Pragmatic Perl

pragmaticperl.com

Выпуск 1. Февраль 2013

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

Авторы статей: Владимир Леттиев (crux), Андрей Шитов (ash)

Выпускающий редактор: Вячеслав Тихановский (vti)

Ревизия: 2013-03-03 13:17

Оглавление

1. Возрождение Perl

Язык Perl последние несколько лет переживает свое возрождение. И когда, как не в первом выпуске нашего журнала, рассказать об этом?

Среди широких масс любителей постучать по клавиатурам язык Perl ассоциируется с чем-то прошедшим и не стоящим внимания современных программистов. Действительно, после начала разработки шестой версии языка, предыдущей, т.е. пятой, не было уделено достаточно внимания. Однако, Perl 6 — это совершенно другой язык, который развивается параллельно, и не собирается (впрочем, пока и не может) заменить Perl 5. В этой статье везде, где используется Perl, подразумевается пятая версия, если не указано иное.

Выпуски интерпретатора

Возрождение Perl началось с выхода perl 5.10 18 декабря 2007 г., через двадцать лет после выхода первой версии интерпретатора. Предыдущая стабильная версия 5.8 была выпущена пятью годами ранее, в 2002 г. Такой промежуток между версиями никого не устраивал. При разработке 5.11 было решено перейти на месячные минорные и ежегодные стабильные релизы. На сегодняшний день четные версии выходят в апреле каждого года, в то время как нечетные — в 20-х числах каждого месяца. Текущими (на февраль 2013 г.) стабильной и минорной версиями являются 5.16.2 и 5.17.9 соответственно.

Сообщество

Несмотря на перерыв в выпуске версий интерпретатора активность Perl-сообщества нисколько не спадала, а наоборот — только росла.

Количество авторов на CPAN с момента его открытия в 1995 г. с каждым годом все увеличивалось. На сегодняшний день это более 117 000 модулей в 26 000 дистрибутивах написанных и поддерживае-

мых более 10 400 авторами по всему миру. А это более 20 млн. строк кода.

Хакатоны, воркшопы, конференции все также собирают людей в разных городах и странах. Только за последний год было проведено 2 хакатона, 11 воркшопов и 5 крупных YAPC-конференций.

По всему миру зарегистрировано 257 локальных Perl-групп.

Код

Perl-библиотеки росли не только количественно, но и качественно.

Устаревшие и неудобные модули для запуска веб-приложений были заменены универсальным стандартизированным интерфейсом `PSGI` с его реализацией `Plack`. Разработав веб-приложение, совместимое с `PSGI`, можно запустить его практически на каждом популярном веб-сервере, включая асинхронные и многопоточные.

Установка Perl-модулей сделалась простой и удобной благодаря `cpanminus`. Теперь установить модуль можно и без прав администратора, и без утомительной настройки конфигурации.

Если для нескольких приложений требуется свое окружение модулей специфических версий, а также быстрый способ разрешить и скопировать зависимости на удаленную машину где нет доступа в интернет, то можно воспользоваться `carton`.

Если есть множество машин, для которых нужен отдельный репозиторий Perl-модулей и хочется удобно этим управлять и замораживать версии, а также использовать свои внутренние модули, то можно воспользоваться `pinto`.

Если необходимо иметь несколько версий Perl, если нужна слишком новая или слишком старая версия, которой нет в дистрибутиве, то можно воспользоваться `perlbrew`.

Если хочется быть на волне популярности асинхронных приложе-

ний и иметь универсальный интерфейс под различные мультиплексоры, то можно воспользоваться `AnyEvent`.

Если не удовлетворяет встроенная реализация ООП и хочется автоматических конструкторов, проверки параметров, расширяемых типов, ролей, можно воспользоваться `Moose` и `Method::Signatures`. Если это слишком тяжело, то можно воспользоваться `Moo`.

Юникод

В современном многоязычном мире поддержка Юникода является важной характеристикой языка программирования. Perl является абсолютным лидером в этой категории. Perl 5.16 почти полностью поддерживает Юникод 6.1.

Perl-ресурсы

В 2013 г. веб-сайты выполненные в стиле 90-х перестали вызывать даже улыбку. Несмотря на то, что один из самых популярных вебсайтов посвященных Perl perlmonks.org все еще вызывает смешанные чувства, большинство Perl-ресурсов были переработаны и выглядят вполне современно. Это perl.org, perldoc.perl.org и другие.

В силу того, что в интернете ничего не проходит бесследно, практически каждый запрос на урок посвященный Perl выдаст вам давно устаревшую информацию. В связи с этим поддерживается вебресурс perl-tutorial.org, где можно найти самые свежие и рекомендуемые уроки для изучения современного Perl. А можно попробовать пройти уроки и не устанавливая Perl perltuts.com.

Поиск по CPAN также не остался без внимания. С помощью metacpan.org можно не только удобнее искать модули, но, используя API, создавать интересные ресурсы по типу github-meets-cpan.com или perlresume.org.

Книги

За последнее время было издано несколько современных Perl-книг, в том числе *Beginning Perl*, *Intermediate Perl* (2-е издание), *Programming Perl* (4-е издание), *Modern Perl*, *Learning Perl* (6-е издание), *Beginning Perl* (3-е издание). На сегодняшний день ведется работа по переводу книги *Modern Perl* на русский язык.

■ *vti*

2. YAPC::Europe 2013 «Future Perl»

С 12 по 14 августа этого года в Киеве пройдет очередная европейская Perl-конференция YAPC::Europe.

История

Аббревиатура YAPC расшифровывается как Yet Another Perl Conference (еще одна Perl-конференция). «Еще одна» из-за того, что она появилась в 1999 году как альтернатива существовавшей с 1997 года ежегодной конференции The Perl Conference (которая теперь превратилась в OSCON — O'Reilly Open Source Convention). Считают, что одной из причин сделать еще одну конференцию было желание радикально снизить стоимость входного билета (с 1000 до 100 долларов).

Первая YAPC прошла в американском Питтсбурге летом 1999 года. В 2000-м в Европе названием YAPC::Europe появилась своя конференция, а американская стала называться YAPC::NA (North America). YAPC::Europe каждый год перемещается по разным европейским городам, а американская, соответственно, по городам США, лишь однажды (в 2003) ее провели в Канаде.

Помимо этих двух исторических конференций появились еще несколько ответвлений. В 2003–2005 годах проводилась YAPC::Israel, а потом она стала open source-мероприятием OSDC::Israel. Появились две конференции в Южной Америке: YAPC::SA (South America) и YAPC::Brazil (YAPC::SA, если я ничего не напутал, пытаясь разобраться в истории, проводилась до 2009 года в рамках Free Software Forum в Бразилии же). В прошлом году YAPC::Brazil прошла совместно с конференцией W3C.

Отдельно нужно сказать о YAPC::Asia, которая фактически является YAPC::Токио: она появилась в 2005 году как YAPC::Taipei, а с 2006 неизменно проходит в Токио. Особенность этой конференции в том, что она собирает по 600–700 человек, в отличие от 300–400 в Америке и Европе.

В 2008 появилась и YAPC::Russia. Ее предшественником был первый российский Perl-воркшоп «Perl Today», проведенный в Москве в октябре 2007 года. Начиная с 2008, YAPC::Russia проводится через год то в Москве, то в Киеве. Почему в Киеве? А потому что те же организаторы провели в 2008 году в Киеве первый украинский Perl-воркшоп «Perl Mova» («Язык перл»), а потом решили, что полезнее делать одну крупную конференцию вместо двух поменьше, и стали перемещаться между этими столицами. В 2013 YAPC::Russia не будет, потому что мы выиграли конкурс на проведение в Киеве европейской конференции YAPC::Europe и решили потратить все силы именно на нее.

Сайты конференций YAPC

- YAPC::NA yapcna.org;
- YAPC::Europe yapc.eu;
- YAPC::Asia yapc.asia;
- YAPC::Brazil yapcbrasil.org.br;
- YAPC::Russia yapcrussia.org.

YAPC::Europe 2013 в Киеве

Конференция YAPC::Europe 2013 пройдет в Киеве с 12 по 14 августа. Место проведения выбирает специальный комитет — YAPC::Europe Venue Committee, который рассматривает заявки от представителей местных Perl-сообществ и выбирает лучшее из них. Несмотря на то, что место выбирает головной комитет, вся фактическая работа по организации конференции, наполнение ее смыслом, образовательной и развлекательной программами для участников ложится на плечи местных организаторов.

Киев станет самой восточной европейской столицей из всех, где когда-либо проводились YAPC::Europe (предыдущий рекорд был установлен в 2011 году, когда конференция прошла в Риге).

Мы долго выбирали помещение для конференции, пытаясь найти баланс между ценой, качеством и географией, и в итоге остановились на лучшем варианте, который только можно найти в центре Киева. YAPC::Europe 2013 пройдет в Украинском Доме по ул. Крещатик, 2. Здание расположено в пяти минутах ходьбы от Майдана; рукой подать до Днепра и смотровой площадки, не говоря уже о многочисленных кафе и гостиницах, которые в центре найти очень просто.

Сайт конференции: yapc.eu/2013

Твиттер: @yapcrussia

Страница на Фейсбуке: facebook.com/yapceu

Электронная почта: mail@yapcrussia.org

Еженедельные новости: act.yapc.eu/ye2013/news/

Тема

Тема нашей конференции — Future Perl (Будущий перл). Мы хотим посвятить ее в первую очередь размышлениям о том, каким станет Perl в ближайшие годы, и что будет дальше.

Конференция продлится три дня, и на первый же из них запланированы выступление Ларри Уолла, обзор Дейва Кросса о 25-летней истории языка, доклад Ричарда Йелиника о том, как увеличить популярность перла и круглый стол о будущих версиях Perl.

Инициатива подискутировать о версиях родилась после бурного февральского обсуждения в интернете про Perl 7. Кстати, идею круглого стола и желание стать конференцией, на которой решается будущее языка, переняли и организаторы YAPC::NA 2013, которая пройдет двумя месяцами раньше YAPC::Europe.

Участие в конференции

Чтобы принять участие в конференции, необходимо зарегистрироваться на сайте yapc.eu/2013[1]. Если вы посещали одну из предыдущих YAPC::Europe или YAPC::Russia, то на сайт удастся зайти со своей парой имя–пароль: act.yapc.eu/ye2013/main. Если пароля нет, надо зарегистрироваться по адресу act.yapc.eu/ye2013/register и подтвердить адрес электронной почты, нажав на ссылку в автоматически отправленном письме.

Затем нужно обязательно нажать кнопку Join YAPC::Europe 2013.

На 1 марта на сайте конференции зарегистрировалось 125 человек из 18 стран. Почти половина из этих участников приходится на Украину и Россию. Присоединяйтесь, будет интересно.

Программа

Конференция включает в себя три дня (понедельник–среда 12–14 августа), в течение которых будут проходить доклады. Мы предполагаем организовать три параллельных потока выступлений. Иногда на YAPC::Europe интересных докладов оказывается так много, что открывается и четвертый поток (а в Риге в 2011 в первый день было пять одновременных потоков).

Каждый день мы начинаем в 10 утра выступлением (keynote) одного из известных в мире Perl деятелей. На каждый вечер, с 17 до 18, запланированы часовые сессии пятиминутных блиц-докладов. Все остальное время заполнят докладчики рассказами на самые разные темы, например, о внутренностях перла, о применении перла для создания сайтов или для системного администрирования.

[1]Сайт конференции, как и сайты десятков других перловых мероприятий, работает на Act — A Conference Toolkit, написанный французскими программистами много лет назад. Act очень хорош тем, что можно относительно быстро развернуть и сайт, и сбор докладов, и прием оплаты, поэтому можно закрыть глаза на некоторые интерфейсные неудобства типа необходимости нажимать на кнопку «Join», когда вы уже залогинились. В 2010 году бешеные японцы™ с нуля создали свой код, реализовав функции, доступные в Act (разумеется, с использованием Plack), и пользуются им теперь для сайтов YAPC::Asia.

Довольно часто кто-нибудь делает доклады о том, как работать с текстом (где перл не имеет конкурентов). Иногда рассказывают о работе с прикладными задачами или, например, для написания музыки. Особое место занимает работа с базами данных — как с традиционными SQL, так и NoSQL. CPAN, git, PSGI/Plack, Perl 6 — все это и многое другое всегда появляется в программе выступлений.

Подать свою заявку на выступление можно на сайте конференции по адресу act.yapc.eu/ye2013/newtalk.

Расширенная программа

Конференция не ограничивается только докладами. YAPC::Europe — это в большой степени и социальное мероприятие, место встречи старых знакомых и возможность пообщаться с незнакомыми людьми, которые тоже интересуются перлом. Для того, чтобы общение проходило более продуктивно, мы организуем дополнительные мероприятия внутри и вне помещения. На сегодня наш план включает следующее.

11 августа (воскресенье)

10:00–17:00. Хакатон про Perl 6. Здесь соберутся люди, ответственные за дизайн и реализацию Perl 6, в том числе Ларри Уолл, Патрик Мишо и Джонатан Вортингтон. Присоединиться к хакатону может каждый желающий независимо от его знакомства со внутренним устройством компиляторов. Это и реальная возможность узнать из первых уст о причинах столь долгого этапа разработки. Мероприятие пройдет (скорее всего) в конференц-зале гостиницы «Днипро», расположенной напротив Украинского Дома.

12:00–15:00. Тренинг для докладчиков. Это мероприятие рассчитано на всех докладчиков, которые хотят улучшить свои презентации (даже в последний день перед конференцией это часто еще не позд-

но) или узнать мнение коллег о том, как лучше подать материал. Место тренинга: гостиница «Днипро».

18:00–24:00. Предварительная встреча участников на открытом воздухе или в кафе. Это традиционная встреча для тех, кто приехал на конференцию заранее и не хочет терять время, оставаясь в гостинице.

12 августа (понедельник)

18:00–19:30. Вечеринка после докладов, которую планирует организовать один из наших спонсоров. Подробности появятся позже.

13 августа (вторник)

19:00–23:00. Круиз на теплоходе по Днепру. Организаторы конференций YAPC::Europe традиционно устраивают в один из вечеров большой ужин для всех участников (стоимость ужина входит в стоимость входного билета на конференцию). Например, в 2009 году в Лиссабоне был устроен невероятный ужин с бесконечным потоком национальных мясных блюд, которые подавали до тех пор, пока хватало сил все это есть. Там же Дэмиан Конвей провел командную викторину на знание хитрых особенностей перла. А я там потерял телефон. Другой пример очень хорошего обеда был в 2011 году в Риге. Мы привезли участников в известный рижский ресторан Lido. В общем, там тоже был праздник желудка. В этом году мы решили воспользоваться уникальной атмосферой Киева и вместо выездного ужина организовать катание на теплоходе (разумеется, с фуршетом на борту). В нашем распоряжении будет четыре часа на самом большом пассажирском теплоходе Киева.

14 августа (среда)

А вот на вечер среды пока ничего не запланировано. Некоторые участники к этому времени уже поедут домой, а те, кто остаются,

часто собираются на продолжение где-нибудь в кафе. Если у вас есть идеи — напишите нам.

Стоимость участия

YAPC::Europe всегда старается сделать минимальным стоимость участия. В этом году цена билета на все три дня (включая все доклады и все социальные мероприятия) — 110 евро.

Обратите внимание: до 1 апреля действует специальное предложение «Early bird price», и билет можно купить на 18% дешевле, за 90 евро.

Для студентов действует студенческий тариф 60 евро, но есть вероятность того, что мы сможем существенно снизить стоимость студенческих билетов. Следите за новостями на сайте конференции.

Бесплатное участие

Существует как минимум четыре способа получить бесплатный билет.

Докладчики

Все докладчики, которые предложили доклад продолжительностью более 20 минут (и который был одобрен программным комитетом и включен в расписание), участвуют в конференции бесплатно.

Send-a-Newbie

Так называется инициатива по сбору средств для молодых программистов, которые никогда не были ни на одной конференции

YAPC, и которые не могут позволить себе поездку на YAPC::Europe. В зависимости от суммы собранных денег двое-трое участника получат и бесплатный билет на конференцию, и деньги на проезд и на гостиницу. Подробности на сайте send-a-newbie.enlightenedperl.org.

Волонтер партнерской программы

Еще один способ получить бесплатный билет — стать волонтером партнерской программы. Партнерская программа — это серия экскурсий по Киеву в главные дни конференции, которая организована для партнеров участников, приехавших с ними на конференцию, но лишь за компанию, а не чтобы слушать доклады. Если ваш партнер готов помочь с организацией партнерской программы, он получает для вас бесплатный билет на конференцию.

Как добраться и где жить

Для того, чтобы найти подходящую по цене и удаленности от места проведения конференции гостиницу, мы запустили на сайте поиск, где вы можете сориентироваться в ценах и перейти на сайт партнера для бронирования. Цены на гостиницы, которые вы здесь найдете, ниже цен в других местах. Более того, зарегистрированные участники могут получить скидку на некоторые гостиницы Киева.

В ближайшее время будет открыт поиск авиабилетов в Киев и обратно. Скидок там не будет, но цены и здесь должны быть самыми выгодными.

Пользуясь этими услугами на сайте конференции, вы заодно вносите свой вклад в бюджет конференции, не платя при этом ничего сверх тарифа гостиницы или авиакомпании.

Спонсоры

Конференция — это праздник для всех участников: и для докладчиков, и для слушателей. Для организаторов это праздник вдвойне, поскольку нам требуется прилагать усилия на то, чтобы качественно реализовать все задуманное (или по крайней мере свести концы с концами :-). Мы уже выбрали лучшее помещение, нашли самый большой теплоход, а сейчас занимаемся тем, чтобы обеспечить трехразовое питание и дополнительные мероприятия, упомянутые выше.

Если вы или компания, в которой вы работаете, готова поддержать конференцию, а в конечном итоге и будущее перла, — становитесь нашим спонсором, наш контактный адрес mail@yapcrussia.org. Условия сотрудничества очень разнообразные и гибкие.

Организаторы

Конференцию организовывают участники киевского и московского Perl-сообществ. Основной объем текущей работы по подготовке мероприятия выполняем мы с Вячеславом Тихановским. Ярослав Коршак и Анатолий Шарифулин работают над промоутингом конференции и поиском дополнительных спонсоров. Для работы над расписанием докладов к нам присоединились Алексей Капранов, Руслан Закиров и Дмитрий Карасик.

■ *Андрей Шитов*

3. Moo — современный минимальный ООП-фреймворк

Объектно-ориентированное программирование в Perl 5 представлено очень базовыми вещами. Однако пробелы в реализации ООП были заполнены Perl сообществом в виде модулей на CPAN. В данной статье будет рассказано об истории появления этих реализаций и, в том числе, о Moo, как современной рекомендуемой практике.

ООПокалипсис или немного истории ООП в Perl

Никто не сможет сейчас точно сосчитать сколько тучных троллей было откормлено на флеймах о реализациях ООП в Perl. Чтобы разобраться, почему существует такое большое количество вариантов реализации ООП для Perl на CPAN и почему постоянно на эту тему ломаются капчи, надо вспомнить историю развития Perl. Вы можете смело пропустить последующие параграфы, если слышали эти жалкие оправдания сотни раз.

Вкратце, суть такова, что изначально Perl не был явно ориентирован на какую-либо определённую парадигму программирования. Выпуск Perl 5 предоставил программистам возможность использовать ОО-подход в разработке, создавать модули и компоновать классы. Позже однако выяснилось, что появившийся функционал был достаточно беден и заставлял людей больше думать над способом реализации, чем над самой логикой классов и объектов. Значительный редизайн ООП был запланирован для Perl 6, но эту реализацию увидят вживую разве что только наши внуки, а людям было нужно уже сейчас и в Perl 5.

Так и началось великое народное освободительное движение по расширению пространства имён `Class::*` и иже на CPAN. Безусловным лидером в этом восстании классов стал `Moose`, показав исключительно продуманную и богатую возможностями систему и став де-факто стандартом реализации ООП в Perl. Но как и у любой медали обратная сторона использования `Moose` — развеси-

стые зависимости и тяжёлый взлёт приложений, написанных с его использованием. Это, в свою очередь, породило вторую волну минималистичных реализаций `Moose`. Так появился `Mouse`, который частично решал вопрос со скоростью взлёта за счёт `XS`-кода и несколько облегчённого функционала, и практически не имел зависимостей от других модулей.

Тяжелее всего пришлось разработчикам модулей, которым предстояло выбрать в этом море именно тот фреймворк, который можно было комфортно использовать и не навлечь на себя гнев пользователей за внезапно раздувшиеся зависимости и долгий запуск. Тут на сцену истории на белом коне въезжает `Any::Moose`, который обещает пользователям возможность самим решать какую `Moose`-реализацию использовать. Желание пользователя угадывалось по специальной переменной окружения или в зависимости от того был ли уже загружен `Mouse` или `Moose`.

Moo

В конце 2010 г. появляется новый проект минималистичной реализации объектно-ориентированного программирования для Perl с лаконичным названием Moo. Автором модуля является Matt S. Trout (mst).

Утверждается, что `Moo` — это аббревиатура от *Minimalist Object Orientation* (Минималистичная Объектная Ориентированность), хотя все прекрасно поняли, что это урезанное название ООП-фреймворка `Moose`. И вскоре, под тяжестью фактов, автор `Moo` сам сознаётся, что да, модуль является легковесным подмножеством `Moose`, и поскольку реализовали не все возможности, а только на две трети, то и количество букв в названии соответственно уменьшили.

Цель проекта — дать возможность разработчику начать использовать расширенный синтаксис и возможности ООП как в `Moose`, но при этом не иметь накладных расходов ни на время запуска, ни на объём зависимостей, а также для просты развёртывания не иметь `XS`-зависимостей. Более того, если программа загружает `Moose`,

то Moo автоматически регистрирует метаклассы для ваших Moo-модулей, позволяя работать вашем коду совершенно прозрачно, как если бы он был написан с использованием Moose.

Это означает, что для Moo не требуется никакого Any::Moose — он самостоятельно сможет переключиться на Moose, в случае его загрузки. Самое интересное, что сделает это он даже лучше чем Any::Moose, поскольку для Moo совершенно не важно в каком порядке загружаются модули вашего проекта, в то время как Any::Moose при определённом порядке загрузки может сделать неверный выбор, загрузив сначала Mouse и в последствии уже не может переключиться на использование Moose, что может привести к ошибкам. Самое интересное, что в больших проектах такое может легко произойти, причём воспроизвестись в тот момент, когда вы выкатываете код в боевое окружение.

Это замечательное обстоятельство имело прямое отношение к тому, что последняя версия Any::Moose 0.20, которая вышла в последний день 2012 г. (как символично), объявляет модуль устаревшим и рекомендует всем переходить на Moo.

> Good night, sweet prince…

Стоит отметить один курьёзный момент. Идея с укорачиванием имени Moose пришлась по вкусу Ingy döt Net, который выпустил yet another микро ООП-фреймворк с названием Mo. На что Matt S. Trout ответил созданием модуля M, в описании которого сообщил:

> Moose привёл к созданию Mouse, приведшего к созданию Moo, приведшего к созданию Mo. Но модуль Mo Ingy так мало чего умеет, и меньше делать можно только уже абсолютно ничего не делая. Именно это и делает M — ничего.

Ingy позже выпустил модуль Moos, чтобы закрыть весь ряд букв. Кроме того, ранее у него ещё был создан модуль Mousse. Такой вот занятный CPAN-киберсквоттер.

Быстрый старт с Moo

Итак, вам уже не терпится переписать код с использованием модного и прогрессивного фреймворка. Начнём с азов. Например, попробуем создать модуль для ведения логов. Не обращайте внимание на существование `Log::Dispatch` или `Log4perl`, наш модуль будет базироваться на самом совершенном ООП-фреймворке, поэтому *a priori* заткнёт их за пояс.

```
1 package BestLogEver;
2 use Moo;
3 1;
```

Поздравляю!

Вот в общем-то и всё. На данном этапе ваш класс уже получил несколько замечательных вещей:

- Метод `new()` для создания нового объекта
- Включённая по-умолчанию прагма `strict`
- Включенная по-умолчанию прагма `warnings`, в режиме трактующем все предупреждения как ошибки.

Напишем пример приложения, использующего этот класс:

```
1 use strict;
2 use warnings;
3 use BestLogEver;
4
5 my $log = BestLogEver->new();
```

У нас получилось создать объект нашего нового класса.

Теперь необходимо продумать, что будет делать наш класс. Очевидно записывать сообщения в лог. Таким образом, нашему классу потребуется метод `log`. Но куда записывать? На экран, в файл, в базу данных...? Вариантов может быть много и в будущем могут появляться новые, поэтому логично выделить роль *Хранилища* сообщений для всех видов.

```
1 package BestLogEver::Role::Storage;
2 use strict;
3 use warnings;
4 use Moo::Role;
5
6 requires 'log';
7
8 1;
```

Теперь все создаваемые в последующем хранилища будут использовать эту роль. Обратите внимание на модуль `Moo::Role`, который превращает наш класс в роль. Ключевое слово `requires` требует, чтобы класс, реализующий данную роль обязательно имел указанный метод `log`. Попробуем теперь переписать наш модуль так, чтобы он при создании воспринимал атрибут `storage`, который бы задавал какой тип хранилища мы хотим для ведения нашего лога и загружал нужный модуль.

```
1 package BestLogEver;
2 use Moo;
3
4 has 'storage' => (
5    is => 'ro',
6    isa => sub { die 'bad storage name' unless $_[0] && !
       ref $_[0] }
7 );
8
9 1;
```

С помощью ключевого слова `has` мы объявляем атрибут `storage`, для которого `Moo` автоматически создаст `getter` (метод для получения значения атрибута), но не даст возможности изменения этого значения, т.к. мы объявили его как `ro` (read only — только для чтения). Атрибут `isa` задаёт функцию проверки значения нашего атрибута `storage`. В данном случае мы проверяем, что это не пустой скаляр. В `Moo` нет встроенной системы типов как в `Moose`, хотя это легко решается подключением `MooX::Types::MooseLike`. Теперь в нашем скрипте мы можем инициировать объект с атрибутом `storage`:

```
1 my $log = BestLogEver->new( storage => 'Screen' );
2 say $log->storage;
```

Таким образом, благодаря `Moo` мы уже имеем возможность проверять значение атрибута на допустимость и получить `getter` (и при желании `setter`) методы без лишних телодвижений. Попробуем написать наш первый плагин `BestLogEver::Storage::Screen`, выводящий сообщения на экран:

```
1 package BestLogEver::Storage::Screen;
2 use Moo;
3 with 'BestLogEver::Role::Storage';
4
5 sub log {
6     my ($self, $message) = @_;
7     print $message;
8 }
9
10 1;
```

Мы подключаем роль с помощью ключевого слова `with`. Тем самым, автоматически накладывая на модуль все требования, описанные в роли `BestLogEver::Role::Storage`, и в данном случае обязательное требование наличия метода `log`. Если наш плагин не будет иметь этого метода, то при попытке его загрузить получим ошибку:

```
Can't apply BestLogEver::Role::Storage to BestLogEver::
Storage::Screen — missing log at /usr/share/perl5/Role/
Tiny.pm line 250.
```

Прекрасно, теперь осталось обучить наш главный класс подгружать нужный плагин и реализовать метод `log` через него.

```
1 package BestLogEver;
2 use Moo;
3 use Module::Runtime qw(require_module);
4
5 has 'storage' => (
6     is => 'ro',
7     isa => sub { die 'bad storage name' unless $_[0] && !
          ref $_[0] }
8 );
9
10 has 'storage_object' => (
11     is => 'lazy'
12 );
13
14 sub _build_storage_object {
15     my $self = shift;
```

```
16      my $class = 'BestLogEver::Storage::' . $self->storage
            ;
17      require_module($class);
18      return $class->new;
19  }
20
21  sub log {
22      my ($self, $message) = @_;
23      $self->storage_object->log($message);
24  }
25
26  1;
```

Поясняю по порядку.

У нас появился атрибут `storage_object`. Именно в нём мы будем хранить объект загруженного плагина. Атрибут объявлен как `lazy`, что означает две вещи: во-первых, атрибут будет создан при первом обращении, а, во-вторых, для создания атрибута будет использована функция с префиксом `_build_` + имя нашего атрибута, т.е. `_build_storage_object` .

Далее мы реализуем функцию `_build_storage_object`, которая по значению атрибута `storage` загружает соответствующий класс и возвращает созданный объект.

Теперь реализовать метод `log` довольно просто. Вызываем метод `storage_object`, который возвращает нам нужный объект плагина и вызываем его метод `log`.

Теперь наша программа выведет текст на экран:

```
1 my $log = BestLogEver->new( storage => 'Screen' );
2 $log->log("hello, world!");
```

```
hello, world!
```

Важно отметить, что создание атрибута `storage_object` происходит только один раз, во время самого первого вызова метода `log`.

И для ещё одной демонстрации возможностей Moo, покажем как можно модифицировать методы.

Например, нам не нравится, что при выводе сообщений не добавляется перенос строки, но и лишний перенос строки ставить не нужно, если он уже есть. Чтобы исправить это мы можем изменить поступающее сообщение в метод log у всех плагинов через класс роли Storage:

```
1 package BestLogEver::Role::Storage;
2 use strict;
3 use warnings;
4 use Moo::Role;
5
6 requires 'log';
7
8 before log => sub {
9     chomp $_[1];
10    $_[1] .= "\n";
11 };
12
13 1;
```

Ключевое слово before задаёт функцию, которая вызывается перед запуском log и может модифицировать передаваемые параметры в переменной @_, которая затем поступит в log. Код возврата из этой функции не важен.

Таким образом, мы можем «пропатчить» одним выстрелом все классы, которые выполняют указанную роль. Также вам доступна подмена самой функции (around) и изменение возвращаемых результатов (after).

Кто использует Moo

Moo определённо заслужил признание как современный ООП фреймворк в Perl. Всё больше модулей на CPAN начинают его использовать.

В качестве примеров можно привести Dancer 2 — новая версия популярнейшего фреймворка построена изначально на Moo. Извест-

нейший ORM DBIx-Class использует `Moo`. Email::Sender 1.300000 вышедший в новом 2013 г. переключился с использования `Moose` на `Moo`.

Подумайте, возможны вы следующий, кто оценит модуль по достоинству и начнёт его использование.

■ *crux*

4. Dancer2 — Революция

Ничего не предвещало беды, но 18 февраля 2013 г. Alexis Sukrieh сообщает в своём блоге и в рассылке `dancer-users@` о том, что состоится выпуск `Dancer2` как самостоятельного проекта и он не будет замещать существующий `Dancer`. 22 февраля на CPAN появляется дистрибутив `Dancer2` с версией 0.01.

`Dancer2` изначально создавался с нуля на основе фреймворка `Moo` для получения консистентной ООП-системы удобной для расширения и поддержки. Основная решаемая в новой версии проблема — избавление от глобальных переменных и изолирования приложений от любых возможных коллизий. При этом также планировалось обеспечить 100% совместимость с `Dancer`, чтобы была возможна прозрачная миграция старых приложений и плагинов на новую версию фреймворка.

Фактически с осени прошлого года было заморожено развитие `Dancer` и все силы были брошены на доработку `Dancer2`. Но затем стало ясно, что прозрачная миграция будет невозможна прежде всего по техническим причинам. Движки (`engines`) шаблонизаторов, сессий, логов существенно отличаются в `Dancer` и `Dancer2`, так как в `Dancer` — это обычные классы Perl 5 , которые надо расширять, а в `Dancer2` — это роли (`Moo::Role`), которые подключаются. Эта техническая проблема была очевидна с самого начала. На примере адаптации плагина `Dancer::Template::Xslate` под `Dancer2` мне было видно, что отличия настолько кардинальные, что удерживать систему плагинов в состоянии совместимости под разные версии `Dancer` будет сложнее, чем просто выпустить отдельные плагины под разные версии фреймворка. Более того, настройки движков в `config.yml` также имеют отличия, что полностью ломает обратную совместимость. Даже при условии, что в `Dancer2` будет написана некая обёртка для поддержки обратной совместимости, многие реальные приложения `Dancer` разбиты по множеству пакетов и область видимости приложений может быть нарушена (нет больше глобальных переменных).

Всё это привело к осознанию того, что эволюционного перехода `Dancer` в `Dancer2` не получится и потребуется сделать волевое ре-

шение по разделению экосистем `Dancer` и `Dancer2`:

`Dancer2` выпускается в своём собственном пространстве имён на CPAN и даёт старт собственной экосистеме

- разработка `Dancer` замораживается и направленна на сопровождение существующего набора возможностей;
- пользователи `Dancer` счастливы;
- пользователи `Dancer2` счастливы;
- миграция приложений теперь отдаётся под контроль разработчика, который будет решать надо ему делать переход или оставаться на старой системе;
- будет выбран новый сопровождающий проекта `Dancer` (на эту должность было выбрано сразу двое разработчиков: Yanick Champoux и Alberto Simões).

Сейчас на CPAN уже появились первые выпуски плагинов для `Dancer2` и можно констатировать, что революция `Dancer2` свершилась. Но причин для паники нет, разработчики говорят о том, что для существующих приложений можно продолжать использовать `Dancer`, поскольку его сопровождение не прекращается. А вот приступая к созданию новых приложений, предпочтительно выбирать `Dancer2`.

■ *crux*

5. Padre IDE. В шаге от релиза 1.0

Padre — это графическая среда разработки для языка Perl, написанная на языке Perl. Название является аббревиатурой от Perl Application Development and Refactoring Environment (Среда Разработки и Рефакторинга Перл Приложений).

Проект основал в мае-июне 2008 г. разработчик и Perl-тренер — *Габор Сабо* (Gabor Szabo). Основной причиной, побудившей его к созданию IDE — это фактическое отсутствие свободной комфортной среды разработки для языка Perl, особенно для начинающих разработчиков. Perl-программисту нужен удобный кросс-платформенный инструмент с подсветкой синтаксиса, автоматической проверкой синтаксиса, контекстными подсказками, авто-дополнением, встроенными утилитами для рефакторинга и графическим отладчиком — все те возможности характерные для нормальной IDE. Универсальный текстовый редактор обычно хорошо справляется с подсветкой синтаксиса, но на большее уже бывает не способен.

Габор активно рекламировал свой новый проект в своём блоге, рассказывал о нём на различных Perl-конференциях и привлёк внимание разработчиков, заинтересовав их в участии над проектом. Первый публичный релиз проекта под именем `Padre` на CPAN состоялся уже 26 июля 2008 г. В качестве графического тулкита, по совету Adam Kennedy, был выбран кросс-платформенный `WxWidgets` , что позволяло приложению выглядеть *нативным* на платформе Windows.

К октябрю 2008 г. в проекте уже было 5 разработчиков.

В ноябре 2008 г. было получено одобрение на грант от фонда perlfoundation.org на проект интеграции `Padre` с `Parrot` и `Rakudo` для поддержки подсветки синтаксиса Perl 6 и других языков программирования, поддерживаемых в `Parrot`.

Проект активно выпускал новые версии, появились переводы интерфейса на множество языков, в том числе и русский, неуклонно расширялся список плагинов для `Padre`. Возможности `Padre` стали

практически полностью покрывать те ожидания, на которые рассчитывали пользователи IDE. К июлю 2009 г. количество разработчиков насчитывало уже 40 человек. Были сделаны готовые сборки для Windows, Mac OS, а также готовые пакеты для большинства популярных Linux дистрибутивов и в портах BSD систем. `Padre` появился в социальных сетях ohloh, linkedin, twitter...

С версии 0.70 стабильные релизы стали нумероваться чётными числами. После выпуска версия для разработки получала нечётный номер, но никогда не публиковалась на CPAN. За несколько дней до выпуска, формировалась предвыпускная ветка в которую созывались переводчики для работы над переводом `Padre`. Благодаря этому нововведению, новый выпуск получал полностью локализованный интерфейс.

Последний релиз `Padre` версии 0.96 состоялся 23 апреля 2012 г. С тех пор не было ни одного выпуска. Связано это прежде всего с тем, что один из самых активных контрибуторов проекта Adam Kennedy сменил рабочее место и переехал из Австралии в США. На новом месте у него практически не осталось времени на участие в этом проекте и на поддержку своих CPAN модулей вообще, поскольку он вместе с местом работы сменил и основной язык программирования, им стал C# (новый работодатель работает полностью на стеке языков и продуктов Microsoft).

Несмотря на то, что проект продолжал потихоньку развиваться, со временем инфраструктура проекта стала расшатываться. С первого дня и до сих пор проект был расположен на собственных ресурсах Габора Сабо. Первым перестал работать trac, затем irc-бот Hyppolit, сообщавший о всех svn-комитах на канал разработчиков. Простой ресурсов мог длиться по нескольку недель, пока Габор находил время и возможность разобраться с проблемами. Это усложняло работу разработчиков и иногда приводило к неприятным конфликтам.

В июне 2012 г. у Габора возник конфликт с администраторами irc.perl.org и он попытался перенести канал #padre в сеть freenode.net. К счастью, эту инициативу никто из разработчиков и пользователей не поддержал.

На данный момент проект развивается достаточно медленно и, к

сожалению, теряет внимание пользователей и разработчиков. В начале нового 2013 г. Габор предложил начать ежемесячный выпуск версий `Padre`, и не придавать большого значения грядущему релизу 1.0, т.е. просто выпустить его как есть с тем набором изменений какие будут на тот момент.

На момент написания статьи (февраль 2013 г.) новые версии пока не выпущены, хотя ветка в svn-репозитории для будущего релиза 0.98 уже создана. Хочется скрестить пальцы на удачу, чтобы проект приобрёл второе дыхание и вновь начал активную жизнь. В том числе и благодаря вам, кто прочёл эту статью и задумался о том, как можно помочь проекту.

■ *crux*

6. Всё, что вы хотели знать об AnyEvent, но боялись спросить

AnyEvent на сегодняшний день является самым популярным современным фреймворком событийно-ориентированного программирования (СОП) в Perl. Об этом, в частности, свидетельствует 25-ая позиция в Top-100 рейтинге модулей MetaCPAN.

Нужно отметить, что парадигма СОП является наиболее оптимальным выбором для программирования сетевых приложений в Perl, показывая значительно лучшую масштабируемость, чем использование форков или тредов. Треды в Perl имеют особый статус, но как хорошо заметил Алан Кокс:

> Компьютер — это конечный автомат. Треды для тех людей, которые не умеют программировать конечные автоматы.

Тем не менее, в современных сетевых приложениях замечается тенденция к комбинированию подходов. Например, использовать `prefork` — предварительный запуск нескольких рабочих процессов, каждый из которых имеет свой главный цикл обработки событий.

`AnyEvent` представляет собой слой абстракции над какой-либо из реализаций СОП. В этом отношении он похож на модуль `DBI`, который абстрагируется от различных API баз данных, предоставляя единый интерфейс для всех.

Существует достаточно много модулей реализующих событийное программирование: `EV`, `Event`, `Glib`, `Tk`, `Lib::Event`, `Irssi`, `IO::Async`, `Qt`, `FLTK` и `POE`. Как правило, если в программе начинается использовать одна из реализацией СОП, вы больше не можете начать использовать другую в рамках того же процесса. Это порождает зависимость на выбранный фреймворк и принуждает всех, кто будет использовать ваш модуль также использовать только этот фреймворк. В сложных проектах, составленных из множества компонент это приводит к тому, что для включения какого-либо

модуля его приходиться править под используемую реализацию СОП, а если какой-то функционал выбранного вами фреймворка является уникальным, то возможно придётся переписывать всю логику модуля с нуля.

`AnyEvent` был призван устранить эту проблему путём создания абстрактного интерфейса для программиста для работы с событиями. Таким образом, код перестаёт зависеть от конкретной реализации СОП, позволяя использовать модуль в работе с любой реализацией, выбранной пользователем или доступной в момент запуска. В состав `AnyEvent` включены адаптеры для большинства популярных реализаций: `Cocoa::EventLoop`, `Lib::Event`, `Event`, `EV`, `FLTK`, `Glib`, `IO::Async`, `Irssi`, `POE`, `Qt`, `Tk`. Кроме того, в `AnyEvent` включена простая реализация `AnyEvent::Loop`, подключаемая в том случае, если ничего другого не было найдено. Это бывает удобно, если ваша программа не должна иметь `XS`-зависимостей.

Интерфейс AnyEvent

Рассмотрим интерфейс AnyEvent. Основной сущностью, над которой мы оперируем в программе является *страж* (или *наблюдатель*) — объект, который хранит относящиеся к наблюдаемым событиям данные, как например, функцию-колбэк, файловый дескриптор и т.д. Существует несколько типов стражей.

- *страж ввода/вывода* для файловых дескрипторов:

```
$w = AnyEvent->io (
    fh   => <filehandle_or_fileno>,
    poll => <"r" or "w">,
    cb   => <callback>,
);
```

- *страж времени* (*таймер*) для отслеживания временных интервалов:

```
$w = AnyEvent->timer (
    after    => <fractional_seconds>
    interval => <fractional_seconds>,
    cb => <callback>,
);
```

- *страж сигналов* для отслеживания сигналов, получаемых процессом:

```
$w = AnyEvent->signal (
    signal => <uppercase_signal_name>,
    cb => <callback>,
);
```

- *страж процессов потомков.* Данный тип был выделен, чтобы иметь возможность обрабатывать каждого потомка отдельно. С предыдущим типом это делать затруднительно, так как сигнал `SIGCHLD` не может быть распределён на нескольких обработчиков:

```
$w = AnyEvent->child (
    pid => <process_id>,
    cb => <callback>,
);
```

- *страж простоя* для выполнения задач, когда в очереди нет никаких других событий для обработки:

```
$w = AnyEvent->idle (
    cb => <callback>
);
```

- *переменные состояния* — это уникальный тип, который можно отождествить с *обещаниями*, т.е. будущими значениями, которые мы ожидаем получить. Поскольку в интерфейсе `AnyEvent` нет как такового основного цикла событий (т.е. когда программа переходит в режим блокирующего ожидания), то единственным способ заблокироваться в ожидании — попытаться получить значение переменной состояния:

```
$cv = AnyEvent->condvar;

$cv->send (<list_of_variables>);
my @res = $cv->recv;
```

Одна из интересных особенностей интерфейса `AnyEvent` — это необходимость использования замыканий, поскольку нет другой возможности передать параметры в колбэк-функцию. С одной стороны это смущает новичков, но с другой стороны это имеет свои плюсы, поскольку замыкания работают быстрее и используют

чуточку меньше ресурсов, т.к. не требуют создания локальных переменных для передачи параметров в функцию.

Для понимания рассмотрим небольшой пример. Программа ожидает ввода пользователя с `STDIN`, и если ввода нет в течении 4.2 секунд — прерывает свою работу. Если ввод есть — выводит введённую строку.

```
1 my $done = AnyEvent->condvar;
2
3 my ($w, $t);
4
5 $w = AnyEvent->io (
6     fh => \*STDIN,
7     poll => 'r',
8     cb => sub {
9         chomp (my $input = <STDIN>);
10        warn "read: $input\n";
11        undef $w;
12        undef $t;
13        $done->send();
14    });
15
16 $t = AnyEvent->timer (
17    after => 4.2,
18    cb => sub {
19        if (defined $w) {
20            warn "no input for a 4.2 sec\n";
21            undef $w;
22            undef $t;
23        }
24        $done->send();
25    });
26
27 $done->recv()
```

Чтобы переменные-стражи `$done`, `$w`, `$t` попадали в замыкания функций-колбэков, они должны быть в их области видимости. По этой причине мы сначала объявляем переменные через ключевое слово `my` и только после этого присваиваем значение. Как видно, страж (таймер или страж файлового дескриптора) можно удалить как и любой объект в Perl присвоив ему, например, неопределённое значение. Вызов `$done->recv()` приводит к блокирующему ожиданию, пока в одном из колбэков не будет сделан вызов `$done->send()`.

Модули дистрибутива AnyEvent

Собственно всей этой нехитрой азбуки стражей `AnyEvent` достаточно для реализации задач широкого спектра сложности. Но на практике всё же бывает удобно использовать более высокоуровневый функционал, чтобы не изобретать велосипед каждый раз при написании проектов.

- `AnyEvent::Handle` — это эволюционный потомок стража ввода/вывода `AnyEvent->io`, значительно облегчающий работу с потоковыми сокетами. Поддерживаются очереди чтения и записи, встроенная сериализация/десериализация JSON структур, поддержка TLS и т.п.;
- `AnyEvent::Socket` — специальный модуль для высокоуровневой работы с сокетами (tcp или unix), который предоставляет нам несколько полезных функций, в том числе, `tcp_server` и `tcp_client`, назначение которых ясно из названия. Поддерживается ipv4, ipv6, а также unix-сокеты. Возможно использование SSL/TLS;
- `AnyEvent::Log` — небольшой фреймворк для ведения логов, позволяющий удобно отлаживать приложения, написанные на `AnyEvent`;
- `AnyEvent::Util` — чрезвычайно полезный набор функций, многие из которых являются асинхронными аналогами библиотечных функций:
  - `portable_pipe` — аналог pipe, который работает даже в Windows;
  - `portable_socketpair` — pipe для двустороннего обмена;
  - `fork_call` — асинхронный вызов кода в отдельном процессе и передача возвращаемых данных (через `Storable`) обратно в материнский процесс в указанный колбэк;
  - `run_cmd` — аналог команды `system`, позволяющий запускать программы и асинхронно обрабатывать выводы `STDOUT`, `STDERR`, ввод на `STDIN` и другие открытые файловые дескрипторы.

- `AnyEvent::DNS` — полностью асинхронная реализация разрешения DNS-имён;
- `AnyEvent::IO` — интерфейс для асинхронной работы с файлами. На данный момент этот модуль поддерживает только один бэкенд — `IO::AIO`, который позволяет выполнять неблокирующиеся файловые операции. Как известно Linux не поддерживает неблокирующиеся операции над файлами при использовании стандартных системных вызовов. Именно эту проблему и пытается решить `IO::AIO`. Для этого он запускает подобные вызовы в отдельном треде (при этом не имеет значения собран ли Perl с поддержкой тредов — это внутренняя кухня модуля).

AnyEvent на CPAN

На CPAN появляются всё больше и больше расширений для `AnyEvent`. Перечислять всех смысла нет, можно отметить лишь некоторые интересные.

- `AnyEvent::HTTP` — модуль является своеобразным асинхронным `LWP`, выполняя функции http/https клиента. Есть поддержка прокси, keep-alive соединений и сессий, всех HTTP методов и обработки cookie. Не реализована поддержка HTTP-авторизации, но думаю, что это не за горами;
- `AnyEvent::DBI` — модуль для асинхронной работы с СУБД;
- `Twiggy` — один из самых известных и популярных вебсерверов использующих AnyEvent.

Подводные камни AnyEvent

Начав работать с `AnyEvent` надо всегда помнить о контексте, в котором вызываются функции модулей `AnyEvent`. Многие функции в скалярном контексте возвращают, так называемый охранный объект. Как только охранный объект уничтожается (например, выход

из области видимости), вызываемая функция будет автоматически отменена. При этом если данную функцию вызвать в пустом контексте, охранный объект не будет создаваться и прервать его у вас уже возможности не будет. Так ведёт себя, например, функция `tcp_connect` из модуля `AnyEvent::Socket`.

Создав переменную состояния, никогда не запускайте метод `recv()`, если знаете, что в коде нигде не будет вызываться метод `send()`. Разные реализации СОП ведут себя по-разному в подобной ситуации. `EV`, например, просто начинает загружать процессор на 100%.

Нельзя блокироваться в вызываемых функциях-колбэках. Как только вы попытаетесь вызвать метод `recv()` внутри колбэка, `AnyEvent` обнаружит эту ситуацию и сообщит об ошибке.

Кроме того нельзя использовать `die()` внутри колбэка, поскольку нормально обработать это событие будет затруднительно. Впрочем в EV, будет выдано предупреждение о смерти колбэка, но программа при этом продолжит выполнение.

Не следует менять значение глобальных переменных (таких как `$_` или `$[`) внутри колбэков.

В большинстве СОП библиотек небезопасно использовать `fork`. Это означает, что выполнив `fork` вы не можете начать обрабатывать события в процессе потомке, если до этого в материнском процессе уже была запущена обработка событий. Единственным исключением является библиотека `EV`. Но даже в этом случае процесс потомок получает клоны всех таймеров и стражей I/O, которые начинают работать в обоих процессах, что может быть не совсем то, что вы хотите. Поэтому лучше использовать `fork` до запуска обработки событий или не использовать в дочернем процессе никаких функций `AnyEvent`. Также можно запускать рабочий процесс потомка с помощью комбинации `fork+exec`.

Бенчмарки

Документация `AnyEvent` содержит любопытные результаты бенчмарков по оценке производительности `AnyEvent` с использованием различных бэкендов. В бенчмарке создаётся большое количество I/O стражей, оценивается время создания, уничтожения стражей, потребляемая память на объект и время вызова колбэка.

```
         name watchers bytes create invoke destroy
       EV/EV   100000   223   0.47   0.43    0.27
      EV/Any   100000   223   0.48   0.42    0.26
Coro::EV/Any   100000   223   0.47   0.42    0.26
    Perl/Any   100000   431   2.70   0.74    0.92
 Event/Event    16000   516  31.16  31.84    0.82
   Event/Any    16000  1203  42.61  34.79    1.80
 IOAsync/Any    16000  1911  41.92  27.45   16.81
 IOAsync/Any    16000  1726  40.69  26.37   15.25
    Glib/Any    16000  1118  89.00  12.57   51.17
      Tk/Any     2000  1346  20.96  10.75    8.00
     POE/Any     2000  6951 108.97 795.32   14.24
     POE/Any     2000  6648  94.79 774.40  575.51
```

Результаты подробно поясняются в документации. Как видно, безусловным лидером является `EV`, а аутсайдером — `POE`.

Заключение

Возможно, `AnyEvent` сэкономит вам кучу времени и сил, благодаря своему обширному спектру возможностей. Возможно, у вас появилась идея какой модуль можно написать, чтобы пополнить коллекцию приложений `AnyEvent`. В любом случае знакомство с `AnyEvent` будет полезным и интересным опытом.

■ *crux*

7. Что нового в Perl 5.17.9

20 февраля 2013 г. была выпущена новая версия Perl 5.17.9 для разработчиков. За месяц прошедший с выпуска 5.17.8 были сделаны изменения в примерно 42000 строках кода в 510 файлах 35 авторами.

Поддержка интерполяции в операциях со множествами в регулярных выражениях

В предыдущей версии Perl 5.17.8 появилась новая экспериментальная возможность использовать операции со множествами, такими как пересечение и вычитание, внутри регулярных выражений. Форма записи: `(?[])`. Например, регулярное выражение

```
1 /(?[ \p{Thai} & \p{Digit} ])/
```

совпадает со всеми цифровыми символами в Тайском письме. Поддерживаются операции пересечения (`&`), объединения (`+` или `|`), разности (`-`), симметрической разности (`^`) и унарная операция дополнения (`!`).

В 5.17.9 к этой экспериментальной возможности добавилась поддержка интерполяции, например:

```
1 my $thai_or_lao = qr/\p{Thai} + \p{Lao}/;
2 qr/(?[ \p{Digit} & $thai_or_lao ])/;
```

совпадает с цифровым символом тайского или лаосского письма. Таким образом добавляется поддержка расширения множеств через скомпилированные ранее множества. Данная экспериментальная возможность была введена в соответствии с рекомендациями стандарта Unicode.

Удаление переменной окружения

В версии 5.17.3 была введена новая возможность, когда присвоение переменной окружения `undef` стало эквивалентно удалению данной переменной окружения:

```
1 $ENV{foo} = undef;
2 delete $ENV{foo}
```

В 5.17.9 это изменение было отменено.

Лексическая `$_`

В версии 5.17.7 использование лексической `$_` было помечено как устаревшее. В новом выпуске передумали и решили сделать её экспериментальной.

Прагма `encoding`

Прагма `encoding`, начиная с версии Perl 5.18 и выше, станет устаревшей и возможно будет удалена в более поздних версиях.

Дистрибутивы модулей удалённые из базового состава Perl

`B::Lint`, `CPANPLUS`, `Log::Message`, `Log::Message::Simple`, `Module::Pluggable`, `Object::Accessor`, `Term::UI` удалены из базового состава Perl. Теперь их необходимо устанавливать со CPAN.

Дополнительные символы, которые требуется экранировать в шаблонах /x

В шаблонах регулярных выражений, компилируемых с ключом `/x`, Perl игнорирует 6 символов пробельного типа, такие как пробел и табуляция. Однако в стандарте Юникод рекомендуется рассматривать как пробельные 11 символов. В подготовке соответствия с этим стандартом в будущих версиях Perl, начиная с версии 5.17.9 использование этих недостающих символов в неэкранированном виде будет приводить к выводу предупреждений. Вот эти 5 символов:

```
U+0085 NEXT LINE,
U+200E LEFT-TO-RIGHT MARK,
U+200F RIGHT-TO-LEFT MARK,
U+2028 LINE SEPARATOR,
U+2029 PARAGRAPH SEPARATOR.
```

Новые модули

Добавлен модуль `Config::Perl::V` версии 0.16 для получения структурированной информации вывода perl -V.

Изменения в утилитах

В утилите `corelist` появились опции `--feature` для получения первой версии Perl, в которой появились запрошенные фичи, а также опция `--upstream`, для получения информации о том, где запрошенный модуль поддерживается: в составе Perl или на CPAN с указанием URL на баг-трекер.

Сборка

Появилась новая опция для скрипта `Configure` с длинным названием `useversionedarchname`, при использовании которой

`api_versionstring` добавляется в `archname`. Таким образом имя архитектуры начнёт включать в название версию Perl, например, `x86_64-linux-5.13.6-thread-multi`. По-умолчанию опция отключена.

Исправления

Базовые тесты Perl теперь проходят на системах, на которых отсутствует поддержка локали (например, Android). Слой `:crlf` теперь нормально поддерживает операции чтения наоборот (`unread`), а `ungetc()` корректно обрабатывает UTF-8 данные. Лишний символ `/` в хвосте путей `@INC` больше не появляется.

Теперь, как обычно, запускаем `perlbrew install 5.17.9` и делимся впечатлениями.

■ *crux*

8. Обзор CPAN за февраль 2013 г.

Рубрика с обзором новинок CPAN станет регулярным разделом журнала, рассказывающим о том, какие новые и интересные модули появились на CPAN за прошедший месяц.

Безусловно, понятие «интересный» является очень субъективным, ведь кому-то интересно узнать сколько в этом месяце было выпусков `Mojolicious` и есть ли новые к нему плагины, а другому интересен только раздел `Acme` и он смотрит на CPAN, как на источник «лулзов».

Пытаться угодить всем не получится, но можно попытаться выставить критерии отбора и пытаться им следовать. Это может быть новизна и уникальность идеи для новых модулей. И долгожданный выпуск популярного модуля (ключевое слово «долгожданный») с заметными изменениями для обновлённых. Чтобы не перегружать раздел, постараемся отбирать не более 15–20 модулей.

Статистика

Немного статистики от MetaCPAN за февраль:

- Новых дистрибутивов — 256
- Новых выпусков — 1895

Новые модули

- Dancer 2 Alexis Sukrieh внезапно решил выпустить `Dancer2` в виде отдельного дистрибутива на CPAN. Таким образом, `Dancer2` не станет заменой для `Dancer` и оба проекта будут теперь развиваться независимо.

- Fuse-PerlSSH-FS Модуль, позволяющий монтировать файловые системы с удалённых серверов с помощью FUSE и PerlSSH. Основное отличие от, например, `sshfs` — поддержка расширенных атрибутов (`xattr`).

- Geest Perl порт проекта Kage. Модуль реализует теневой прокси-сервер, который можно использовать для тестирования новой версии бэкенда, сверяя отклики разных серверов на запрос клиента. Модуль примечателен тем, что Kage был написан Tatsuhiko Miyagawa на Ruby. Сам Miyagawa достаточно много портировал идей из мира Python и Ruby в Perl, теперь же сам стал источником для заимствования.

- App::rainbarf Программа позволяет рисовать графики загрузки CPU, RAM и заряда батареи в статусной строке tmux/screen.

- JSON::Pointer Реализация спецификации JSON Pointer draft-09 для Perl. Нечто очень напоминающее XPath, только для JSON:

```
1 JSON::Pointer->get($obj, "/bar/0/qux") эквивалентно
    $obj->{bar}[0]{qux}
```

- Path::Tiny Швейцарский нож для манипуляций с путями и файлами. David Golden собрал в один флакон практически все мыслимые функции.

- Panda::Date Совместимый с `Class::Date` модуль для работы с датами и временем, реализованный на C. Утверждается, что работает в 40–70 раз быстрее `Class::Date`.

- Mango Клиент MongoDB с неблокируемым I/O. В полку клиентов MongoDB пополнение от небезызвестного Sebastian Riedel. Модуль оптимизирован для использования совместно с `Mojolicious`.

Обновлённые модули

- App-cpanminus 1.6001 Выпущена новая мажорная версия лучшей утилиты для установки модулей со CPAN. В новой версии появилась возможность установки определённой

версии модулей, выбор версии из диапазона, установки экспериментальных версий для разработчиков. Используется API MetaCPAN для поиска версий на MetaCPAN и BackPAN. Возможность установки модуля непосредственно из git-репозитория (даже если его нет на CPAN!).

- MCE 1.4004 Модуль предоставляет инструмент для параллельной обработки данных. MCE запускает пул из нескольких рабочих процессов, и распределяет входные данные для обработки между процессами.

- Promises 0.03 Реализация модели *обещаний* в асинхронном программировании для Perl. Данная модель пришла из спецификации *Promises/A*, созданной сообществом Node.JS. Суть модели заключается в том, что результат запуска асинхронного действия — это обещание. Обещание может быть неисполненным, исполненным или быть неудавшимся. Обещание, как объект, имеет метод `then`, который выполняет соответствующие действия, как только наше обещание исполняется или происходит ошибка. Отличие от обычного подхода, когда мы пишем колбеки для каждого вызова асинхронной функции (и наш исходный код превращается в спагетти, которое уезжает в правую сторону экрана с каждым новым колбеком), в том, что мы можем группировать обещания в цепочку и выполнять действия после того как все наши обещание исполнились. При этом модуль не накладывает никаких ограничений на выбор модуля событийного программирования.

- Sereal 0.310 Очередной модуль для сериалазации/десериализации данных. Основное достоинство — в бенчмарках он рвёт в клочья все другие модули из этого класса: `JSON::XS`, `Data::MessagePack`, `Storable` и т.д.

- Perlbrew 0.59 Программа, которая позволяет вам устанавливать и управлять инсталляциями Perl внутри домашнего каталога. В новой версии незначительные исправления.

- Cv 0.24 Perl интерфейс к библиотеке OpenCV. Всем кто интересуется проблемами компьютерного *зрения* и системами распознавания образов знакомство будет крайне полезным.

- Text::Xslate 2.0000 Самый быстрый шаблонизатор для Perl `Text::Xslate` обновлён до новой мажорной версии 2. Основное изменение — переход с `Any::Moose` на `Mouse`, поскольку `Any::Moose` теперь считается устаревшим.

- Git::Raw 0.22 Модуль является Perl обвязкой к `libgit2`. Основное отличие модуля от других реализующих интерфейс к git-репозиториям — он не является обёрткой вокруг команды `git`, а использует C-библиотеку `libgit2`. Модуль работает с нестабильным снапшотом библиотеки, поэтому использовать его в боевом окружении достаточно рискованно.

■ *crux*

9. Интервью с Tatsuhiko Miyagawa

Tatsuhiko Miyagawa — известный японский программист. В мире Perl он автор и сопровождающий таких модулей как cpanm, carton, Plack/PSGI, Starman, Twiggy, Web::Scraper, Plagger и многих других. Возглавляет рейтинг Perl-программистов на GitHub (github-meets-cpan.com). В 2008 году получил награду White Camel за активность в Perl-сообществе и, в частности, за соорганизацию конференций YAPC::Asia, которые являются самыми посещаемыми из всех остальных YAPC. Ходят слухи, что он никогда не спит.

Как и когда начал изучать программирование?

Это было частью компьютерного курса в течение первого семестра в университете.

Какой редактор используешь?

Emacs.

Как и когда познакомился с Perl?

Я занимался не связанной с компьютерами работой в O'Reilly Japan, это было также в течение первого семестра в университете. Тогда был первый интернет бум и было множество «личных страничек», и я подумал, что стоит попробовать сделать какой-нибудь форум с помощью Perl. Вот так вот все и началось.

С какими другими языками программирования приятно работать?

Ruby, немного Python и JavaScript. Среди них больше нравится Ruby.

Какое, по-твоему, самое большое преимущество Perl?

CPAN и сообщество. Также производительность и UNIX-направленность.

Какая, по-твоему, характеристика наиболее важна для языков будущего?

Я не знаю.

Что думаешь по поводу дебатов вокруг версий Perl?

Без комментариев.

Что мотивирует тебя на такую активность в Perl-сообществе и open source в целом?

Это весело выносить компоненты, которые могут использоваться независимо, в модули, загружать на CPAN, получать карму, сообщения об ошибках и исправления к ним, и все это — за просто так. Это приносит такое удовлетворение и привыкание, что нет необходимости думать об этом, это всего лишь часть моего процесса разработки.

В документации к Twiggy можно найти фразу «Возможно я на веществах» (“Maybe I’m on drugs”). Какие вещества, по-твоему, работают лучше при программировании? :)

Спроси mst (*Matt S Trout* — прим. ред.).

Где и над чем сейчас работаешь?

Я в Сан-Франциско, и, так как сегодня выходной, я не программирую.

Проектируешь свои программы заранее или в процессе? Что думаешь насчет рефакторинга и тестирования?

Обычно я начинаю с выбора имени. Это самая важная часть и часто занимает очень много времени (обычно дни) прежде чем я открою файл в редакторе. Затем я начинаю с документации, чтобы обдумать как использовать программу, затем к тестам и собственно кодированию. И вначале тесты насколько это возможно. Для простых «написал и выбросил» программ, я пропускаю эти части и начинаю сразу программировать, но для модулей и рабочих проектов я выполняю цикл: документация → тесты → код.

Стоит ли сейчас советовать молодым программистам учить

Perl?

Конечно, но им стоит также учить другие языки, такие как Ruby, Python и JavaScript — чтобы лучше представлять преимущества и недостатки Perl.

■ *vti*

10. 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!

1. Среди русских и украинских авторов (на 28 февраля 2013 г.) по количеству модулей на CPAN на первом месте ZOFFIX, а кто же на втором?

    1. ZAG
    2. KARASIK
    3. ILYAZ
    4. RUZ
    5. MONS

2. Кто выпустил один из релизов Perl 5 прямо на Perl-конференции во время своего доклада?

    1. Nicholas Clark
    2. Florian Ragwitz
    3. Shlomi Fish
    4. Abigail
    5. Ricardo Signes

3. В каком из этих российских городов нет PM-группы?

    1. Кострома
    2. Москва
    3. Ростов-на-Дону
    4. Владивосток
    5. Новосибирск

4. Сколько сборок Perl 5 существует для MSWin32?

1. 1
2. 3
3. 4
4. 5
5. 7

5. Какого из этих «секретных» операторов не существует?

   1. Червяк на палочке
   2. Космическая станция
   3. Двухклинковый меч с украшениями
   4. Скипетр
   5. Подмигивающая длинная запятая

6. Как называется фишка, которую раздают на различных мероприятиях и выставках где участвует Perl, символизирующая свободное время (которого в данный момент нет) для релиза модуля или исправления ошибки.

   1. tait
   2. tuit
   3. toto
   4. tata
   5. tito

7. Почему логотипом The Perl Foundation является лук?

   1. Он символизирует многогранное Perl-сообщество.
   2. Это название лука-порея, которое перекликается с изначальным названием языка Perl.
   3. От речи The State of the Onion, которую Ларри Уолл произносил на различных конференциях.
   4. Символ выбран чисто случайно и принят после голосования на конференции YAPC::Europe 2003.
   5. Верблюд является эволюционным братом лука.

8. Какая из этих фраз не принадлежит Ларри Уоллу?

    1. Плохие программы пишет программист, хорошие — язык.
    2. 111% дерьма это все.
    3. Это нормально быть временно неправым.
    4. Задача большинства языков программирования это увеличить длину резюме на слово и запятую.
    5. Размер собственной головы можно измерить только изнутри.

9. Где был проведен первый Perl-воркшоп?

    1. США
    2. Англия
    3. Германия
    4. Франция
    5. Канада

10. Сколько подписчиков на The Perl Journal в 1996 г. было из Украины?

    1. 2
    2. 5
    3. 15
    4. 30
    5. 45

■ *vti*