что нам нужно решить две проблемы:

1. как определить достаточно концептуально похожие поведение и состояние, что они могут быть полиморфными;
2. как делиться кодом без навязывания метода как это делать (делегирование, наследование, композиция, копипаста, роли).

Авторы оригинальной статьи о Smalltalk Traits пытались решить похожие проблемы, поэтому, я думаю, мы были на правильном пути.

Можно ими злоупотреблять? Да. Можно ли «прострелить себе ногу»? Конечно. Являются ли они базовыми для хорошего ООП-проекта? Зависит от проекта.

Исходя из моего опыта, эффективное использование ролей и параметрических ролей улучшило дизайн нескольких проектов. Требуется время и опыт для понимания, как использовать роли эффективно, но если вы дисциплинированный программист, который хочет переработать дизайн своей системы, когда это может потребоваться, роли это еще один инструмент в ООП-проектировании и анализе, на который стоит обратить внимание.

Почему тестирование важно?

Я хочу усилить свою уверенность в том, что код, который я использую, работает. Тестирование не дает такой уверенности. Мне вполне достаточно, что у меня нет стопроцентной уверенности, но если я могу посмотреть на тесты и они мне понятны, у меня становится меньше забот.

Меня сильно не заботит разница между юнит-, интеграционными, поведенческими, спецификационными тестами и так далее. Я стараюсь оставаться прагматичным. Я хочу знать, что важное поведение,

на которое рассчитываю, работает. Я хочу знать, что вы подумали о граничных значениях. Я хочу знать, что если я внесу изменение и сломаю что-нибудь, запуск тестов выявит это.

Это все о качестве, уверенности и надежности. Это также показатель дисциплины и качества программиста. Вам не обязательно иметь автоматические тесты для хорошего продукта, но если у вас нет автоматических тестов, я хочу знать, почему я должен доверить продукту, который вы пишете.

По прошествию уже более десяти лет соответствует ли `Test::Builder` современным практикам в тестировании?

В том что он делает — да. В том как он выглядит внутри? Нет. Надеюсь, что `Test::Builder` 1.5 или 2.0 наконец выйдет и подчистит код.

Что побудило тебя написать книгу Modern Perl?

Perl требовалось короткое введение, которое бы не притворялось, что покрывает весь язык. Я подумал, что объяснение философии языка, как использовать perldoc, необходимость CPAN и для чего использовать и для чего не использовать язык, было бы полезным.

Почему решил сделать книгу свободно доступной?

Три причины. Во-первых, люди всегда достанут ее бесплатно, разными способами нарушая копирайт, если действительно этого захотят. Борьба с этим не стоит моего времени. Во-вторых, я надеялся на более широкий охват (с продвижением платной печатной и электронной версии), если книга будет доступна бесплатно. В-третьих, я знаю, что есть люди в восточной Европе и западно-восточной Азии и других регионах, которые бы никогда не купили печатную версию из-за дорогой доставки (или дороговизны самой книги). Я не могу писать на всех языках, но могу дать бесплатное издание на английском, которое кто-то может перевести.

Планируется ли обновление книги, совместно с эволюцией Perl?

Я хочу выпустить новое издание до конца текущего года.

Что случилось с perl.com? Кажется, что не было никаких обновлений с января.

Не так просто найти авторов, а я не хочу быть единственным автором (как это нам знакомо — *прим. ред.*).

У тебя несколько бизнесов. Чем в основном занимаешься сейчас?

На данный момент я консультирую один стартап в медицинской индустрии.

Каким видишь будущее Perl?

Надеюсь, что такие проекты как MOP Stevan Little и сигнатуры функций Peter Martini войдут в ядро языка.

Сомневаюсь в проектах, который хотят перенести Perl на новый движок, во всяком случае без исправления ситуации с XS.

CPAN продолжит расти.

Не знаю, что произойдет с рекламой Perl. (Некоторые люди называются это «маркетингом».)

Perl продолжит помогать моим клиенам и проектам получать правильные ответы.

Есть ли место для Perl 6?

Если он найдет что-то, что он делает лучше других существующих языков, то возможно.

Стоит ли советовать молодым программистам изучать Perl?

Это зависит от того, что они хотят написать. Вчера я разговаривал с ученицей старших классов, которая хочет изучать компьютеры, у нее есть опыт работы с HTML и она хочет выучить JavaScript. Нужен ли ей Perl? (Есть ли простой Windows-дистрибутив, который, используя магию Plack, позволяет разворачить Perl-программы так же просто как и PHP-программы, но не требует использования CGI.pm?

Возможно.)

Вопросы от читателей

На PerlMonks у тебя в качестве аватара футболка Perl 7. Что думаешь о дискуссиях вокруг Perl-версий?

В ретроспективе, возможно, название следующей версии Perl 6 в 2000 году было не очень удачным — но это не изменится, если только Ларри не захочет этого, поэтому мы застряли с Perl 5.«что-то там» в обозримом будущем.

Что думаешь о ситуации вокруг smart match?

У Perl 5 всегда была идея, что переменные должны быть полиформными, в то время как операторы — мономорфными. Другими словами, если вы видите `$a + $b`, то вы знаете, что это сложение (неважно, что находится в `$a` и `$b`), и если вы видите `$a . $b`, то вы знаете, что это конкатенация (неважно, что находится в `$a` и `$b`).

Со smart match все иначе, и каждый раз, когда вы отходите от изначального дизайна языка, вам необходимо осторожно учесть все плюсы и минусы. Smart match похож на способ решить несколько проблем одновременно. При отсутствии четкого множества мономорфных типов переменных, вы не можете посмотреть на код и сразу же понять, что происходит.

Такая же проблема и у `each()` на ссылках; вы не можете понять, когда `each $foo` вернет ключи и значения хеша или индексы и значения массива.

Откуда берешь энергию для регулярных статей о программировании в своем блоге?

Я всегда был писателем, и как любое регулярное упражнение, это развивает мышцы (буквально и фигурально).

Между тем, моя работа и личная жизнь требуют много времени, и

частота моих постов снизилась от нескольких раз в неделю до еженедельных выпусков. Было бы хорошо иметь немного больше свободного времени.

Было сложно после нескольких лет бросить разработку Perl 6?

И да, и нет.

Мне уже продолжительное время это не доставляло удовольствие по многим причинам. Самым большим было, наверное, мое разочарование, что после девяти лет работы над не очень интересными вещами, такими как исправление сегфолтов и оптимизации, до сих пор ничего не было готовым для использования в работе.

Мне было сложно оправдать разработку в рабочие часы, потому что у меня было достаточно работы, которая требовала работающего кода, и у меня было достаточно потенциальных проектов, которыми я занимался в свободное время и которые мне действительно нравились.

С другой сторой, если ты над чем-то работаешь девять лет, у тебя возникает чувство привязанности. Я горд некоторой работой, которую мы сделали, и я многому научился, но если бы мне сказали в 2001 или 2002, что будет в 2013, я бы потратил свои силы на что-то другое.

На этот урок я потратил очень много времени и сил. Я чувствовал себя виноватым, что это навредит Parrot, если меня там не будет, чтобы чинить баги и писать код. Но затем я осознал, что ни для кого Parrot не был полезным, поэтому, наверное, это и хорошо. Если проект не может выжить, когда теряет разработчиков — если он не используется для чего-нибудь продуктивного или он никого не волнует — то, наверное, ресурсы затрачиваемые на него, стоит приберечь для чего-то другого. Несомненно, это было мое личное понимание.

Я потратил практически десятилетие, работая над этими проектами, поэтому я считаю, что сделал все возможное, чтобы они были успешными. Если кто-то другой хочет продолжить мою работу, код открыт. Но ни один из этих проектов не стоит уже моего времени

или энергии, и я комфортно себя чувствую, объявляя это. Если вы видите себя в похожей ситуации, то любой, кто пытается навязать вам чувство вины, если вы думаете о вещах поважнее разработки продукта, от которого у вас нет никакого дохода, поступает совершенно несправедливо. (Даже если это *вы* себя накручиваете.) Вкладывание своего свободного времени и таланта в свободное ПО это дар, и это такой дар, который вы имеете право отменить.

Нравятся Perl-конференции?

В течение года я не посещал конференции по нетехническим причинам и наслаждался тем, что не путешествую. В то же время, мне всегда доставляет удовольствие проводить время среди великолепных людей Perl-сообщества, поэтому я скучаю по этому году.

■ *Вячеслав Тихановский*

7. Perl Golf

Perl Golf — это соревнование по поиску Perl-кода наименьшего размера (меньше всего символов), который решает поставленную задачу.

По мотивам проведённого на YAPC::Eurore 2013 соревнования по Perl Golf, устроенного компанией Reg.ru, которое привлекло множество людей к решению головоломки, возникла идея публиковать задания в журнале, собирать решения в течение месяца и делать в последующем номере обзор решений с объявлением победителя соревнования.

Цифры

Наверняка многие из вас в школьные годы играли в игру под названием «Цифры» (или «Числа»). Суть игры такова: на тетрадный лист выписываются цифры, каждая цифра записывается в отдельную клеточку, слева направо, по девять цифр в ряд. Начальная последовательность зафиксирована:

```
123456789
111213141
516171819
```

Задача состоит в том, чтобы зачёркивать две соседние цифры (по горизонтали или по вертикали), которые имеют одинаковое значение или их сумма равна десяти. Соседними считаются те цифры, между которыми нет других цифр или все промежуточные цифры зачёркнуты. Кроме того считается, что последняя цифра ряда, является соседней с первой цифрой последующего ряда (т.е. следующий ряд является продолжением предыдущего по горизонтали).

Пример удаления цифр:

```
123456789   _23456789   _2345678_
111213141 → _11213141 → __1213141
516171819   516171819   516171819
```

Сначала удаляются две цифры `1`, которые расположены вертикально, на следующем шаге удаляются `9` и `1`, которые являются соседними, поскольку между ними нет других цифр (промежуточная `1` была удалена на предыдущем шаге).

Количество вариантов для удаления может быть несколько, и в этом и заключается интерес игры, подобрать такие сочетания, чтобы удалить максимальное число цифр.

Как только все доступные для удаления цифры были вычеркнуты, ряд цифр, сразу после последней записанной позиции, дополняется последовательностью, состоящей из оставшихся на поле цифр. Например:

```
1234567__    1234567__
____1314_ → ____1314_
51617181_    51617181_
             123456713
             145161718
             1
```

После чего процесс повторяется до бесконечности. Задача играющего — попытаться удалить с поля как можно больше цифр.

Задание

Задание для создаваемой программы:

- Получить на `STDIN` последовательность из цифр по девять символов в ряд, ряды цифр разделены символом переноса строки `\n`. Позиция удалённого символа обозначается пробелом.
- Программа должна выдать на `STDOUT` последовательность, в которой удалены все возможные сочетания цифр по правилам игры (эти цифры заменяются символом пробела). Вывод идёт также по девять символов в ряд с символом переноса строки для разделения рядов.
- Если после удаления цифр один или несколько рядов оказались пустыми, то эти ряды должны быть удалены из вывода.

Понятно, что для одних и тех же входных данных может существовать несколько вариантов вывода. Но решением будет признаваться любое корректное решение, в котором не осталось возможных комбинаций для удаления и не пропали цифры, которые не могли быть удалены согласно правилам.

Как обычно в Perl Golf, в длину решения не входит первая строка, в которой указан *шебанг* для вызова perl и переданных ему параметров:

```
#!/usr/bin/perl -n
```

Нельзя использовать какие-либо модули и вызывать внешние программы.

Чтобы убедиться, что задача имеет решение, я написал свой вариант, который кажется работает. Указывать его размер не буду, чтобы ~~не позориться~~ была интрига.

Для участия в соревновании сделайте форк репозитория golf-08, создайте в директории `script` файл `your_github_login.pl` с вашим вариантом решения. Проверьте, что ваш вариант проходит тесты (с помощью команды `prove`) и сделайте pull request в основной репозиторий.

Дерзайте! Победителя ждёт, как обычно, уважение и почёт.

■ *Владимир Леттиев*

8. Ответы на Perl Quiz

Ответы из предыдущего выпуска: 1) 2 и 4, 2) 1, 3) 2, 4) 2, 5) 2, 6) 4, 7) 3, 8) 3, 9) 1, 10) 4.