Содержание

Предисловие

Предисловие к третьему изданию

Вчера я читал статью на сайте Wired.com о моде в среде компьютерных фриков. Так там пишут, что если человек носит футболку с надписью Rubyconf 2012, значит, он хочет сообщить «Я работаю в Oracle».

Ну надо же! Далеко же мы ушли за последние 10 лет!

Было время, когда Ruby определенно выбивался из господствующих тенденций. Но теперь мы, похоже, в струе. Однако для этого пришлось пройти долгий и необычный путь.

По нынешним стандартам, чтобы войти в обиход, Ruby потребовалось много времени. Я читал эту книгу в 2005 году, и уже тогда первому изданию исполнилось четыре года. Тогда только-только намечался второй всплеск интереса к Ruby благодаря DHH и началу повальной увлеченности Rails. Казалось, что во всем англоговорящем мире не наберется и двух сотен человек, использующих Ruby. А первому изданию книги уже было четыре года. Вот насколько она опередила свое время.

В новом издании книги сохранен стиль, завоевавший ей признание у опытных программистов. В первых четырех длинных главах описываются основы объектно-ориентированного подхода и самого языка Ruby. Их обязательно должен прочитать всякий, кто не знаком с языком. Но изложение стремительное, без задержки на деталях – предполагается, что читатель уже знает, как создается программное обеспечение.

Последующие главы устроены иначе. Краткая предыстория, а затем мощный залп сведений о языке Ruby. Для иллюстрации обсуждаемой темы приводятся многочисленные фрагменты кода. Примеры можно вставлять в собственные программы практически без изменений – особенно когда вы дойдете до глав, в которых рассматриваются практические приложения.

Будет уместно сказать пару слов о себе. Я очень благодарен Хэлу за эту книгу и за то, как именно он ее написал. В 2005 году по договору с издательством Addison Wesley я начал работать над книгой об использовании Ruby on Rails на предприятии. То была моя первая попытка попробовать себя в роли автора и, сочинив две главы, я застрял. В то время Ruby и Rails можно было встретить на предприятиях лишь эпизодически, и я должен был постоянно напоминать себе, что пишу не беллетристику.

Обсудив варианты с редактором, мы решили, что будет правильнее отказаться от первоначальной идеи и подойти к книге по-другому. Книга «Путь Rails» стала попыткой осветить только зарождающийся каркас Ruby on Rails в духе, присущем этой книге. Я взял на вооружение краткий повествовательный стиль с изобилием приме-

ров кода. Вместо длинных листингов я чередовал код и комментарии, стремясь не переусердствовать, а просто проиллюстрировать смысл отдельных частей каркаса.

Как и в «Пути Ruby», я ставил себе целью добиться ширины охвата, а не глубины рассмотрения. Я хотел, чтобы «Путь Rails» занял постоянное место на столах серьезных разработчиков приложений под Rails. Я хотел, чтобы моя книга, как и «Путь Ruby», стала справочным пособием по умолчанию. В отличие от других книг по Rails, я полностью опустил вводный материал и проигнорировал интересы начинающих.

И это был успех! Будет справедливо сказать, что без книги Хэла не было бы и моей книги, а моя карьера сложилась бы не так удачно.

Но довольно ретроспективных поздравлений. Вернемся к дню сегодняшнему и только что вышедшему изданию «Пути Ruby», которое вы сейчас читаете. На этот раз к Хэлу присоединился безмерно талантливый Андрэ Арко[1]. Отличная получилась команда! Изрядно потрудившись, они привели текст в соответствие с последней версией всеми нами любимого языка Ruby.

Лично я хотел бы отметить следующие отличия от предыдущих изданий.

- Целая глава, посвященная углубленному рассмотрению новой подсистемы регулярных выражений Onigmo. Мне очень нравятся их красивые и краткие объяснения таких концепций, как позитивное и негативное заглядывание и оглядывание.
- В главе об интернационализации обсуждаются трудные вопросы, касающиеся кодировки объектов типа String и нормализации в Unicode. Блогеры уже много лет обсуждают различные аспекты этой темы, но наконец-то все собрано в одном месте.
- В главе о приложениях Ruby в веб авторам удалось соединить в одном кратком курсе, занимающем меньше 30 страниц, начальные сведения о Rack, Sinatra и Rails.

Я предсказываю этому изданию «Пути Ruby» не меньший успех, чем предыдущим. С удовольствием включаю его в нашу серию книг «Professional Ruby».

Оби Фернандес
15 сентября 2014

Предисловие ко второму изданию

В древнем Китае люди, в особенности философы, полагали, что под внешней оболочкой мира и любого существа скрыто нечто. Его нельзя ни объяснить, ни описать словами. Это нечто китайцы называли Тао, а японцы – До. На русский язык это слово можно перевести как Путь. Слово «до» входит в такие названия, как дзюдо, кендо, карате-до и айкидо. Это не просто боевые искусства, а целая философия и взгляд на жизнь.

---

[1] Желаете увидеть пример изобретательности Андрэ? Почитайте статью по адресу http://andre.arko.net/2014/06/27/rails-in-05-seconds/, где он описывает, как уменьшить время загрузки реального приложения Rails до 500 мс, а то и меньше.

Так и в языке программирования Ruby есть своя философия и способ мышления. Этот язык заставляет думать по-новому. Он помогает программистам получать удовольствие от своей работы. И не потому, что Ruby был создан в Японии, а потому что программирование стало важной частью существования (по крайней мере, для некоторых людей), а Ruby призван улучшить жизнь.

Как всегда, описать, что такое Tao, трудно. Я чувствую, но не могу подыскать нужных слов. Это трудно сделать даже на японском, моем родном языке. Но парень по имени Хэл Фултон попытался, и его первая попытка (первое издание этой книги) оказалась довольно удачной. А результат второго подхода к задаче описать Tao Rubi еще лучше, чему немало способствовала помощь многих людей из сообщества пользователей Ruby. По мере того как Ruby набирает популярность (отчасти благодаря продукту Ruby on Rails), все важнее становится овладение секретами мастерства производительного программирования на этом языке. Надеюсь, что эта книга поможет вам в решении этой задачи.

Удачной работы.

Юкихиро «Мац» Мацумото
Август 2006, Япония

Предисловие к первому изданию

Вскоре после того, как я впервые познакомился с компьютерами в начале 1980-х годов, меня заинтересовали языки программирования. И с тех пор я помешался на этой теме. Думаю, что причина такого интереса состоит в том, что языки программирования – это способ выражения мыслей. Они по сути своей предназначены для человека.

Но вопреки этому факту языки программирования почему-то всегда оказывались в большей степени машинно-ориентированными. Многие из них спроектированы с учетом удобства для компьютеров.

По мере того как компьютеры становятся мощнее и дешевле, ситуация постепенно меняется. Возьмем к примеру структурное программирование. Машине все равно, насколько хорошо структурирована программа, она просто исполняет ее команда за командой. Идеи структурного программирования обращены к людям, а не к машинам. То же относится и к объектно-ориентированному программированию.

Пришло время проектировать языки, удобные для людей.

В 1993 году я разговаривал со своим коллегой о сценарных языках, их выразительности и перспективах. Я считал, что программирование пойдет именно по этому пути и будет ориентироваться на человека.

Но я не был удовлетворен такими существующими языками, как Perl и Python. Я хотел видеть язык, более мощный, чем Perl, и более объектно-ориентированный, чем Python. Найти идеальный язык мне не удалось, поэтому я решил изобрести свой собственный.

Ruby – не самый простой язык, но и человеческая душа не проста. Ей равно нравятся простота и сложность. Она не приемлет ни слишком простых, ни слишком сложных вещей. Она ищет равновесия.

Поэтому при проектировании ориентированного на человека языка – Ruby – я следовал принципу наименьшего удивления. Я считал, что хорошо то, что не кажется мне странным. Поэтому я ощущаю себя естественно и даже испытываю радость, когда программирую на Ruby. А с момента выхода в свет первой версии в 1995 году многие программисты во всем мире разделили со мной эту радость.

Как всегда, я хочу выразить величайшую благодарность всем членам сообщества, сложившегося вокруг Ruby. Они – причина успеха Ruby.

Я благодарен также автору этой книги, Хэлу Фултону за то, что он показал другим Путь Ruby.

В этой книге объясняется философия, стоящая за языком Ruby. Это квинтэссенция моих мыслей и ощущений членов сообщества. Интересно, как Хэлу удалось прочитать мои мысли и раскрыть секрет Пути Ruby. Я никогда не встречался с ним лично, надеюсь, что скоро это все-таки произойдет.

Я хотел бы, чтобы эта книга и сам язык Ruby помогли вам получить удовольствие и радость от программирования.

Юкихиро «Мац» Мацумото
Сентябрь 2001, Япония

Об авторах

Хэл Фултон начал использовать Ruby в 1999. В 2001 году он приступил к работе над книгой «The Ruby Way», второй книгой об этом языке на английском языке. Фултон был участником самой первой конференции по Ruby, состоявшейся в 2001 году, и выступал с докладами на многих других конференциях, проходивших на трех континентах, в том числе на первой европейской конференции по Ruby в 2003 году. Он обладатель двух ученых степеней по информатике, полученных в Университете штата Миссисипи, и в течение четырех лет преподавал информатику. Больше 25 лет он работает с различными версиями операционных систем UNIX и Linux. Сейчас он работает в компании Simpli.fi, которая находится в городе Форт Уорт, штат Техас, где пишет преимущественно на Ruby.

Андрэ Арко впервые познакомился с Ruby в 2004 году, когда еще учился в университете, а первое издание этой книги сыграло немалую роль в его решении строить карьеру программиста на Ruby. Он руководит проектом Bundler, менеджером зависимостей Ruby, и является автором или соавтором десятка других проектов с открытым исходным кодом. Андрэ работает консультантом в компании Cloud City Development, где отвечает за обучение сотрудников Ruby и Rails, а также занимается разработкой веб-приложений.

Андрэ с удовольствием делится добытыми тяжким трудом знаниями и опытом с другими разработчиками, он выступал на дюжине конференций по Ruby, проходивших на четырех континентах. Он регулярно работает волонтером на информационных мероприятиях RailsBridge и RailsGirls, посвященных программированию, и всячески стремится увеличить разнообразие и расширить качественный состав сообщества Ruby и способствовать признанию этой технологии как отрасли знаний. Живет он в Сан-Франциско, штат Калифорния.

Введение

Путь, который можно описать словами, –
это не истинный Путь.

– Лао Цзы, «Tao Te Ching»

Эта книга называется «Путь Ruby». Название нуждается в некотором пояснении.

Я ставил себе целью выразить в этой книге философию языка Ruby, насколько это в моих силах. Ту же цель преследовали мои добровольные помощники. Успех должно разделить между всеми, а ошибки остаются моей и только моей виной.

Конечно, я не могу абсолютно точно сказать, в чем же состоит истинный дух Ruby. Эту задачу я оставляю Мацу, но подозреваю, что даже ему трудно будет выразить ее словами.

Короче говоря, «Путь Ruby» – это всего лишь книга, а Путь Ruby – удел создателя языка и сообщества в целом. Втиснуть его в рамки книги довольно трудно.

И все же я попытаюсь в этом введении поймать неуловимый дух Ruby. Мудрый ученик не воспримет эту попытку как окончательный вердикт.

О третьем издании

Все меняется, и Ruby – не исключение. В это издание внесено немало изменений и добавлено много нового материала. В каком-то смысле все главы книги «новые». Я подверг пересмотру и переработке каждую, внес сотни мелких и десятки крупных изменений. Я исключил вещи, которые устарели или утратили значимость, изменил материал, так чтобы он лучше соответствовал самому языку Ruby, добавил примеры и комментарии.

Будучи второй книгой о Ruby на английском языке (после «Programming Ruby» Дэйва Томаса и Энди Ханта), «Путь Ruby» был построен так, чтобы служить дополнением к первой, не пересекаясь с ней; это верно и сегодня.

Между версией Ruby 1.8, рассматривавшейся во втором издании, и текущей версией 2.1 существует много различий. Но важно понимать, что изменения вносились очень бережно, на протяжении нескольких лет. Ruby по-прежнему остается Ruby. Своей красотой Ruby в немалой степени обязан политике неторопливых и продуманных изменений, мудро направляемой Мацем и другими разработчиками.

Сегодня нет недостатка в книгах по Ruby, а статей публикуется больше, чем мы в состоянии охватить взглядом. Пособия и документация представлены в веб в изобилии.

Появились новые инструменты и библиотеки. Наиболее распространенные — инструменты, создаваемые одними разработчиками для других: веб-каркасы, средства ведения блогов, средства разметки и интерфейсы к экзотическим хранилищам данных. Но, конечно, есть и много других: графические интерфейсы, средства численных расчетов, веб-службы, обработка изображений, управления версиями исходного кода и т. д.

Поддержка Ruby включена практически во все редакторы и весьма развита. Имеются полезные и зрелые интегрированные среды (IDE), которые отчасти включают средства построения графического интерфейса пользователя (ГИП).

Безусловно, значительно выросло и изменилось сообщество. Сегодня Ruby уже никто не назовет нишевым языком; он используется в государственных учреждениях, например в НАСА и в Национальном управлении океанических и атмосферных исследований (NOAA), в корпорациях, например IBM и Motorola, и на таких хорошо известных сайтах, как Wikipedia, GitHub и Twitter. Он применяется для создания графических приложений, работы с базами данных, численных расчетов и многого другого. Короче говоря, Ruby вошел в струю – я говорю в самом что ни на есть положительном смысле.

Я занимался переработкой этой книги с любовью. Надеюсь, что плод моих трудов окажется вам полезен.

Как организована эта книга

Вряд ли вы станете изучать Ruby по этой книге. В ней не так уж много вводного и учебного материала. Если вы еще ничего не знаете о Ruby, то лучше начать с какой-нибудь другой книги.

Но при этом программисты – народ упорный, и я допускаю, что научиться Ruby только по этой книге возможно. В главе 1 «Обзор Ruby» приводится краткое введение в язык и очень скромное руководство.

Также в главе 1 есть довольно полный перечень «скользких мест» (который трудно поддерживать в актуальном состоянии). Для разных читателей этот перечень полезен в разной мере, поскольку что для одного интуитивно очевидно, для другого выглядит странно.

В основном, эта книга призвана отвечать на вопросы типа «Как сделать?». И потому вы, вероятно, многое будете пропускать. Я почту за честь, если кто-то прочтет книгу от корки до корки, но не надеюсь на это. Скорее я ожидаю, что вы будете искать в оглавлении темы, которые вас интересуют в конкретный момент.

Впрочем, с момента выхода первого издания я беседовал с разными людьми, и оказалось, что многие прочли книгу целиком. Более того, несколько человек писали мне, что выучили по ней Ruby. Что ж, все возможно.

Некоторые рассматриваемые в книге вопросы могут показаться элементарными. Но ведь у разных людей и опыт разный; то, что очевидно одному, будет откровением для другого. Я старался сделать изложение как можно более полным. С другой стороны, было стремление уложиться в разумный объем (ясно, что эти цели противоречивы).

Можно назвать эту книгу «справочником наоборот». Вы ищете то, что нужно, не по имени класса или метода, а по функции или назначению. Например, в классе `String` есть несколько методов для манипулирования регистром букв: `capitalize`, `upcase`, `casecmp`, `downcase` и `swapcase`. В настоящем справочнике они встречались бы в алфавитном порядке, а в этой книге собраны в одном месте.

Конечно, в борьбе за полноту я иногда сворачивал на путь, которому следуют справочные руководства. Во многих случаях я старался компенсировать это, предлагая не совсем обычные примеры или разнообразя их по сравнению со справочниками.

Я старался не перегружать код комментариями. Если не считать первой главы, то думаю, что достиг этой цели. Писатель может стать не в меру болтливым, но программист-то хочет видеть код (а если не хочет, то должен хотеть).

Иногда примеры выглядят искусственными, за что я приношу свои извинения. Проиллюстрировать какой-то прием или принцип в отрыве от реальной задачи бывает сложно. Но чем сложнее задача, чем выше ее уровень, тем большие усилия я прилагал к подысканию реального примера. Так, если речь идет о конкатенации строк, то, наверное, вы увидите безыскусный фрагмент кода с упоминанием пресловутых “`foo`” и “`bar`”, но когда рассматривается тема разбора XML-документа, будет приведен куда более содержательный и реалистичный пример.

Есть в этой книге два-три каприза, в которых хочу заранее сознаться. Во-первых, я всеми силами старался избегать «уродливых» пришедших из языка Perl глобальных переменных типа `$_` и ей подобных. Они есть в Ruby и прекрасно работают, даже применяются в повседневной работе всеми или большинством программистов. Но почти всегда от их использования можно уйти, что я и позволил себе чуть ли не во всех примерах.

Другой каприз состоит в том, что я избегаю пользоваться обособленными выражениями, если у них нет побочных эффектов. В Ruby выражения – одна из основ языка, и это прекрасно, я старался извлечь их этой особенности максимум пользы. Но во фрагментах кода я предпочитаю не употреблять выражения, которые просто возвращают никак не используемое значение. Например, для иллюстрации конкатенации строк достаточно было бы написать `"abc" + "def"`, но я в этом случае напишу что вроде `str = "abc" + "def"`. Кому-то это покажется многословием, но выглядит естественным для программиста на языке C, привыкшего к тому, что бывают функции типа void и не-void (а также программисту на Pascal’e, мыслящему в терминах процедур и функций).

Третий каприз заключается в моем нежелании употреблять символ решетки для обозначения методов экземпляра. Многие поклонники Ruby считают, что я проявляю излишнюю болтливость, когда пишу «метод экземпляра `crypt` класса `String`», а не просто `String#crypt`, но я полагаю, что так никто не запутается. (На самом деле, я постепенно смиряюсь с использованием такой нотации, так как ясно, что она уже не исчезнет.)

Я старался давать ссылки на внешние ресурсы там, где это уместно. Ограничения по времени и объему не позволили мне включить в книгу все, что я хотел бы, но надеюсь, что это хотя бы отчасти компенсируется указаниями на то, где найти

недостающую информацию. Из всех источников самым главным, наверное, следует считать архив приложений Ruby (Ruby Application Archive) в сети; вы не раз встретите ссылки на него.

В начале книги принято приводить соглашения об использовании шрифтов, применяемых для выделения кода, и о том, как отличить пример от обычного текста. Но я не стану оскорблять вас недоверием к вашим умственным способностям, все вы и раньше читали техническую литературу.

Хочу подчеркнуть, что примерно 10% текста книги было написано другими людьми. И это не считая технического редактирования и корректуры. Вы просто обязаны прочитать благодарности, приведенные в этой (и любой другой) книге. Большинство читателей пропускают их. Прошу, прочтите прямо сейчас. Это будет так же полезно, как питание овощами.

Об исходных текстах, приведенных в книге

Все сколько-нибудь значительные фрагменты кода собраны в архив, который можно загрузить из сети. Этот архив есть на сайте informit.com и на сайте самой книги therubyway.io.

Он предлагается в виде tgz-файла и в виде zip-файла. Совсем короткие фрагменты, которые нельзя исполнить «вне контекста», в архив обычно не включаются.

Что такое «Путь Ruby»?

Попробуем сразиться со злом, не выразимым словами,
и кто знает, возможно, мы его все же одолеем.

— Дуглас Адамс
«Детективное агентство Дирка Джентли»

Что мы имеем в виду, говоря о Пути Ruby? Я полагаю, что тут есть два взаимосвязанных аспекта: философия проектирования Ruby и философия использования этого языка. Естественно, что дизайн и применение связаны друг с другом, будь то программное или аппаратное обеспечение. А иначе зачем бы существовала наука эргономика? Если я снабжаю устройство ручкой, то, наверное, предполагаю, что кто-то за эту ручку возьмется.

В языке Ruby имеется невыразимое словами качество, которое делает его тем, что он есть. Мы наблюдаем это качество в дизайне синтаксиса и семантики языка, присутствует оно и в написанных на нем программах. Но стоит попытаться сформулировать, в чем состоит эта отличительная особенность, как она размывается.

Очевидно, Ruby – не просто инструмент для написания программ, но и сам по себе является программой. Почему работа программ, написанных на Ruby, должна следовать законам, отличным от тех, которым подчинена работа интерпретатора? В конце концов, Ruby – исключительно динамичный и расширяемый язык. Могут найтись причины, по которым эти два уровня где-то расходятся, быть может, ста-

раясь приспособиться к несовершенству реального мира. Но в общем случае мыслительные процессы могут и должны быть сходными. Интерпретатор Ruby можно было бы написать на самом Ruby, в полном соответствии с принципом Хофштадтера, хотя в настоящее время это еще не сделано.

Мы не часто задумываемся над этимологией слова «путь», но оно употребляется в двух разных смыслах. Во-первых, это метод или техника, а во-вторых – дорога. Ясно, что оба значения взаимосвязаны, и, говоря «путь Ruby», я имею в виду как то, так и другое.

Следовательно, мы говорим о мыслительном процессе, но одновременно и о дороге, по которой движемся. Даже величайшие гуру программирования не могут сказать о себе, что достигли совершенства, они лишь идут по дороге к нему. Таких дорог может быть несколько, но я здесь говорю только об одной.

Привычная мудрость гласит, что форма определяется функцией. Это, конечно, правильно. Но Фрэнк Ллойд Райт[1] (имея в виду свою собственную область интересов) как-то сказал: «Форма определяется функцией, которая было понята неправильно. Форма и функция должны быть едины, сливаться в духовном единении».

Что Райт имел в виду? Я бы сказал, что на этот вопрос вы найдете ответ не в книгах, а в собственном опыте.

Однако я думаю, что Райт выразил эту мысль где-то еще, разбив ее на части, которые проще переварить. Он был великим поборником простоты, который однажды заметил: «Самые полезные инструменты архитектора – это ластик рядом с чертежной доской и гвоздодер на строительной площадке».

Итак, одним из достоинств Ruby является простота. Надо ли цитировать других мыслителей, высказывавшихся на эту тему? Согласно Антуану де Сент-Экзюпери: «Совершенство достигнуто не тогда, когда нечего добавить, а тогда, когда нечего убрать».

Но Ruby – сложный язык. Почему же я называю его простым?

Если бы мы лучше понимали мироздание, то, наверное, открыли бы «закон сохранения сложности» – факт, который вмешивается в нашу жизнь подобно энтропии, которую мы не можем избежать, а способны лишь рассеивать.

И в этом ключ. Нельзя избежать сложности, но можно укрыться от нее. Мы можем убрать ее из виду. Это тот же старый добрый принцип черного ящика, внутри которого решается сложная задача, хотя на поверхности лишь голая простота.

Если вам еще не наскучили цитаты, то будет уместно привести слова Альберта Эйнштейна: «Все должно быть просто настолько, насколько возможно, но не проще».

Таким образом, на взгляд программиста, Ruby – это воплощенная простота (хотя у человека, отвечающего за сопровождение интерпретатора, взгляд может быть иной). Но вместе с тем имеется пространство для компромиссов. В реальном мире всем нам приходится немного сгибаться. К примеру, все сущности в програм-

---

[1] Фрэнк Ллойд Райт (1867–1959) – знаменитый архитектор и дизайнер. Одна из самых известных работ – музей Музей Гуггенхайма в Нью-Йорке. – *Прим. перев.*

ме на Ruby должны были бы быть истинными объектам, однако некоторые, в том числе целые числа, хранятся как непосредственные значения. Это компромисс, знакомый всем студентам отделений информатики уже много десятилетий: элегантность дизайна приносится в жертву практичности реализации. По существу, мы обменяли одну простоту на другую.

То, что Ларри Уолл говорил о языке Perl, остается справедливым: «Когда вы хотите что-то выразить на маленьком языке, оно становится большим. А когда вы пытаетесь выразить то же самое на большом языке, оно становится маленьким.» Это верно и в отношении английского языка. Если биолог Эрнст Хэккель смог всего тремя словами выразить глубокую мысль «онтогенез повторяет филогенез», то лишь потому, что эти слова с весьма специфическим смыслом были в его распоряжении. Мы соглашаемся на внутреннюю сложность языка, потому что она позволяет избежать сложности в отдельных высказываниях.

Переформулирую этот принцип по-другому: «не пишите 200 строк кода, когда достаточно 10».

Я считаю само собой разумеющимся, что краткость в общем случае хороша. Короткий фрагмент кода занимает меньше места в мозгу программиста, его проще воспринять как единое целое. Благоприятным побочным эффектом следует считать и то, что в короткой программе будет меньше ошибок.

Конечно, не нужно забывать предупреждение Эйнштейна о простоте. Если расположить краткость слишком высоко в списке приоритетов, то получится совершенно загадочный код. Согласно теории информации, сжатые данные статистически похожи на белый шум. Если вы видели код на C или APL или регулярное выражение – особенно плохо написанные, то понимаете, что я имею в виду. «Просто, но не слишком просто» – это ключевая фраза. Стремитесь к краткости, но не жертвуйте понятностью.

Трюизмом следует считать мысль о том, что краткость в сочетании с понятностью – это хорошо. Но тому есть причина, причем настолько фундаментальная, что мы часто о ней забываем. А состоит она в том, что компьютер создан для человека, а не человек для компьютера.

В старые добрые дни все было почти наоборот. Компьютеры стоили миллионы долларов и пожирали многие киловатты электричества. Люди же вели себя так, будто компьютер – божество, а программисты – его скромные жрецы. Час машинного времени стоил дороже часа личного времени.

Когда компьютеры стали меньше и дешевле, приобрели популярность языки высокого уровня. Они неэффективны с точки зрения машины, зато эффективны с позиции человека. Ruby – это просто одно из последних достижений на этом пути. Некоторые даже называют его языком сверхвысокого уровня (VHLL – Very High-Level Language). Хотя этот термин еще не получил четкого определения, я думаю, что он оправдан.

Компьютер призван быть слугой, а не хозяином, а, как сказал Мац, толковый слуга должен выполнять сложное задание при минимуме указаний. Так было на протяжении всей истории информатики. Мы начали с машинного языка, перешли к языку ассемблера, а потом добрались и до языков высокого уровня.

Мы сейчас говорим о смещении парадигмы: от машинно-центрической к человеко-центрической. На мой взгляд, Ruby дает великолепный пример человеко-центрического программирования.

Я хочу взглянуть на вопрос под несколько иным углом. В 1980 вышла чудесная книжка Джеффри Джеймса «Тао программирования» (*The Tao of Programming*, Geoffrey James). Каждая строчка из нее достойна цитирования, но я ограничусь лишь одной выдержкой: «Программа должна следовать 'закону наименьшего удивления'. Что это за закон? Все просто: программа должна отвечать пользователю так, чтобы вызывать у него как можно меньше удивления.» (Конечно, если речь идет об интерпретаторе языка, то пользователем является программист.)

Не знаю, Джеймс ли придумал этот термин, но я впервые узнал его из этой книги. Этот закон хорошо известен и часто цитируется в сообществе пользователей Ruby. Правда, обычно его называют «Принципом Наименьшего Удивления» или POLS (Principle of Least Surprise). (Лично я упрямо придерживаюсь акронима LOLA – Law of Least Astonishment.)

Но, как ни называй, правило остается справедливым и служит основополагающим принципом продолжающейся работы над языком Ruby. О нем полезно помнить и тем, кто разрабатывает библиотеки и пользовательские интерфейсы.

Конечно, одна проблема остается: разные люди удивляются разным вещам; не существует всеобщего согласия о том, как «должен» вести себя объект или метод. Но мы можем стремиться быть последовательными и находить веские обоснования принимаемым проектным решениям, а каждый человек должен тренировать собственную интуицию.

Кстати, Мац как-то заметил, что «принцип наименьшего удивления» должен относиться и к нему как к дизайнеру. Чем больше ваше мышление походит на его, тем меньше удивления будет вызывать Ruby. И смею уверить, подражание Мацу большинству из нас только пойдет на пользу.

Какой бы ни была логическая конструкция системы, тренировать свою интуицию необходимо. Каждый язык программирования – это отдельный мир, со своими допущениями, в этом они ничем не отличаются от естественных языков. Когда я начал изучать немецкий, то обнаружил, что все существительные пишутся с заглавной буквы. За исключением слова *deutsch*. Я пожаловался профессору, подчеркивая, что ведь это же *название* самого языка. Он улыбнулся и ответил: «Не надо с этим бороться».

Он говорил, что надо дать немцу оставаться немцем. Продолжая эту мысль, хочу дать совет всем, кто приходит в Ruby из других языков. Пусть Ruby остается Ruby. Не ожидайте, что это будет Perl, это не так. Не ждите от него поведения, характерного для языков LISP или Smalltalk. С другой стороны, у Ruby есть элементы, присущие любому из этих трех языков. Для начала действуйте в соответствии с априорными представлениями, но когда они оказываются неверны, не боритесь с языком. (Если только Мац не согласится с тем, что необходимо изменение.)

Каждый программист сегодня знает о принципе ортогональности (хотя лучше было бы назвать его принципом *ортогональной полноты*). Вообразим пару осей, по одной из которых откладываются языковые сущности, а по другой – множество

атрибутов и возможностей. Когда мы говорим об «ортогональности», то обычно имеем в виду, что пространство, определяемое этими осями настолько «полно», насколько это логически возможно.

Одна из составных частей Пути Ruby – стремление к ортогональности. Массив в некоторых отношениях подобен хэшу, а потому и операции над ними должны быть похожи. До тех пор, пока мы не вступаем в область, где эти сущности начинают отличаться друг от друга.

Мац говорит что «естественность» важнее ортогональности. Но чтобы понять, что естественно, а что нет, надо долго думать и писать программы.

Ruby стремится быть дружелюбным к программисту. Например, у многих методов есть синонимы; оба метода `size` и `length` возвращают число элементов в массиве. Два разных написания слова – `indexes` и `indices` – относятся к имени одного и того же метода. Некоторые называют это досадным недоразумением, но я склонен считать такую избыточность хорошим дизайном.

Ruby стремится к последовательности и единообразию. В этом нет ничего мистического, во всех жизненных ситуациях мы жаждем регулярности и размеренности. Сложнее научиться понимать, когда этот принцип следует нарушить.

Например, в Ruby принято добавлять вопросительный знак (?) в конец имени метода, ведущего себя как предикат. Это хорошо и удобно, программа становится яснее, а в пространстве имен легче ориентироваться. Но менее последовательным является аналогичное употребление восклицательного знака для обозначения потенциально «деструктивных» или «опасных» методов (в том смысле, что они модифицируют внутреннее состояние вызывающего объекта). Непоследовательность состоит в том, что не все деструктивные методы помечаются таким образом. Нужно ли восстановить справедливость?

Нет, на самом деле не нужно. Некоторые методы по сути своей изменяют состояние (например, методы `replace` и `concat` класса `Array`). Одни являются «методами установки», которые допускают присваивание атрибуту класса; ясно, что не следует добавлять восклицательный знак к имени атрибута или к знаку равенства. Другие в каком-то смысле изменяют состояние объекта, например, `read`, но это происходит так часто, что нет смысла особо отмечать этот факт. Если бы имя каждого деструктивного метода заканчивалось символом `!`, то программа превратилась бы в рекламную брошюру фирмы, занимающейся многоуровневым маркетингом.

Вы замечаете действие разнонаправленных сил, тенденцию нарушать все правила? Тогда позвольте мне сформулировать второй закон Фултона: «У каждого правила есть исключения, кроме второго закона Фултона». (Доля шутки тут есть, но небольшая.)

В Ruby мы видим не «педантичную непротиворечивость», а строгое следование набору простых правил. Быть может, отчасти Путь Ruby состоит в том, что его подход не является закостенелым и неподвижным. Мац как-то сказал, что при проектировании языка нужно «следовать велениям своего сердца».

И еще один аспект философии Ruby: «Не бойтесь изменений во время выполнения, не бойтесь быть динамичными.» Мир динамичен, так почему язык про-

граммирования должен быть статичным? Ruby – один из самых динамичных среди существующих языков.

Я бы с некоторой долей неуверенности выделил еще один аспект: «Не будьте рабом производительности». Если производительность оказывается недопустимо низкой, проблему придется решать, но не следует с самого начала выводить ее на первый план. Предпочитайте элегантность эффективности в тех случаях, когда эффективность не слишком критична. Впрочем, когда вы пишете библиотеку, которая будет использоваться непредвиденными способами, о производительности следует задуматься с самого начала.

Когда я смотрю на язык Ruby, то вижу равновесие между разными проектными целями, вижу сложное взаимодействие, напоминающее о задаче n тел в физике. Я могу представить себе, что он моделировался как мобил Александра Кальдера. Быть может, больше всего завораживает само взаимодействие, гармония, лежащая в основе философии Ruby, а не отдельные составные части. Программисты знают, что их ремесло – не просто сплав науки и технологии, но еще и искусство. Мне неловко говорить, что в компьютерных дисциплинах есть какой-то духовный аспект, но строго между нами – он, безусловно, присутствует. (Если вы не читали книгу Роберта Пирсига «Искусство ухода за мотоциклом» (Robert Pirsig *Zen and the Art of Motorcycle Maintenance*), советую прочитать.)

Источником Ruby стала человеческая потребность создавать полезные и красивые вещи. Программы, написанные на Ruby, должны проистекать из того же боговдохновенного источника. Это, на мой взгляд, и является квинтэссенцией Пути Ruby.