Pragmatic Perl 6

pragmaticperl.com

Выпуск 6. Август 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:

- Андрей Шитов (ash)
- Дмитрий Шаматрин (justnoxx)
- Владимир Леттиев (crux)

Корректор: Андрей Шитов (ash)

Выпускающий редактор: Вячеслав Тихановский (vti)

Ревизия: 2014-11-29 23:30

Оглавление

1 От редактора

Конференция YAPC::Europe 2013 в Киеве состоится уже чуть менее, чем через неделю. Отличная возможность познакомиться и пообщаться с известными Perl-программистами, в том числе и с автором языка — Ларри Уоллом. Кроме самой конференции участников ожидает круиз на теплоходе.

В нескольких предыдущих номерах было объявлено о розыгрыше билета на конференцию. Из 705 подписчиков на сайте конференции было зарегистрировано 71. Победитель был выбран с помощью Perl-функции `rand`. Им стал — Алексей Вавилкин. Поздравляем!

Мы продолжаем искать авторов для следу-

ющих номеров. Если у вас есть идеи или желание помочь, пожалуйста, свяжитесь с нами.

Приятного чтения.

■ *Вячеслав Тихановский*

2 Подготовка к YAPC::Europe 2013

Рассказано про последние приготовления перед конференцией, общий план мероприятия, а также некоторые интересные детали.

Конференция YAPC::Europe 2013, о которой мы уже рассказывали в прошлых выпусках, состоится чуть менее, чем через неделю после выпуска текущего номера. Очень много работы было проделано, но и немало осталось рутины и мелочей, без которых не обходится ни одна конференция.

Место проведения

Традиционно на проведение конференции местные Perl-группы подают заявки, из которых выбирается одна команда. Город,

в котором будет проводиться следующая конференция, обычно объявляется на текущей.

В 2013 году нам повезло, и конференция проходит в Киеве. Это очень удобно для жителей СНГ, которым не нужно оформлять никакие документы для въезда.

В качестве места выбран Украинский дом по адресу: Крещатик, 2. Здание находится в самом центре города, и его невозможно не заметить. Внутри для нас подготовлены три вместительных зала и зал для обедов и кофе-брейков. Перед главным залом находится холл, где можно будет расположиться для общения и совместного хакинга прямо во время конференции (специально для этого будут заказаны пуфики).

Порядок мероприятия

11 августа

С 10 до 18 часов в гостинице «Днипро» в зале «Византия» состоится хакатон, посвященный Perl 6. Его будет проводить известный Perl 6-хакер Jonathan Worthington.

С 19 часов состоится pre-conference встреча в вареничной «Катюша» по адресу: Владимирская, 51. На него зарегистрировалось уже более ста человек.

12 августа

Открытие конференции и доклады первого дня. В этот день выступит Ларри Уолл,

а также множество других известных Perl-хакеров.

После завершения первого дня состоятся вечеринка с мороженым!

13 августа

Второй день конференции. Продолжаются интересные доклады.

После завершения первого дня с 19 до 23 часов состоятся вечеринка и фуршет на самом большом теплоходе в Украине «Rosa Victoria»!

14 августа

Финальный день конференции. Закрытие.

В течение всех трех дней для тех, кто приехал вместе с участниками конференции, будут проводиться экскурсии по Киеву.

Футболки

В этом году мы пошли на эксперимент и предоставили участникам (а их почти 300) возможность самим выбрать цвет и надпись на футболке. Подавляющее большинство участников восприняли это с энтузиазмом.

Благодарности

Хочется поблагодарить всех тех, кто помог нам подготовиться к мероприятию, поучаствовал в организации и оказал медийную поддержку. Особенное спасибо нашим спонсорам, которые обеспечили нам возможность проведения конференции на самом высоком уровне.

Увидимся на конференции!

■ *Вячеслав Тихановский*

3 DBIx::Class в примерах

DBIx::Class — это один из самых популярных ORM для Perl для выполнения SQL-запросов к базам данных через объектно-ориентированный интерфейс. Этот инструмент, при хорошем уровне владения, позволяет быстро разрабатывать приложения, работающие с данными в реляционных базах, позволяя абстрагироваться от языка SQL и нюансов его реализации в конкретной СУБД и оперировать привычными для программиста классами и методами для работы с данными

Зачем нужен DBIx::Class да и ORM вообще?

Аббревиатура `ORM` означает *Object-relational mapping* (Объектно-реляционное отображение) и представляет собой технику, которая позволяет преобразовывать данные из представления их в реляционной базе (таблицы, строки, столбцы и запросы) в объектное представление (объекты, атрибуты и методы).

Разрабатывая программы с использованием объектно-ориентированной парадигмы, программисту привычнее и естественнее обращаться к данным как к объектам, создавая или удаляя их с помощью привычных операторов, применяя методы для изменения атрибутов и т.д. Необходимость использования реляционной

базы данных добавляет программисту новый уровень сложности в разработке: использование языка SQL для манипуляции с данными. Это в общем случае приводит к замедлению разработки (время на составление и проверку SQL-запросов), частому дублированию строк с запросами, плохой читаемости такого кода (особенно когда требуется собрать конечную строку SQL запроса из отдельных кусочков) и необходимости переделки всех запросов при добавлении поддержки другой базы данных (или миграции).

Сравнить это можно с поваром ресторана, который готовит еду и периодически выбегает на огород, чтобы раскидать навоз на грядки, прополоть овощи и выкопать картошки для нового блюда. При этом надо не забыть каждый раз переодеться и вымыть руки. А если времени становится мало (дед-

лайн), то можно увидеть, что повар начинает нарезать лук на грядке или перетаскивать в тележке клумбы с зеленью на кухню, чтобы поменьше выбегать на огород.

Таким образом, ORM создаёт посредника между базой данных и ООП-кодом. С одной стороны это упрощает работу с базой данных, сокращает код и уменьшает время разработки. С другой стороны эта прослойка в некоторых ситуациях может оказаться неэффективной, поскольку генерируемые SQL-запросы могут оказаться медленными или может оказаться, что они разбиваются на несколько подзапросов, увеличивая нагрузку на СУБД.

История DBIx::Class

Проект основал в 2005 году Matt S. Trout как альтернативу `Class::DBI`. Поскольку `Class::DBI` в своё время был вещью достаточно уникальной и имел огромную пользовательскую базу, то попытка сделать его коммерческим продуктом с закрытым кодом привела к тому, что, во-первых, возник сам проект `DBIx::Class`, а во-вторых, он сразу получил большое количество пользователей-перебежчиков и разработчиков, которые начали активно использовать и разрабатывать библиотеку.

В 2008 году на роль «бензопильщика» (*chainsaw delegate* — шуточное прозвище для сопровождающего) был назначен Peter Rabbitson, который заведует релизами проекта и по сей день. Совсем недавно Peter решил поставить вопрос ребром о

своём статусе в данном проекте, поскольку на данный момент он практически является той «единой точкой отказа»: если по каким-то причинам он не может уделить времени проекту, то проект просто замораживается и не развивается, так как не находится такого смелого «бензопильщика», который смог бы подхватить проект. Это вынуждает его снова и снова возвращаться, чтобы срочно сделать накопившиеся задачи. Peter готов полностью сконцентрироваться на `DBIx::Class`, найти финансирование и написать долгосрочный план развития, сообществу нужно лишь подкрепить его уверенность в своих силах словами поддержки и одобрения.

Таким образом на сегодняшний день модуль имеет отличные перспективы по дальнейшему развитию и поддержке.

Создание приложения на DBIx::Class

На момент создания статьи текущей стабильной версией `DBIx::Class` является версия *0.08250*, вышедшая в конце апреля 2013 г., имеющая заметные улучшения в производительности кода. Поэтому имеет смысл перед началом изучения поставить самую свежую версию модуля, чтобы получить адекватное представление о нём.

Для лучшего понимания начнём изучение `DBIx::Class` на примере разработки веб-приложения для форума или коллективного блога. Блоги и форумы — это хороший образец приложений, в которых используется база данных для хранения создаваемого контента, где присутствуют как операции создания и удаления записей,

так и чтения и поиска.

Блог будет построен на основе веб-фреймворка `Dancer`. Создадим базовую структуру приложения с названием `Web::Log`:

```
$ dancer -a Web::Log
$ cd Web-Log
```

Теперь можно перейти к проектированию приложения.

Схема базы данных

В самом простом случае блог содержит три основные сущности: авторы, посты и комментарии. Таким образом в схеме базы данных будут присутствовать три таблицы: `authors`, `posts`, `comments`. Так

будет выглядеть SQL-запрос для создания базы (диалект *SQLite*):

```
CREATE TABLE authors (
    authorid    INTEGER PRIMARY
       KEY AUTOINCREMENT,
    author      VARCHAR(255)
       UNIQUE
);

CREATE TABLE posts (
    postid      INTEGER PRIMARY
       KEY AUTOINCREMENT,
    post        TEXT,
    postime     DATETIME NOT NULL
        DEFAULT CURRENT_TIMESTAMP
       ,
    authorid    REFERENCES
       authors(authorid)
);

CREATE TABLE comments (
```

```
14      commentid   INTEGER PRIMARY
           KEY AUTOINCREMENT,
15      comment     TEXT,
16      comtime     DATETIME NOT NULL
            DEFAULT CURRENT_TIMESTAMP
           ,
17      authorid    REFERENCES
           authors(authorid),
18      postid      REFERENCES posts(
           postid)
19  );
```

В данной схеме мы создаём для каждой таблицы отдельный первичный ключ, даже несмотря на то, что есть другие уникальные поля, которые могут быть использованы в этом качестве. Это нужно прежде всего для удобного использования внешних ключей, кроме того это является одним из требований `DBIx::Class` к таблицам.

Схема также иллюстрирует отношения: автор может создавать несколько постов и комментариев, поэтому таблицы posts и comments имеют внешний ключ `authorid`, у каждого поста могут быть свои комментарии, поэтому таблица `comments` имеет также внешний ключ на `postid`.

Объектное представление схемы базы данных

Для работы с базой данных с помощью `DBIx::Class` мы должны создать классы для данной схемы. В терминах данной ORM схема базы данных — это отдельный класс, задающий пространство имён для базы, например, `Web::Log::Schema`, который мы загружаем и используем в коде. Каждая таблица представляет собой отдельный класс в пространстве имён

`Web::Log::Schema::Result`. В качестве имени класса используется имя таблицы **без окончания** множественного числа — `s`.

Существует два варианта создания необходимых классов: ручной и автоматизированный. Рассмотрим самый простой — автоматизированный, чтобы увидеть как должны выглядеть наши классы. Для этого воспользуемся утилитой `dbicdump` из дистрибутива `DBIx::Class::Schema::Loader`, которая умеет генерировать классы из существующего экземпляра базы данных и автоматически формировать их POD-документацию.

Создадим базу данных (SQLite) из описанного выше SQL (`weblog.sql`):

```
$ sqlite3 weblog.db < weblog.sql
```

Генерируем классы:

```
$ dbicdump -o dump_directory=./lib \
  -o components='["InflateColumn::DateTime"]' \
  Web::Blog::Schema dbi:SQLite:./weblog.db
```

Классы создаются под директорией `lib`, для классов таблиц (*Result*) подключается модуль `InflateColumn::DateTime`, который позволяет автоматически преобразовывать данные типа `datetime` при извлечении в объект `DateTime`, также задаётся пространство имён `Web::Log::Schema` и указывается стандартная строка подключения к существующей базе данных (`dsn`).

В результате создаются все необходимые классы, рассмотрим их по порядку. Глав-

ный класс схемы Web::Log::Schema:

```
use utf8;
package Web::Log::Schema;

use strict;
use warnings;

use base 'DBIx::Class::Schema';

__PACKAGE__->load_namespaces;

1;
```

Данный класс является наследником DBIx ::Class::Schema. Метод load_namespaces производит загрузку всех прочих классов таблиц в пространстве имён Web::Log:: Schema::Result.

Класс Web::Log::Schema::Result:: Author представляет таблицу authors.

```
use utf8;
package Web::Log::Schema::Result
    ::Author;

use strict;
use warnings;

use base 'DBIx::Class::Core';

__PACKAGE__->load_components("
    InflateColumn::DateTime");
__PACKAGE__->table("authors");
__PACKAGE__->add_columns(
  "authorid",
  { data_type => "integer",
     is_auto_increment => 1,
     is_nullable => 0 },
  "author",
  { data_type => "varchar",
     is_nullable => 0, size =>
     255 },
);

```

```
18 __PACKAGE__->set_primary_key("
     authorid");
19
20 __PACKAGE__->
     add_unique_constraint("
     author_unique", ["author"]);
21
22 __PACKAGE__->has_many(
23   "comments",
24   "Web::Log::Schema::Result::
       Comment",
25   { "foreign.authorid" => "self.
       authorid" },
26   { cascade_copy => 0,
       cascade_delete => 0 },
27 );
28
29 __PACKAGE__->has_many(
30   "posts",
31   "Web::Log::Schema::Result::Post
       ",
32   { "foreign.authorid" => "self.
       authorid" },
```

```
33    { cascade_copy => 0,
        cascade_delete => 0 },
34 );
35
36 1;
```

Метод `table` указывает название таблицы, метод `add_column` задаёт поля таблицы и их атрибуты. Метод `set_primary_key` задаёт первичный ключ таблицы, а метод `add_unique_constraint()` позволяет указать, что поле `author` —– уникальное. С помощью метода `has_many` мы можем выразить отношения между таблицами. В данном случае указывается, что от данной таблицы зависят две другие таблицы `comments` и `posts`, имеющие внешний ключ на поле `authorid`.

Класс `Web::Log::Schema::Result::Post` представляет таблицу posts:

```
use utf8;
package Web::Log::Schema::Result
    ::Post;

use strict;
use warnings;

use base 'DBIx::Class::Core';

__PACKAGE__->load_components("
    InflateColumn::DateTime");

__PACKAGE__->table("posts");

__PACKAGE__->add_columns(
  "postid",
  { data_type => "integer",
     is_auto_increment => 1,
     is_nullable => 0 },
  "post",
  { data_type => "text",
     is_nullable => 1 },
  "postime",
```

```
    {
      data_type     => "datetime",
      default_value => \"
         current_timestamp",
      is_nullable   => 0,
    },
    "authorid",
    { data_type => "",
       is_foreign_key => 1,
       is_nullable => 1 },
);

__PACKAGE__->set_primary_key("
    postid");

__PACKAGE__->belongs_to(
    "authorid",
    "Web::Log::Schema::Result::
       Author",
    { authorid => "authorid" },
    {
      is_deferrable => 0,
      join_type     => "LEFT",
```

```
    on_delete     => "NO ACTION",
    on_update     => "NO ACTION",
  },
);

__PACKAGE__->has_many(
  "comments",
  "Web::Log::Schema::Result::
     Comment",
  { "foreign.postid" => "self.
     postid" },
  { cascade_copy => 0,
     cascade_delete => 0 },
);

1;
```

По сравнению с предыдущим классом здесь появляется метод `belongs_to`, который отражает отношения между таблицей `posts` и `authors`: использование внешнего ключа `authorid`. По смыслу `belongs_to`

противоположен has_many.

Последний класс Web::Log::Schema::Result::Comment представляет таблицу comments (аналогичен предыдущим, поэтому листинг не приводится).

Второй вариант создания классов — ручной. В этом случае все четыре класса пришлось бы писать в ручную. Но с другой стороны это позволило бы быстро создать базу данных, не используя ни строчки SQL-кода с помощью метода deploy:

```
1 use Web::Log::Schema;
2
3 Web::Log::Schema->connect('dbi:
    SQLite:weblog.db')->deploy;
```

Это открывает возможность для прозрачной миграции приложения на произвольную базу данных, сменив лишь одну

строку подключения!

Подключение к базе данных

В приложении потребуется делать подключение к базе данных. Поскольку процесс приложения будет постоянно присутствовать в памяти и последовательно обрабатывать запросы, то выгоднее иметь одно постоянное подключение к базе данных, а не открывать и закрывать его на каждый запрос. В этом случае нам удобно создать специальный класс-синглтон для подключения к базе данных.

```
1 package Web::Log::DB;
2 use strict;
3 use warnings;
4 use Web::Log::Schema;
5
6 my $db;
```

```
my $db_file = $ENV{'WEB_LOG_DB'}
    || 'weblog.db';

# db singleton
sub db {
    $db ||= Web::Log::Schema->
        connect('dbi:SQLite:'.
        $db_file);
}

1;
```

Как видно, подключается модуль базы данных `Web::Log::Schema`, который автоматически выполняет загрузку всех модулей таблиц. Создаётся приватная переменная `$db` объекта базы данных. Первый вызов функции `db` этого модуля создаст подключение к базе данных, все последующие вызовы в рамках одного процесса будут возвращать существующее подключение.

RESTful API

Воспользуемся популярной парадигмой *REST* при проектировании API веб-приложения и создадим перечень ресурсов и операций над ними, которые будут соответствовать задачам нашего приложения. Такой подход упрощает разработку, поскольку позволяет разбить сложную задачу на несколько независимых простых подзадач, позволяя в будущем легко дополнять и развивать интерфейс.

Сначала создадим модуль контроллера API веб-приложения `Web::Log::API`:

```
1 package Web::Log::API;
2 use Dancer ':syntax';
3 use Web::Log::DB;
4
5 prefix '/api';
6 set serializer   => 'JSON';
```

```
7 set content_type => 'application/
    json';
8 ...
```

Для тех, кто не знаком с фреймворком Dancer, поясню, что создаётся модуль контроллера, URI всех маршрутов которого будет иметь префикс `/api`. Все возвращаемые Perl-структуры данных будут проходить сериализацию в формат `JSON` и будет устанавливаться заголовок типа данных `application/json`. Таким образом, мы заодно определяем и представление наших данных.

Авторы

Наше приложение достаточно простое и нам требуется иметь возможность созда-

вать новых авторов и получать список существующих авторов.

На каждое выполняемое действие определяется HTTP-ресурс:

1. Список авторов — http-запрос `GET /api/authors`
2. Добавление нового автора — http-запрос `PUT /api/authors/:author`

Реализуем код таких операций:

1). Список авторов

```
get '/authors' => sub {
    my $db = Web::Log::DB->db;
    my $rs = $db->resultset('
       Author');
    [ $rs->get_column('author')->
       all ];
```

```
5 };
```

В первую очередь в маршруте получаем объект подключения к базе данных — `$db`. Затем применяется метод `resultset` для получения одноимённого объекта. *ResultSet* можно отождествить с запросом к заданной таблице базы данных для получения результата. Важно понимать, что сам по себе вызов метода `resultset` не приводит к выполнению SQL-запроса, он лишь формирует необходимые данные для выполнения такого запроса. Обратите внимание, что под значением аргумента `resultset Author` имеется ввиду не имя таблицы в базе данных, а имя класса `Web::Log::Schema::Result::Author`.

Последующий вызов `get_column` уточняет, какой столбец нас интересует для извлечения, ну а метод `all` уже непосред-

ственно формирует и исполняет запрос к базе данных и возвращает обычный массив с результатами выборки. Можно сказать, что данная последовательность вызовов эквивалентна одному SQL-запросу:

```
SELECT author FROM authors
```

2). Следующий маршрут для добавления нового автора

```
put '/authors/:author' => sub {

    my $author = param('author');
    my $db = Web::Log::DB->db;
    my $rs = $db->resultset('
       Author');

    eval {
        $rs->create( { author =>
           $author } );
    };

```

```
11      { status => $@ ? 'failure' :
           'success' };
12 };
```

Получив нужное значение имени автора из параметра в *URI*, выполняется запрос на создание записи в базе данных с помощью метода create, который принимает в качестве первого аргумента хэш со списком пар столбец/значение. Это эквивалентно такому SQL-запросу:

```
1 INSERT INTO authors(author)
     VALUES ($author)
```

Вызов происходит внутри `eval`, поскольку нам может потребоваться перехватить ошибку создания повторного или пустого имени, что вызовет исключение, т.к. в базе данных данный столбец объявлен ненулевым и уникальным.

Функция возвращает хэш с информацией о результатах операции создания записи: успешная или неудачная.

Также можно было бы добавить и операцию удаления, но это влияет на целостность данных, поскольку записи в других таблицах зависят от записей в этой таблице и нам нужно будет принимать решение, что делать с записями в других таблицах при удалении. Базы данных имеют средства для каскадного удаления связанных данных, но в случае форума или блога — это может быть нежелательным явлением. Поэтому на практике обычно заводят дополнительное поле в таблице, которое указывает, что автор *удалён* или *забанен* и т.д. Ну а данный пример сразу перестанет быть простым, если реализовывать подобное решение.

Посты Требуется создавать новые посты, получать один пост или список постов, причём желательно не все стразу, а *постранично*, т.е. небольшими блоками, а также иметь возможность обновить содержимое поста.

1. Один пост по указанному номеру — http-запрос `GET /api/posts/post:postid`
2. Список постов на заданной странице — http-запрос `GET /api/posts/page:page`
3. Добавление нового поста — http-запрос `POST /api/posts`
4. Обновление поста — http-запрос `PUT /api/posts/post:postid`

1). Создадим маршрут для извлечения заданного поста:

```
get '/posts/post:postid' => sub {
    my $postid = param('postid');
    my $db = Web::Log::DB->db;
    my $post = $db->resultset('
       Post')->find($postid);

    {
        postid  => $post->postid,
        postime => $post->postime
           ->datetime,
        post => $post->post,
        author => $post->authorid
           ->author,
    }
};
```

Для получения одного результата удобно использовать метод `find`, который в качестве первого аргумента может принимать значение первичного ключа записи. Возвращается объект *Row*, представляющий собой полученный ряд таблицы. Как видно,

чтобы получить актуальные значения для каждого столбца у данного объекта существуют одноимённые методы-акцессоры (они же будут и мутаторами, если нам захочется изменить значение столбца):

- `postid` возвращает значение столбца `postid`
- `postime` возвращает объект класса `DateTime` (поэтому для получения строкового представления применяется метод `datetime`)
- `post` возращает значение столбца `post`
- `authorid` возвращает *Row* `Author`, поскольку столбец является внешним ключом, и, для получения актуального имени автора, используется вызов акцессора `author`

Использование именованных акцессоров удобно, но иногда требуется просто получить «сырые» данные из базы данных без создания объектов. Такое возможно, если указать в параметрах поиска тип возвращаемого класса ‘DBIx::Class::ResultClass::HashRefInflator’:

```
my $post =
  $db->resultset('Post')
  ->find( $postid,
    { result_class => 'DBIx::
       Class::ResultClass::
       HashRefInflator' } );
```

В результате будет получена ссылка на хэш, в котором ключи — это названия столбцов, а значения — соответствующие значения столбца.

2). Список постов на заданной странице

```
get '/posts/page:page' => sub {
```

```
2
3       my $page = param('page');
4       my $db = Web::Log::DB->db;
5       [
6           $db->resultset('Post')->
                search(
7               undef,
8               {
9                   page         =>
                        $page,
10                  rows         =>
                        10,
11                  result_class => '
                        DBIx::Class::
                        ResultClass::
                        HashRefInflator
                        '
12              }
13          )->all
14      ];
15  };
```

Итак, в данном запросе мы используем метод `search` для выполнения поиска заданных записей. Нам нужны любые записи, поэтому первый аргумент — фильтр устанавливается в `undef`. Атрибут `page` задаёт удобный пейджер для выборки записей, начиная с определённой позиции, а параметр `rows` определяет сколько записей требуется извлечь. Метод `all` выполняет SQL-запрос к базе. Это аналогично подобному SQL-запросу:

```
SELECT * FROM posts LIMIT $row, (
    $page-1)*$row
```

Поскольку в результате такого запроса `DBIx::Class` вернёт коллекцию объектов, а нам требуется «сырые» данные в виде массива хэшей, то указывается параметр `result_class` со значением `DBIx::Class::ResultClass::HashRefInflator`, который выполнит нужную нам операцию и

не будет создавать объекты из полученных данных.

Чтобы продемонстрировать возможности `DBIx::Class` по созданию сложных запросов, можно несколько усложнить задачу и выдавать не просто список из десяти постов на странице, но и заодно указать, сколько комментариев есть к каждому посту. В этом случае нам придётся воспользоваться операцией объединения `JOIN`, чтобы получить выборку, содержащую данные из двух таблиц. Вот так будет выглядеть финальный запрос:

```
$db->resultset('Post')->search(
    undef,
    {
        join          => 'comments
            ',
        '+select'     => [ { count
            => 'commentid' } ],
```

```
6           '+as'          => [qw(
               comments_count)],
7           group_by       => ['me.
               postid'],
8           page           => $page,
9           rows           => 10,
10          result_class => 'DBIx::
               Class::ResultClass::
               HashRefInflator'
11      }
12  )->all
```

Как видно появился параметр `join`, который указывает на таблицу, которую мы подключаем в результирующий набор. Поскольку мы определили отношения между таблицами, `DBIx::Class` знает по какому столбцу происходит объединение. Параметр `+select` определяет, какие дополнительные столбцы будут присутствовать в запросе, в данном случае мы задаём колонку, где будут содержаться результат

функции COUNT(), которая просуммирует количество комментариев для каждого поста. А чтобы нам было удобно ссылаться в полученных результатах на данный столбец, мы задаём этому столбцу псевдоним comments_count с помощью параметра +as. Поскольку мы используем агрегирующую функцию, необходимо применить GROUP_BY, чтобы запустить подсчёт комментариев — это выполняется с помощью параметра group_by => ['me.postid'], который будет группировать выборку по столбцу me.postid (me – это общепринятый в DBIx::Class синоним текущей таблицы при формирования SQL-запроса)

Такой код сформирует следующий SQL-запрос:

```
1 SELECT me.postid, me.post, me.
    postime, me.authorid, COUNT(
    commentid )
```

```
2 FROM   posts me LEFT JOIN
     comments comments ON comments.
     postid = me.postid
3 GROUP BY me.postid
4 LIMIT  $row, ($page-1)* $row
```

3). Для добавления нового поста создадим такой маршрут:

```
1 post '/posts' => sub {
2
3     my $author = param('author');
4     my $post   = param('post');
5     my $id;
6
7     eval {
8         my $post_ref = Web::Log::
              DB->db->resultset('
              Post')->create(
               {
10                 post     => $post
                       ,
```

```
11                      authorid => {
                           author =>
                           $author }
12                  }
13              );
14              $id = $post_ref->id;
15          };
16
17          { status => $@ ? 'failure' :
               'success', id => $id };
18  };
```

В маршрут передаются два параметра: автор и, собственно, сам пост. Поскольку в таблице `posts` задаётся не имя автора, а его первичный ключ, то предварительно требуется получение значения этого ключа по имени. Как видно, в `DBIx::Class` это реализовано элегантно в виде подзапроса к таблице внешнего ключа:

```
SELECT authorid, author FROM
    authors WHERE author= "$author
    "
INSERT INTO posts ( authorid,
    post) VALUES ( "$post", "
    $authorid" )
```

У данного решения есть побочный эффект: если указанного автора раньше не существовало, то он будет создан. Чтобы избежать этого, можно попробовать действовать по-другому:

```
my $post_ref =
  Web::Log::DB->db->resultset('
     Author')->find( { author =>
     $author } )
  ->create_related( posts => {
     post => $post } );
```

В данному случае мы сначала ищем автора и затем с помощью метода `create_related`

создаём в зависимой таблице `posts` новый пост. В случае, если автор не найден, то `find` вернёт `undef`, и метод создания `create_related` не выполнится (произойдёт исключение).

Если пост создан успешно, то может быть полезно вернуть информацию о том, какой номер первичного ключа получил данный пост. Для этого используется метод `id()`. Теперь фронтенд, зная номер поста, может, например, выполнить переход на его страницу и т.п.

4). Реализуем маршрут для обновления содержимого поста

```
put '/posts/post:postid' => sub {
    my $post   = param('post');
    my $postid = param('postid');

    eval {
```

```
6           Web::Log::DB->db->
               resultset('Post')->
               find($postid)
7             ->update( { post =>
                 $post } );
8       };
9       { status => $@ ? 'failure' :
           'success' };
10  };
```

Изменить поле `post` можно напрямую через метод `update()`, либо предварительно воспользовавшись мутатором `post`:

```
1  my $p = Web::Log::DB->db->
       resultset('Post')->find(
       $postid);
2  $p->post($post);
3  $p->update();
```

Комментарии Требуется создавать комментарии, получать список комментариев для заданного поста и удалять комментарии:

1). Создание комментария

```
post '/comments/post:postid' =>
    sub {

    my $author  = param('author');
    my $comment = param('comment');
    my $postid  = param('postid');
    my $id;

    eval {
        $id =
          Web::Log::DB->db->
             resultset('Author')
          ->find( { author =>
```

```
                $author } )
12            ->create_related(
13              comments => {
14                  {
15                      comment =>
                           $comment,
16                      postid  =>
                           $postid,
17                  }
18              }
19            )->id();
20      };
21
22      { status => $@ ? 'failure' :
           'success', id => $id };
23  };
```

2). Получение комментариев к посту

```
1  get '/comments/post:postid' =>
       sub {
2       my $postid = param('postid');
3       [
```

```
        Web::Log::DB->db->
            resultset('Comment')->
            search(
             { postid => $postid
                },
             {
                 order_by => '
                    commentid',
                 result_class => '
                    DBIx::Class::
                    ResultClass::
                    HashRefInflator
                    '
             }
         )->all
    ];
};
```

3). Удаление комментария

```
del '/comments/comment:commentid'
    => sub {

```

```
3       my $commentid = param('
           commentid');
4
5       eval {
6           Web::Log::DB->db->
               resultset('Comment')
7             ->find($commentid)->
                 delete();
8       };
9
10      { status => $@ ? 'failure' :
           'success' };
11  };
```

Запуск приложения

Полученное API бэкенда реализует так называемые *CRUD*-операции (Create, Read, Update, Delete — Создание, Чтение, Обновление, Удаление) над базой данных.

Основной же класс веб-приложения будет выглядеть так:

```
1  package Web::Log;
2  use Dancer ':syntax';
3  use Web::Log::API;
4
5  our $VERSION = '0.1';
6
7  prefix undef;
8  set content_type => 'text/html';
9
10 get '/' => sub {
11     template 'index';
12 };
13
14 true;
```

Здесь подключается модуль API — `Web::Log::API` и задаётся маршрут корня веб-приложения. Запускается веб-приложение с помощью команды

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

Таким образом, работа над бэкендом завершена, отображение данных может быть реализовано на фронтенде с помощью различных фреймворков (например, *Backbone.js*), использующих API бэкенда для извлечения и отображения необходимых данных.

Заключение

В статье был рассмотрен пример создания бэкенда веб-приложения, использующего `DBIx::Class` для работы с базой данных через объектный интерфейс. Разъяснены ключевые понятия *Schema*, *ResultSet* и *Row*, а также типичные CRUD-операции с помощью объектных методов `DBIx::Class`.

По понятным причинам этот обзор не может охватить весь спектр возможностей `DBIx::Class`, поэтому рекомендуется за дополнительной информацией всегда обращаться к первоисточнику — документации модуля `DBIx::Class`. Также действует почтовый список рассылки и irc-канал `#dbix-class` на сервере irc.perl.org, где можно смело задавать интересующие вопросы.

■ *Владимир Леттиев*

4 Секретные операторы Perl и не только

Рассмотрены интересные неочевидные конструкции Perl.

Как известно, язык программирования Perl очень выразителен и имеет в своем арсенале множество средств, которые позволяют выразить намерения программиста в коде множеством совершенно разных образов. Также, в виду весьма хитрого синтаксиса, некоторые комбинации операторов могут приводить к интересным эффектам.

Также, некоторые вещи можно использовать не совсем стандартным образом для получения желаемого результата. Такие действия иногда называются «забивать гвозди микроскопом». Именно этим мы и будем заниматься.

Прежде чем читать эту статью дальше, необходимо понимать несколько вещей:

- Поведение операторов, приведенных в этой статье, может меняться от версии к версии Perl.
- Данные операторы, скорее всего, не предназначены для использования их в production.
- Большинство этих операторов были созданы людьми, которым приносит удовольствие исследовать любимый язык.
- Все, что приведено ниже и названо операторами, на самом деле ими не является.

Поехали.

Оператор «Венера»

```
0+
+0
```

Название

Свое название оператор получил от внешней схожести с символом Венеры.

Что делает?

Приводит аргумент слева, или справа, в зависимости от версии, к числовому виду. Например:

```
print 0+ '23a';
print 0+ '3.00';
print 0+ '1.2e3';
print 0+ '42 EUR';
```

```
print 0+ 'two cents';
```

Результат:

```
23
3
1200
42
0
```

Этот оператор с натяжкой *можно* использовать у себя в проектах, но не следует забывать о том, что с точки зрения Perl числами, например, являются: **0 but true 0E0** и некоторые еще хитрые константы. Еще следует отметить, что `0+` это метод, используемый для числового преобразования по умолчанию при использовании `overload`.

«Черепашка», «Детская коляска» или «Тележка из супермаркета»

```
@{[ ]}
```

Название

Свое название оператор получил из-за внешней схожести с черепахой или коляской. Примечателен тем, что был открыт Ларри Уоллом в 1994 году.

Что делает?

Это так называемый контейнерный оператор, который позволяет производить интерполяцию массива внутри строки. Элементы массива в результате будут разделены содержимым переменной $".

Как работает?

Сначала содержимое `[ ]` принудительно вычисляется в списковом контексте, затем, незамедлительно, проводится разыменование (`@{ }`).

Пример:

```
print "Test i am @{[ die()]}";
print "here";
```

Результат работы:

```
Died at demo.pl line 1.
```

Этот оператор можно использовать для выполнения произвольного кода в строках, когда происходит интерполяция. Например, с его помощью можно сделать некоторые вещи проще, например, построение SQL-запросов:

```
my $sql = "SELECT id, name,
    salary
     FROM employee WHERE id IN (@
        {[ keys %employee ]})
     SQL
";
```

но выигрыш в пару строк кода не оправдывает потенциальные проблемы в безопасности (хорошая практика в SQL — использование т.н. bind variables, которые будут подставлены при prepare). В виду неочевидности, использовать оператор в production не стоит, потенциальная уязвимость и уменьшение читабельности.

Bang Bang

```
!!
```

Данный оператор использовался еще тогда, когда Perl не было, его часто использовали программисты на C.

Как работает?

Этот оператор делает следующую элементарную вещь — двойное отрицание, суть которого сводится к булевому преобразованию. Вспоминаем, что булевых типов в Perl нет.

```
my $true  = !! 'a string';
my $false = !! undef;
```

В результате `$true` содержит `1`, а `$false` пустую строку `''`. Данный оператор можно использовать для проверки, есть ли значение, например, в ссылке на хеш, например

```
!! $hash->{value} or die("Missing
    value");
```

Червяк

```
1 ~~
```

Название?

Просто и незатейливо похож на гусеницу-пяденицу, и на многих червей также.

Как работает?

Оператор ~~ внешне очень похож на smart matching, но им не является потому, что это *унарный* оператор.

> Унарная операция — это операция над одним операндом

Тогда как smart matching — операция бинарная (операция над двумя операндами, сложение, например).

Что делает?

Всего-лишь сокращенное на четыре (!) символа ключевое слово `scalar`.

Пример:

```
perl -Esay~~localtime
Tue Jul 30 17:43:16 2013
```

Принцип действия оператора схож с оператором Bang Bang (`!!`), но отличается тем, что в Perl оператор ~ операторозависимый. Более подробно можно узнать, посмотрев в документации про побитовые операции в Perl. Применять этот оператор можно, но

следует быть осторожным, неизвестно, как он будет себя вести на всех версиях perl.

Червяк-на-палочке

```
1 —~
2 ~—
```

Это высокоприоритетный оператор инкремента/декремента. —~ инкрементирует только числа, которые **меньше** нуля, а ~— декрементирует числа **больше** нуля

Приоритет этого оператора выше арифметических операторов, кроме возведения в степень (**). Например:

```
1 $y = ~—$x * 4;
```

Будет исполняться идентично:

```
$y = ($x–1)*4;
```

Но не как:

```
$y = ($x * 4) –1
```

Для того, чтобы данные операторы работали с беззнаковыми типами данных, необходимо использовать прагму `integer` (`use integer`). Этот оператор работает весьма неплохо и его можно применять в production, но делать этого не стоит, т.к. на нестандартных архитектурах его поведение может отличаться от вышеуказанного. Хотелось бы попробовать его в действии на ARM-платформах, но автор статьи не располагает подобным устройством.

Космическая станция

```
–+–
```

Открыт Alistair McGlinchy в 2005 году.

Этот оператор производит высокоприоритетное приведение к числовому виду, по поведению он похож на оператор «символ Венеры», но он отличается следующими вещами. Оператор Венеры (`0+` или `+0`) использует *бинарный* оператор `+`, тогда как «Космическая станция» использует сконкатенированные три унарных оператора, а потому имеет более высокий приоритет.

Также, стоит помнить, что этот оператор имеет меньший приоритет, чем сконкатенированные операторы `*` и `x`. Принципы его работы можно проиллюстрировать следующим примером, в котором мы попробуем распечатать приведенное к числовому виду `'20GBP'` три раза:

Неправильно, т.к. возвращает преоб-

разованный вариант от строки '20 GBP20GBP20GBP':

```
print 0+ '20GBP' x 3; # 20
```

Неправильно, т.к. эквивалентно (print '20')x 3:

```
print( 0+ '20GBP' ) x 3; # 20
```

Правильно, но сильно длинно, сильно «лисповo»:

```
print( ( 0 + '20GBP' ) x 3 ); # 202020
```

Правильно — используя оператор «Космическая станция»:

```
print –+– '20GBP' x 3; # 202020
```

Однако, т.к. унарный минус – и унарный + просто заменяет результат строки на его

значение, то данный оператор **не будет** работать со следующего вида строками:

- С теми, что начинаются на —.
- С теми, что начинаются на *не числовой* символ.

Тогда как оператор Венеры работает во всех этих случаях, но имеет меньший приоритет.

Goatse или Сатурн-оператор

```
=( )=
```

Дабы не травмировать ничью психику, автор статьи не рекомендует искать в интернете первое название оператора, если оно вам кажется незнакомым. Правда,

не надо, вам не понравится.

Что делает?

Этот оператор вводит списковый контекст справа от него и возвращает количество элементов слева.

Как работает?

Список в скалярном контексте возвращает количество элементов. Не имеет значения, сколько элементов из них было присвоено переменным. В таком случае правая часть выражения будет приведена к пустому списку, а затем, следовательно, отброшена.

Пример:

Посчитать количество слов в $_:

```
$n =()= /word1|word2|word3/g;

$n =()= "abababab" =~ /a/; # $n =
    1

$n =()= "abababab" =~ /a/g; # $n
    = 4
```

К тому же, данный оператор является контейнерным (!), что позволяет запросто «втянуть» в него результат правой части. Это значит, что мы можем поступать следующим образом:

```
$n =($b)= "abababab" =~ /a/g; #
    $n = 4; $b = 'a'

$n =(@c)= "abababab" =~ /a/g; #
    $n = 4; @c = qw( a a a a )
```

Следующий «хитрый» пример его исполь-

зования, похоже, имеет право на жизнь, но существует другой секретный оператор, который может делать то же самое.

Допустим, мы хотим узнать, на сколько частей разобьет строку split, а сами элементы нас не интересуют, может быть, есть смысл попробовать нечто следующего вида:

```
1 my $count = split /:/, $string;
```

Этот пример вернет нам необходимое число, но при этом ругнется на:

```
1 Use of implicit split to @_ is
    deprecated
```

Для того, чтобы решить данную проблему мы можем воспользоваться данным оператором и написать нечто подобное:

```
1 my $count =()= split /:/, $string
    ;
```

Что не вызовет warning, но при этом будет всегда возвращать 1 потому, что `split` **никогда** не разбивает строку на большее количество частей, чем необходимо. В свою очередь, компилятор расценивает попытку сохранить в `()` как утверждение в том, что данные элементы нам не надо и вернет неизмененную строку, что будет приведено к списку с одним элементом.

Есть два возможных решения.

Первое заключается в том, что мы можем запретить компилятору проводить оптимизацию `split()`, указав при помощи -1 желание получить бесконечное количество кусков строки:

```
my $count =()= split /:/, $string
    , -1;
```

Или использовать другой секретный опера-

тор, «черепашку»:

```
my $count = @{[ split /:/,
    $string ]};
```

Воздушный змей

```
~~<>
```

На самом деле, данный оператор является всего-лишь комбинацией Червяка и <>. Он предоставляет скалярный контекст для операции `readline()`, но полезный он только в контексте списка.

Богато украшенный двусторонний меч

```
<<m=~m>> m ;
```

Этот секретный оператор предоставляет мультистрочные комментарии и не более того. Пример использования:

```
<<m=~m>>
    Use the secret operator on
       the previous line.
    Put your comments here.
    Lots and lots of comments.

    You can even use blank lines.
    Finish with a single
m
;
```

Но следует учитывать тот факт, что данный комментарий — просто строка, заключенная в двойные кавычки, а потому может

иметь некоторые побочные эффекты.

Отверточные операторы

Обнаруженный Дмитрием Карасиком в процессе поиска операторов, базирующих-ся на !. Как и отвертки, эти операторы бывают четырех основных типов, но с разной длиной рукоятки:

Прямая отвертка — обеспечивает декремент по условию:

```
$x —=!! $y
# $x—— if $y;

$x —=!  $y
# $x—— unless $y;
```

Крестовая отвертка — инкремент по условию:

```
$x +=!! $y;
# $x++ if $y;

$x +=!  $y;
# $x++ unless $y;
```

Отвертка-звездочка — сброс переменной в 0 по условию:

```
$x *=!! $y;
# $x = 0 unless $y;

$x *=!  $y;
# $x = 0 if $y;
```

Крестообразная отвертка-шлиц — сброс переменной в ” по условию:

```
$x x=!! $y;
# $x = '' unless $y;

$x x=!  $y;
# $x = '' if $y;
```

Enterprise-оператор

```
()x!!
```

Довольно часто возникает необходимость добавить элемент в список по условию. Это можно сделать следующим образом:

```
my @shopping_list = ('bread', '
    milk');
push @shopping_list, 'apples'
    if $cupboard{apples} < 2;
push @shopping_list, 'bananas'
    if $cupboard{bananas} < 2;
push @shopping_list, 'cherries'
    if $cupboard{cherries} < 20;
```

```
5 push @shopping_list, 'tonic'
      if $cupboard{gin};
```

Но при помощи этого оператора можно сделать следующим образом:

```
1 my @shopping_list = (
2      'bread',
3      'milk',
4      ('apples'   )x!! ( $cupboard{
          apples} < 2 ),
5      ('bananas'  )x!! ( $cupboard{
          bananas} < 2 ),
6      ('cherries' )x!! ( $cupboard{
          cherries} < 20 ),
7      ('tonic'    )x!! $cupboard{
          gin},
8 );
```

Скобки необходимы из-за возможных проблем с приоритетом выполнения.

Секретные константы

Космический флот

```
<=><=><=>
```

На первый взгляд эта конструкция похожа на космический корабль, но на самом деле, действительно, космическим кораблем является только средний <=>. Крайние корабли — вызов `glob("=")`. Константа = 0.

Двуходка

```
<~>
```

На операционных системах семейства Unix — реальный путь к домашнему каталогу пользователя, на Windows — значение переменной окружения `$ENV{HOME}`.

Еще раз напоминаю, что большинство данных операторов были придуманы или исключительно шутки ради, или ради изучения механизмов языка.

■ *Дмитрий Шаматрин*

5 Обзор CPAN за июль 2013 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц.

В этом месяце стала заметна активность по исправлению бага #RT116479, были обновлены множество CORE-модулей, которые

теперь спокойно могут быть инсталлированы в каталоге модулей `sitelib`.

Статистика

- Новых дистрибутивов — 197
- Новых выпусков — 866

Новости веб-фреймворкостроения

Как и было ранее обещано, сначала краткий обзор урожая веб-фреймворков этого месяца.

Новые веб-фреймворки:

- Yeb Новый веб-фреймворк, специально заточенный для создания сайтов конференций Perl, базирующийся на новом поколении программ Act — YACT (Yet Another Conference Toolkit).
- Puncheur По примеру в описании модуля его трудно отличить от Mojolicious. Фреймворк базируется на `Plack`. В описании честно сказано об «ALPHA QUALITY» продукта.
- Jedi Фреймворк для настоящих джедаев. Никакого DSL в синтаксисе, построено на основе `Moo`, выходной формат приложения — `psgi`, что позволяет запускать его на любом psgi-совместимом веб-сервере.

Обновлённые веб-фреймворки:

- Nephia 0.37
- Web-Reactor 0.04
- Kelp 0.4011
- Mololicious 4.22
- HiD 0.4
- Kossy 0.17
- Dancer 1.3117
- Dancer2 1.06
- Cot 0.09

Новые модули

- App-Module-Setup Ещё одна утилита для генерации файлов при создании нового модуля. Особенностью данной утилиты является поддержка

файлов конфигурации и системы шаблонов, позволяющая очень гибко настраивать получаемый результат.

- UnQLite Модуль является обвязкой к `UnQLite` — встраиваемой транзакционной NoSQL-базой данных. `UnQLite` может быть использована для хранения документов (как `MongoDB`, `Redis` и `CouchDB`), так и как обычный DBM (как `BerkeleyDB`, `LevelDB`).

- Canella Простая система для развёртывания программного обеспечения. Задачи и конфигурация описываются с помощью специального DSL, а утилита `canella` позволяет позволяет запускать на выполнение требуемые задачи.

- Geo::JSON Модуль, реализующий спецификацию GeoJSON — формата, предназначенного для кодирования

различного рода географических данных для межсистемного обмена.

- IO::Vectored Интерфейс к системным вызовам чтения `readv(2)` и записи `writev(2)` для векторных операций ввода/вывода. Модуль позволяет выполнять атомные операции записи из различных буферов (не делая предварительного их объединения) и читать в буферы, расположенные непоследовательно в памяти. Это позволяет экономить на системных вызовах и быть уверенным, что операция будет выполнена атомно (исключается вклинивание другого процесса при выполнения транзакции).

- App-Zapzi Утилита командной строки, которая позволяет преобразовывать текстовые файлы, файлы в форматах markdown и html (как локальные,

так и с удалённых веб-серверов), в формат Ebook MOBI, которые можно удобно прочесть позже на устройстве для чтения электронных книг.

- App::Chorus Утилита для преобразования markdown-файла в веб-приложение с презентацией.

- Data::Dumper::GUI Графический интерфейс к `Data::Dumper`, позволяющий отображать вывод `Data::Dumper` в виде древовидной структуры, с возможностью раскрытия/скрытия вложенных узлов.

- Exporter::Lexical Модуль позволяет экспортировать лексические подпрограммы из вашего модуля, т.е. экспортированные подпрограммы будут видны только в той области видимости, в которой выполняется

‘use’ модуля. Работает только в Perl >= 5.18.

- POD2::RU Документация о Perl на русском языке.

Обновлённые модули

- IO::Socket::SSL 1.953 Обновлённая версия `IO::Socket::SSL` имеет важное изменение в поведении: по умолчанию значение опции `ssl_verify_mode` стало `verify_peer`, вместо `verify_none` для клиента, а также путь к ключу и сертификату теперь не имеет значения по умолчанию. Будьте внимательны при обновлении, код, который рассчитывал на прежние значения по умолчанию, окажется

сломанным.

- Try::Tiny 0.16 После длительного перерыва вышло заметное обновление для `Try::Tiny` с исправлением множества багов и небольшим ускорением в работе модуля.

- Storable 2.45 Устранены проблемы с утечкой памяти и `SIGSEGV` в фазе глобального уничтожения. Кроме того в документацию модуля добавлено важное предупреждение безопасности о том, что `Storable` не может быть использован при обработке данных, полученных из ненадёжного источника, поскольку при десереализации возможно выполнение кода, который содержится в полученных данных.

- Eval::Closure 0.11 В новом релизе модуля для безопасного создания

замыканий через eval строковых выражений появилась поддержка лексических подпрограмм (доступных в Perl 5.18).

- Moo 1.003000 Вышел новый релиз минималистичного ООП-фреймворка `Moo` с исправлениями ошибок в коде и документации.

- Perl::Tidy 20130717 В новой версии модуля для форматирования исходного кода исправлены ошибки сборки на bleed Perl >= 5.19

- SOAP::Lite 1.02 После затяжной серии версий 0.71X наконец-то вышел первый мажорный релиз `SOAP::Lite` с исправлением ошибок.

- Underscore 0.03 В новом релизе `Underscore`, портированной на Perl

библиотеки `Underscore.js`, реализованы новые функции и произошла синхронизация API с `underscore.js` версии 1.4.3.

- App::cpangrep 0.04 Обновилась утилита для поиска по исходному коду модулей на CPAN (через веб-сервис grep.cpan.me). В новой версии улучшена поддержка прокси.

- CHI 0.58 Вышла новая версия унифицированного интерфейса к управлению кэшем. В новой версии произошла миграция с `Moose` на `Moo`.

- Template::Toolkit 2.25 После долго перерыва обновлён `Template::Toolkit` с исправлением ошибок, включая корректную работу на новых версиях Perl >= 5.18.

- MongoDB 0.702.0 Новый релиз драйвера `MongoDB` для Perl получил экспериментальную поддержку аутентификации Kerberos на Linux и восстановил совместимость с версиями Perl 5.8.x

■ *Владимир Леттиев*

6 Интервью с брайаном ди фоем про будущее. Часть 2

25 и 26 мая в Варшаве прошел первый польский перл-воркшоп. На него приехал и брайан ди фой (brian d foy), который, по его словам, всегда старается побывать на мероприятии в городе, где он еще не был. Брайан уделил нашему журналу час и ответил на вопросы о том, каким он видит будущее: будущее книг, перла, его синтаксиса и сообщества. В этом номере — вторая и заключительная часть интервью (публикуется с сокращениями).

Perl 7

— Вы, разумеется, читали недавние обсуждения про Perl 7. Мы находимся в ситуации, когда Perl 5 развивается, но

при этом люди, не знакомые близко с перлом, считают его устаревшим просто потому, что он был выпущен 20 лет назад, и у нас до сих пор пятая версия. Является ли это проблемой, и если да, как ее решить? Нужно ли выпустить Perl 7, пропустив версию 6, или надо сначала доделать и выпустить Perl 6? Проблема не столько в смене версии, а в том, чтобы убедить окружающих, что перл жив.

— Проблема с названиями действительно непростая. Итак, у нас есть Perl 5, и каждый знает, что это. Далее, мы думали, что будет что-то под названием Perl 6, и все, работающие с Perl 5, перейдут на Perl 6. Этого не случилось. Не важно, по какой причине.

Не думаю, что какое-либо название исправит это. Мне кажется, это введет в еще большее заблуждение. Если мы сделаем Perl 7,

то придется сказать, что Perl 6 умер. Это не Perl 5, это какая-то следующая версия Perl 6. Остальные названия вроде Pumpkin Perl и другие, которые упоминались, на мой взгляд приведут к расщеплению, которое произошло со многими другими проектами, и никто не знает, что там использовать. То ли Perl 5, то ли perl5i, то ли Perl 6, то ли Perl 7. Никто в этом не разберется, если на это не обращать внимания, и вот это проблема.

Такие вещи могут работать, только если быть внимательным, но дело в том, что большинство как раз не такие. Поскольку те, кто в курсе, не спутают Perl 5 и Perl 6, они знают, что там происходит.

Проблема остается на месте, потому что мы продолжаем делать из этого проблему. На польском Perl-воркшопе был хороший

доклад про бизнес и Perl. Суть в том, что никого не интересуют вещи, о которых мы беспокоимся. Желание у бизнеса только одно — чтобы работало. Например, люди пользуются Вордпрессом, потому что он решает задачу, а не потому что кто-то выбирает его из-за того, что он написан на PHP. Я сам пользуюсь Вордпрессом и понятия не имею, какая у меня версия PHP. Мне это неважно.

Если кому-то надо использовать Perl 5, чтобы запустить то, что написано на Perl 5, то им не важно, на какой версии это было создано. Это может быть 5i, Pumpkin, Strawberry или что-то другое.

С точки зрения программистов это выглядит немного по-другому. Они хотят нацеливаться на что-то конкретное. Но в тоже время надо понимать, что если вы хотите

создать что-то для других, то хотелось бы, чтобы это можно было бы запустить с минимальными усилиями.

Так что я не знаю. У меня нет одного-единственного ответа. Я думаю, что чем больше мы говорим об этом, тем меньше это кому-то поможет. Если мы просто прекратим об этом говорить, то никто не заметит. Чем больше мы обсуждаем проблему, тем больше она кажется.

У меня нет ответа, поэтому я и не участвовал в этой дискуссии.

— Как вы считаете, достаточно ли хорош нынешний Perl 5 (будь то 5.18 или 5.20 вместе с фичами, которые появились в 5.10), чтобы с него начинать?

— Мне кажется, что текущие версии перла

так же хороши для начала, как и, например, 5.6. Потому что все зависит от того, что ты за программист и какой у тебя опыт с другими языками. Есть люди, которые могут легко освоить новый язык, и они знают, что хотят. Если же ты только начинающий программист, то не думаю, что есть какая-то разница, потому что сначала придется научиться самым основам перла. И здесь неважно, 5.8 ли это, 5.10 или 5.20, потому что это в основном скаляры, массивы и хеши, где все одинаково. Не исключено, что до сложных возможностей и не дойдет. Есть какие-то тонкие синтаксические улучшения, но вряд ли они сильно важны.

Будущее CPAN

— В прошлом CPAN был одним из главных преимуществ (killer feature) перла. А так ли это сейчас? Во-первых, там много устаревших и плохих модулей, а во-вторых, есть GitHub, который может посоревноваться со спаном.

— Да, CPAN был нашей киллер-фичей. Если что-то было написано, оно наверняка было на CPAN. CPAN предлагает простой способ установить модуль, но теперь это сложнее, а люди становятся разборчивыми по отношению к скриптам установки модулей. Мы тут немножечко переусердствовали, как и в зависимостях. Иногда вытягивается полспана. Вот сегодня я что-то загрузил и попробовал установить, но это заняло около часа. За это время я, наверное, мог бы решить задачу на чистом перле. Тут нам надо

действовать всем вместе. Я бы с радостью посоветовал кому-то много разного привлекательного на CPAN, но я не хочу, чтобы они возились с установкой.

Хранить модули в разных местах — интересная идея. Например, иметь копию на CPAN для установщика типа cpanminus. Потом можно зайти на GitHub или в какой-нибудь частный репозиторий. Не знаю, правда, насколько это поможет. Иметь ссылки на GitHub интересно, но при этом надо быть готовым к возможным поломкам. CPAN мне нравится, потому что я могу с уверенностью сказать, что там лежит конкретный законченный релиз. Ну, на GitHub есть теги и все такое, можно и этим пользоваться, но все это начинает усложнять дело. Очень легко понять, что вот это — версия 1.23 на CPAN. Куда сложнее на гитхабе: «Ага, вот этот аккаунт, или,

погоди, нет, кто-то форкнул, и мне нужна именно та версия, а если он ее замерджит обратно, то и я переключусь обратно». Хорошо, конечно, иметь такую гибкость, но, по-моему, ключевая особенность CPAN в том, что там все хранится в едином месте.

На наших глазах другие языки пытались делать нечто подобное. И если там нет чего-то, похожего на CPAN или какого-то центрального хранилища, то заканчивалось это разговорами: “Вот здесь я смогу взять что угодно, мне не придется помнить все детали, и я не хочу помнить, что тот парень покинул проект, и я не смогу больше загрузить из его аккаунта на гитхабе, а нужный модуль надо поискать где-то в другом месте”. А на CPAN сразу можно выяснить, кто это написал, когда оно было зарелизено, какая версия последняя и так далее.

— В гитхабе можно работать над одним проектом совместно. Можно сделать pull request, например. А на CPAN такое не получится.

— Это верно. Возможность совместной работы в GitHub просто замечательна. Я могу зайти в проект на GitHub, форкнуть его, посмотреть на файл, нажать кнопку «Редактировать», прямо в браузере сделать правку, сохранить ее и отправить pull request, ничего не загружая к себе. Наверняка, это принесет мне в десяток раз больше патчей по сравнению со сценарием, когда надо сначала загрузить все к себе, сделать изменение, подготовить патч, отправить его по электронной почте, а потом дергать-дергать-дергать меня. А если у меня есть перед глазами pull request и похоже, что он работает, то — бам! — готово, и я иду дальше.

Но в какой-то момент, когда будет понятно, что получилось что-то осмысленное, можно сказать: «Да, это пора показывать людям, положу-ка я его на CPAN».

Интересно было бы полностью заиметь CPAN на GitHub, храня только ссылки на дистрибутивы, которые уже есть на GitHub, и просто скачивать оттуда каталог, а не копию репозитория. Если кто-то это сделает, я готов это продвигать. Кстати, для Perl 6 ведь была идея указывать при подключении модуля имя автора, версию и источник. Так что, если мы могли бы сделать это с гитхабом для Perl 5, был бы успех.

Будущее Perl

— Кроме CPAN преимуществами считаются регулярные выражения и тот факт, что перл идет почти с любым юниксом. Что еще? Удобная работа со строками. Но все это уже в прошлом, оно уже есть и известно. А можем ли мы добавить в перл что-то эдакое очень хорошее и полезное, чего еще не было? Ведь другие скриптовые языки в каком-то смысле похожи на перл. Может ли Perl предложить что-то действительно новое?

— Важная киллер-фича, которая уже сейчас есть в перле, —– это поддержка юникода. Вроде в 5.18 добавили экспериментальную фичу, которую я еще не пробовал, — операции со множествами в символьных классах. Это означает, что можно сделать символьный класс, который содержит

какой-то набор юникодных символов *минус* несколько других символов. Ну, сейчас это тоже можно делать с помощью юникодных свойств, но вот эта фича —– а Карл Вильямсон (Karl Williamson), по-моему, проделал фантастическую работу, чтобы добавить это в перл, —– последнее, чего не хватало для поддержки юникода на низком уровне, чтобы стать совместимым с рекомендациями консорциума Unicode. Я думаю, Perl может оказаться единственным языком, который совместим во всех деталях.

Проблема в том, что многие еще не программируют для юникода и по-прежнему пользуются Latin-1 или чем-то похожим. В США люди все еще думают в терминах ASCII и не слишком заботятся о юникоде. Мы должны всегда указывать, что вывод ведется в такой-то кодировке, а ввод про-

исходит в такой-то. В США был ASCII, в большинстве стран Европе — Latin-1, в России — что в России?

— Да штук пять было.

— Вот, пять разных кодировок, ОК. Так что вы наверное уже делаете все правильно.

Это реальная мощь, и Perl может ее предоставить.

Мне очень хотелось бы видеть вот что. Когда мы собрались на первую встречу по Perl 6, мы сидели вместе в комнате и начинали думать о том, что нам надо сделать что-то новое, потому что Perl 5 умирает (и об этом постоянно говорят все то время, что я занимаюсь перлом, хотя я вполне себе зарабатываю, так что не понимаю, о чем это). Так вот, что новое надо сделать? Ну, я сказал

что-то свое, другие тоже высказались, а Ларри —– я помню, он сидел за столом напротив — долго молчал и ничего не говорил. А потом сказал: «Распределенные вычисления». Вот это и было первым, что мы захотели сделать в Perl 6, а не какие-то интересные фичи или синтаксис. Он хотел, чтобы была возможность сказать: «Я хочу, чтобы эта задача разбилась на несколько маленьких, и язык сам должен решить, как это сделать».

В Perl 6 это, например, заложено в кросс-операторы, когда с одной стороны оператора набор данных и с другой стороны набор; оно все распараллеливается, а потом собирается и результат просто помещается на выход. Как это работает — неважно, оно просто работает. Мне бы такого очень хотелось. Это было бы по-настоящему круто.

Мне кажется, нам надо научиться чему-то из Erlang и Go, мы это вполне можем наверстать.

А еще было бы хорошо, наконец, разобраться с сообщениями об ошибках `Can't locate`. Это сообщение об ошибке должно выглядеть так: «Невозможно найти такой-то модуль, но если он вам нужен, то установить его можно так-то и так-то». Посмотрите на Stackoverflow, например. Думаю, вопрос номер один там — «Что значит сообщение об ошибке `Can't locate`?». В сообщении приводятся пути, которые не имеют никакого отношения к исходному коду. Люди видят большой фрагмент текста и не понимают, что с этим делать. Такой подход не решит задачу, но мне кажется, что перлом станут пользоваться чаще.

Будущее сообщества

— Есть еще одна вещь, связанная с перлом, — сообщество, которое, вроде как, непохоже на те, что есть у других языков. Когда-то давно вы основали PM-группы. Как вам кажется, нужны ли там какие-то организационные изменения? Работают ли группы сейчас?

— Это интересная тема. Давным-давно, году в 97-м или 98-м, мы вместе с Дейвом Адлером и Адамом Туроффым (Dave Adler, Adam Turoff) придумали нью-йоркских перл-монгеров. Мы возвращались с Оско… нет, тогда это, наверное, была The Perl Conference, потому что там был только перл. Возвратились оттуда в Нью-Йорк и немного позависали друг с другом. Я как раз туда перебрался, и мне хотелось там с кем-то познакомиться, а я понятия не имел,

где. Я работал с перлом, они тоже; у нас были общие темы для обсуждения, так что мы могли вместе посидеть и поговорить. Фильмы там, Доктор Кто. Если мне что-то требовалось, я мог спросить у тех, кто меня знал, например, что мне нужна работа или попросить помочь решить какую-то задачу или еще что-то.

О чем-то можно спросить в онлайне. Но мне по-прежнему кажется, что социальный аспект — самое главное. Но в то же время я понимаю, что у разных групп разные интересы. Вот, к примеру, Тим Маер (Tim Maher) из Сиэтла. Когда он создал группу, он хотел сделать это образовательным коллективом (Educational Collective), по-моему он так это называл *(На сайте seattleperl.org упоминается Educational Cooperative — А. Ш.)*, где он хотел обучать друг друга перлу. Он сделал так, чтобы были и пре-

зентации, и обучение, и оно там работало. Джим Кеннан (Jim Kennan) сделал что-то похожее в Нью-Йорке.

Мне нравится, что я могу поехать на конференцию почти куда угодно в мире и встретить там перловиков. Конференции делают те, кто создает пользовательские группы на местах. Вот, например, польский воркшоп организован Warsaw.pm. Если я еду в Лондон, могу выпить с London.pm. Если я оказался Амстердаме, то скорее всего появлюсь в пабе. Это возможно потому, что мы знаем друг друга. Мне не обязательно смотреть презентацию или делать технический доклад или что-то другое формальное. Я могу просто сказать: «Эй, мы знакомы, давай замутим что-нибудь». И мне кажется, что это действительно важно — знать друг друга. Если ты находишься внутри Perl-сообщества, то наверняка

знаешь кого-то в каждой временной зоне.

Мне кажется, что у перла самое большое число мероприятий по сравнению с другими языками. В то время как другие языки проводят по одному большому корпоративному мероприятию в год, может несколько поменьше, перловые мероприятия проходят почти каждую неделю. Про это пишут в блогах, записывают и выкладывают видео. Мы знаем, как выглядят люди, которые программируют то, чем мы пользуемся. Мы знаем их в лицо. Не знаю, могут ли другие сообщества сказать то же самое.

Не думаю, что надо что-то менять. Я пытался не вмешиваться. Я знаю, что был у истоков и мне посчастливилось там быть. Но я не сделал ничего особенного. Мне просто повезло быть первым. Если бы не я, то это

сделал бы кто-нибудь другой. Я надеюсь.

Мне нравится социальный аспект. И понимание того, что он каким-то непонятным мне образом пойдет в итоге на пользу. Мне кажется, мы извлекли много хорошего, и попытки оптимизировать могут все только разрушить.

■ *Андрей Шитов*

7 Perl Quiz

Perl Quiz — уже ставшая традиционной на многих Perl-конференциях викторина на «знание» Perl. Почему в кавычках? Это вы поймете из самих вопросов. Ответы на викторину в текущем выпуске будут опубликованы в следующем. Итак, поехали!

Ответы из предыдущего выпуска: 1) 1, 2 или 4 (точный ответ будет известен 12 августа), 2) 4, 3) 3, 4) 4, 5) 1, 6) 2, 7) 1, 8) 4, 9) 4, 10) 3.

1. Как называется система публикации модулей на CPAN?

    1. START
    2. STOP
    3. PAUSE

    4. EJECT

2. Сколько детей у Ларри Уолла?

    1. 1
    2. 2
    3. 3
    4. 4

3. Какая аббревиатура не означает архив программ?

    1. CPAN
    2. CRAN
    3. CTAN
    4. SWAN

4. Что в перле общепризнанно считается лучше, чем это сделано в других языках?

    1. Простота синтаксиса

2. Понятность кода
3. Поддержка юникода
4. Поддержка работы с сетевыми ресурсами

5. Насколько реализация Rakudo Perl 6 на JVM быстрее реализации на Parrot?

    1. В четыре раза
    2. В сорок раз
    3. Такая же по скорости
    4. Вообще медленнее, чем было

6. Какой вопрос о перле на Stackoverflow стоит на первом месте по числу просмотров?

    1. Что означает `1;` в конце модуля?
    2. Почему при обращении к элементу массива `@a` надо писать доллар: `$a[1]`?

3. Почему в современном перле поддержка UTF-8 не включена по умолчанию?
4. Когда выйдет Perl 6?

7. Когда символ `\N{BELL}` стал соответствовать коду U+1F514 вместо прежнего U+0007?

    1. Начиная с Perl 5.8
    2. Начиная с Perl 5.10
    3. Начиная с Perl 5.18
    4. Изменение запланировано на Perl 5.20

8. Какую конференцию YAPC::Europe не посетил Abigail?

    1. 2010 в Пизе
    2. 2011 в Риге
    3. 2012 во Франкфурте
    4. 2013 в Киеве

9. Какой секретный оператор придумал Abigail?

    1. <=><=><=>
    2. +=!
    3. }{
    4. =()=

10. Как называется традиционный доклад Матта С. Траута?

    1. The State of the Onion
    2. The State of America
    3. The State of the Velociraptor
    4. The State of the State

■ *Андрей Шитов*