



Содержание

Предисловие	22
Благодарности	26
Об авторах	29
Введение	30
Глава 1. Обзор Ruby	39
1.1. Введение в объектно-ориентированное программирование.....	40
1.1.1. Что такое объект	40
1.1.2. Наследование.....	41
1.1.3. Полиморфизм	43
1.1.4. Еще немного терминов	44
1.2. Базовый синтаксис и семантика Ruby	45
1.2.1. Ключевые слова и идентификаторы	46
1.2.2. Комментарии и встроенная документация.....	47
1.2.3. Константы, переменные и типы	47
1.2.4. Операторы и приоритеты.....	49
1.2.5. Пример программы	50
1.2.6. Циклы и ветвление.....	52
1.2.7. Исключения	57
1.3. ООП в Ruby	59
1.3.1. Объекты	59
1.3.2. Встроенные классы	60
1.3.3. Модули и классы-примеси.....	61
1.3.4. Создание классов.....	62
1.3.5. Методы и атрибуты.....	66
1.4. Динамические аспекты Ruby.....	68
1.4.1. Кодирование во время выполнения	68
1.4.2. Отражение.....	70
1.4.3. Отсутствующие методы	71
1.4.4. Сборка мусора.....	71
1.5. Потренируйте свою интуицию: что следует запомнить.....	72

1.5.1. Синтаксис.....	72
1.5.2. Отличия от других языков	74
1.5.3. Предложение case в Ruby	77
1.5.4. Рубизмы и идиомы	79
1.5.5. Ориентация на выражения и прочие вопросы	84
1.6. Жаргон Ruby	86
1.7. Заключение	89

Глава 2. Строки 90

2.1. Представление обычных строк	90
2.2. Альтернативная нотация для представления строк	91
2.3. Встроенные документы	92
2.4. Получение длины строки	93
2.5. Построчная обработка.....	93
2.6. Побайтовая обработка.....	94
2.7. Специализированное сравнение строк.....	94
2.8. Разбиение строки на лексемы	96
2.9. Форматирование строк	97
2.10. Строки в качестве объектов ввода-вывода.....	98
2.11. Управление регистром	98
2.12. Вычленение и замена подстрок	99
2.13. Подстановка в строках.....	101
2.14. Поиск в строке	102
2.15. Преобразование символов в коды ASCII и обратно	103
2.16. Явные и неявные преобразования	103
2.17. Дописывание в конец строки	105
2.18. Удаление хвостовых символов новой строки и прочих	105
2.19. Убирание лишних пропусков.....	106
2.20. Повтор строк	106
2.21. Включение выражений в строку	107
2.22. Отложенная интерполяция.....	107
2.23. Разбор данных, разделенных запятыми	108
2.24. Преобразование строки в число (десятичное или иное)....	108
2.25. Кодирование и декодирование строк в кодировке rot13....	110
2.26. Шифрование строк	110
2.27. Сжатие строк	111
2.28. Подсчет числа символов в строке	112
2.29. Обращение строки.....	112
2.30. Удаление дубликатов	112
2.31. Удаление заданных символов	113
2.32. Печать специальных символов.....	113

Содержание

2.33. Генерирование последовательности строк	113
2.34. Вычисление 32-разрядного CRC.....	114
2.35. Вычисление SHA-256-свертки строки	114
2.36. Вычисление расстояния Левенштейна между двумя строками	115
2.37. base64-кодирование и декодирование строк.....	117
2.38. Замена символов табуляции пробелами и сворачивание пробелов в табуляторы	117
2.39. Перенос строк по словам.....	118
2.40. Заключение	119

Глава 3. Регулярные выражения

120

3.1. Синтаксис регулярных выражений.....	120
3.2. Компиляция регулярных выражений	122
3.3. Экранирование специальных символов	123
3.4. Якоря.....	123
3.5. Кванторы	124
3.6. Позитивное и негативное заглядывание вперед	126
3.7. Позитивное и негативное оглядывание назад	127
3.8. Обратные ссылки.....	128
3.9. Именованные соответствия	130
3.10. Классы символов	132
3.11. Обобщенные регулярные выражения.....	133
3.12. Сопоставление точки символу конца строки	134
3.13. Внутренние модификаторы	134
3.14. Внутренние подвыражения	135
3.14.1. Рекурсия в регулярных выражениях.....	135
3.15. Примеры регулярных выражений.....	136
3.15.1. Сопоставление с IP-адресом	136
3.15.2. Сопоставление с парой «ключ-значение».....	137
3.15.3. Сопоставление с числами, записанными римскими цифрами	138
3.15.4. Сопоставление с числовыми константами	138
3.15.5. Сопоставление с датой и временем.....	139
3.15.6. Обнаружение удвоенных слов в тексте	140
3.15.7. Поиск слов, записанных одними заглавными буквами	140
3.15.8. Сопоставление с номером версии	140
3.15.9. Еще несколько образцов.....	140
3.16. Заключение	141

Глава 4. Интернационализация в Ruby	142
4.1. Исторические сведения и терминология	143
4.2. Работа с кодировками	147
4.2.1. Нормализация	148
4.2.2. Преобразование из одной кодировки в другую	151
4.2.3. Транслитерация.....	152
4.2.4. Упорядочение строк	152
4.3. Перевод	155
4.3.1. Значения по умолчанию.....	157
4.3.2. Пространства имен.....	158
4.3.3. Интерполяция.....	158
4.3.4. Формы множественного числа	159
4.4. Локализованное форматирование	160
4.4.1. Дата и время	161
4.4.2. Числа.....	161
4.4.3. Денежные величины	162
4.5. Заключение	162
Глава 5. Численные методы	163
5.1. Представление чисел в языке Ruby.....	163
5.2. Основные операции над числами	164
5.3. Округление чисел с плавающей точкой	165
5.4. Сравнение чисел с плавающей точкой	167
5.5. Форматирование чисел для вывода.....	168
5.6. Вставка разделителей при форматировании чисел	169
5.7. Работа с очень большими числами	169
5.8. Использование класса BigDecimal	169
5.9. Работа с рациональными числами	171
5.10. Перемножение матриц	172
5.11. Комплексные числа	176
5.12. Библиотека mathn.....	176
5.13. Разложение на простые множители, вычисление НОД и НОК.....	177
5.14. Простые числа	178
5.15. Явные и неявные преобразования чисел.....	179
5.16. Приведение числовых значений.....	180
5.17. Поразрядные операции над числами	181
5.18. Преобразование системы счисления.....	182
5.19. Извлечение кубических корней, корней четвертой степени и т.д.	183

Содержание

5.20. Определение порядка байтов	183
5.21. Численное вычисление определенного интеграла	184
5.22. Тригонометрия в градусах, радианах и градах	185
5.23. Вычисление логарифмов по произвольному основанию ...	186
5.24. Вычисление среднего, медианы и моды набора данных ...	187
5.25. Дисперсия и стандартное отклонение	188
5.26. Вычисление коэффициента корреляции	189
5.27. Генерирование случайных чисел	190
5.28. Кэширование функций с помощью метода memoize	191
5.29. Заключение	192
Глава 6. Символы и диапазоны	193
6.1. Символы.....	193
6.1.1. Символы как перечисления	195
6.1.2. Символы как метазначения.....	195
6.1.3. Символы, переменные и методы	196
6.1.4. Преобразование строки в символ и обратно.....	196
6.2. Диапазоны.....	197
6.2.1. Открытые и замкнутые диапазоны	198
6.2.2. Нахождение границ диапазона	198
6.2.3. Обход диапазона	198
6.2.4. Проверка принадлежности диапазону	199
6.2.5. Преобразование в массив	200
6.2.6. Обратные диапазоны.....	200
6.2.7. Оператор переключения.....	200
6.2.8. Нестандартные диапазоны	203
6.3. Заключение	205
Глава 7. Дата и время	207
7.1. Определение текущего момента времени	208
7.2. Работа с конкретными датами (после точки отсчета)	208
7.3. Определение дня недели.....	209
7.4. Определение даты Пасхи.....	209
7.5. Вычисление n-ого дня недели в месяце	210
7.6. Преобразование из секунд в более крупные единицы.....	211
7.7. Вычисление промежутка времени, прошедшего от точки отсчета	212
7.8. Високосные секунды	212
7.9. Определение порядкового номера дня в году.....	213
7.10. Контроль даты и времени	213
7.11. Определение недели в году	214

7.12. Проверка года на високосность	214
7.13. Определение часового пояса.....	215
7.14. Манипулирование временем без даты.....	215
7.15. Сравнение моментов времени.....	216
7.16. Прибавление интервала к моменту времени	216
7.17. Вычисление разности между двумя моментами времени.....	217
7.18. Работа с конкретными датами (до точки отсчета).....	217
7.19. Взаимные преобразования объектов Date, Time и DateTime	217
7.20. Извлечение даты и времени из строки	218
7.21. Форматирование и печать даты и времени	219
7.22. Преобразование часовых поясов	219
7.23. Определение числа дней в месяце	220
7.24. Разбиение месяца на недели	220
7.25. Заключение	221

Глава 8. Массивы, хэши и другие перечисляемые структуры223

8.1. Массивы.....	223
8.1.1. Создание и инициализация массива.....	224
8.1.2. Доступ к элементам массива и присваивание им значений.....	224
8.1.3. Определение размера массива	226
8.1.4. Сравнение массивов	226
8.1.5. Сортировка массива.....	228
8.1.6. Выборка из массива по заданному критерию	230
8.1.7. Специализированные функции индексирования.....	231
8.1.8. Реализация разреженной матрицы.....	233
8.1.9. Массивы как математические множества	233
8.1.10. Рандомизация массива	237
8.1.11. Многомерные массивы.....	237
8.1.12. Нахождение элементов, принадлежащих одному массиву и не принадлежащих другому	238
8.1.13. Преобразование или отображение массивов.....	238
8.1.14. Удаление элементов, равных nil, из массива	239
8.1.15. Удаление заданных элементов из массива	239
8.1.16. Конкатенирование массивов и добавление в конец массива	240
8.1.17. Использование массива в качестве стека или очереди	241

8.1.18. Обход массива	241
8.1.19. Преобразование массива в строку с разделителями	242
8.1.20. Обращение массива	242
8.1.21. Удаление дубликатов из массива	242
8.1.22. Чередувание массивов	243
8.1.23. Вычисление частоты различных значений в массиве	243
8.1.24. Инвертирование массива для получения хэша	243
8.1.25. Синхронная сортировка нескольких массивов	244
8.1.26. Задание значения по умолчанию для новых элементов массива	245
8.2. Хэши	245
8.2.1. Создание нового хэша	246
8.2.2. Задание значения по умолчанию для хэша	247
8.2.3. Доступ к парам ключ-значение и добавление новых пар	247
8.2.4. Удаление пар ключ-значение	248
8.2.5. Обход хэша	249
8.2.6. Инвертирование хэша	249
8.2.7. Поиск ключей и значений в хэше	250
8.2.8. Копирование хэша в массив	250
8.2.9. Выборка пар ключ-значение по заданному критерию	250
8.2.10. Сортировка хэша	251
8.2.11. Объединение двух хэшей	251
8.2.12. Создание хэша из массива	252
8.2.13. Вычисление разности и пересечения хэшей	252
8.2.14. Хэш как разреженная матрица	252
8.2.15. Реализация хэша с повторяющимися ключами	253
8.2.16. Другие операции с хэшем	256
8.3. Перечисляемые структуры в общем	256
8.3.1. Метод inject	257
8.3.2. Кванторы	258
8.3.3. Метод partition	258
8.3.4. Обход с группировкой	259
8.3.5. Преобразование в массив или множество	260
8.3.6. Перечислители	260
8.4. Дополнительные сведения о перечисляемых объектах	262
8.4.1. Поиск и выборка	262
8.4.2. Подсчет и сравнение	263

8.4.3. Итерирование	263
8.4.4. Извлечение и преобразование	264
8.4.5. Ленивые перечислители	265
8.5. Заключение	265

Глава 9. Более сложные структуры данных266

9.1. Множества	266
9.1.1. Простые операции над множествами	267
9.1.2. Более сложные операции над множествами	268
9.2. Стеки и очереди	269
9.2.1. Более строгая реализация стека	271
9.2.2. Обнаружение несбалансированных скобок.....	271
9.2.3. Стек и рекурсия	272
9.2.4. Более строгая реализация очереди	274
9.3. Деревья.....	275
9.3.1. Реализация двоичного дерева.....	275
9.3.2. Сортировка с помощью двоичного дерева.....	277
9.3.3. Использование двоичного дерева как справочной таблицы	279
9.3.4. Преобразование дерева в строку или массив	279
9.4. Графы.....	280
9.4.1. Реализация графа в виде матрицы смежности	281
9.4.2. Является ли граф связным?	283
9.4.3. Есть ли в графе эйлеров цикл?.....	284
9.4.4. Есть ли в графе эйлеров путь?	285
9.4.5. Инструменты для работы с графами в Ruby	286
9.5. Заключение	286

Глава 10. Ввод-вывод и хранение данных287

10.1. Файлы и каталоги	288
10.1.1. Открытие и закрытие файлов.....	288
10.1.2. Обновление файла	289
10.1.3. Дописывание в конец файла	290
10.1.4. Прямой доступ к файлу	290
10.1.5. Работа с двоичными файлами.....	291
10.1.6. Блокировка файлов	292
10.1.7. Простой ввод-вывод.....	293
10.1.8. Буферизованный и небуферизованный ввод-вывод.....	294
10.1.9. Манипулирование правами владения и разрешениями на доступ к файлу.....	295
10.1.10. Получение и установка временных меток.....	296

10.1.11. Проверка существования и получение размера файла	297
10.1.12. Опрос специальных свойств файла	298
10.1.13. Каналы	300
10.1.14. Специальные операции ввода-вывода	301
10.1.15. Неблокирующий ввод-вывод	302
10.1.16. Применение метода readpartial	302
10.1.17. Манипулирование путевыми именами	303
10.1.18. Класс Pathname	304
10.1.19. Манипулирование файлами на уровне команд	305
10.1.20. Ввод символов с клавиатуры	306
10.1.21. Чтение всего файла в память	306
10.1.22. Построчное чтение из файла	307
10.1.23. Побайтное и посимвольное чтение из файла	307
10.1.24. Работа со строкой как с файлом	308
10.1.25. Копирование потока	308
10.1.26. Работа с кодировками	308
10.1.27. Чтение данных, встроенных в текст программы	309
10.1.28. Чтение исходного текста программы	309
10.1.29. Работа с временными файлами	309
10.1.30. Получение и изменение текущего каталога	310
10.1.31. Изменение текущего корня	310
10.1.32. Обход каталога	311
10.1.33. Получение содержимого каталога	311
10.1.34. Создание цепочки каталогов	311
10.1.35. Рекурсивное удаление каталога	311
10.1.36. Поиск файлов и каталогов	311
10.2. Доступ к данным более высокого уровня	312
10.2.1. Простой маршалинг	313
10.2.2. «Глубокое копирование» с помощью метода Marshal	313
10.2.3. Более сложный маршалинг	314
10.2.4. Маршалинг в формате YAML	315
10.2.5. Сохранение данных с помощью библиотеки JSON	316
10.2.6. Работа с данными в формате CSV	317
10.2.7. SQLite3 как SQL-хранилище данных	319
10.3. Подключение к внешним базам данных	320
10.3.1. Подключение к базе данных MySQL	320
10.3.2. Подключение к базе данных PostgreSQL	322
10.3.3. Объектно-реляционные отображения (ORM)	323



10.3.4. Подключение к хранилищу данных Redis	324
10.4. Заключение	326

Глава 11. ООП и динамические механизмы в Ruby327

11.1. Рутинные объектно-ориентированные задачи	328
11.1.1. Применение нескольких конструкторов	328
11.1.2. Создание атрибутов экземпляра	329
11.1.3. Более сложные конструкторы	331
11.1.4. Создание атрибутов и методов уровня класса	333
11.1.5. Наследование суперклассу.....	336
11.1.6. Опрос класса объекта.....	338
11.1.7. Проверка объектов на равенство	340
11.1.8. Управление доступом к методам	341
11.1.9. Копирование объектов.....	343
11.1.10. Метод initialize_copy	344
11.1.11. Метод allocate	345
11.1.12. Модули	346
11.1.13. Трансформация или преобразование объектов	349
11.1.14. Классы, содержащие только данные (Struct).....	351
11.1.15. Замораживание объектов	352
11.1.16. Использование метода tap в цепочке методов	353
11.2. Более сложные механизмы.....	354
11.2.1. Посылка объекту явного сообщения	354
11.2.2. Специализация отдельного объекта	356
11.2.3. Вложенные классы и модули.....	359
11.2.4. Создание параметрических классов	360
11.2.5. Хранение кода в виде объектов Proc	362
11.2.6. Хранение кода в виде объектов Method	364
11.2.7. Использование символов в качестве блоков	364
11.2.8. Как работает включение модулей?.....	365
11.2.9. Опознание параметров, заданных по умолчанию.....	367
11.2.10. Делегирование или перенаправление	367
11.2.11. Автоматическое определение методов чтения и установки на уровне класса	370
11.2.12. Поддержка различных стилей программирования ...	371
11.3. Динамические механизмы	373
11.3.1. Динамическая интерпретация кода	373
11.3.2. Метод const_get.....	375
11.3.3. Получение класса по имени	375
11.3.4. Метод define_method	375
11.3.5. Получение списка определенных сущностей	378

11.3.6. Удаление определений	380
11.3.7. Ссылки на несуществующие константы	382
11.3.8. Вызовы несуществующих методов	383
11.3.9. Повышение безопасности с помощью taint	384
11.3.10. Определение чистильщиков для объектов	386
11.4. Интроспекция программы	387
11.4.1. Обход пространства объектов	387
11.4.2. Просмотр стека вызовов.....	388
11.4.3. Отслеживание изменений в определении класса или объекта.....	388
11.4.4. Мониторинг выполнения программы	391
11.5. Заключение	393
Глава 12. Графические интерфейсы для Ruby	394
12.1. Shoes 4	395
12.1.1. Начало работы с Shoes	395
12.1.2. Интерактивная кнопка	396
12.1.3. Текст и поле ввода	396
12.1.4. Компоновка	397
12.1.5. Картинки и фигуры	399
12.1.6. События.....	400
12.1.7. Прочие замечания	400
12.2. Ruby/Tk.....	401
12.2.1. Обзор	401
12.2.2. Простое оконное приложение.....	402
12.2.3. Кнопки	403
12.2.4. Текстовые поля	407
12.2.5. Прочие виджеты	411
12.2.6. Дополнительные замечания	413
12.3. Ruby/GTK3.....	414
12.3.1. Обзор	414
12.3.2. Простое оконное приложение.....	415
12.3.3. Кнопки	416
12.3.4. Текстовые поля	417
12.3.5. Прочие виджеты	420
12.3.6. Дополнительные замечания	424
12.4. QtRuby	425
12.4.1. Обзор	425
12.4.2. Простое оконное приложение.....	426
12.4.3. Кнопки	426
12.4.4. Текстовые поля	428



12.4.5. Прочие виджеты	430
12.4.6. Дополнительные замечания	434
12.5. Swing	435
12.6. Другие библиотеки для создания графических интерфейсов	437
12.6.1. UNIX и X11	437
12.6.2. FXRuby (FOX).....	437
12.6.3. RubyMotion для iOS и Mac OS X	437
12.6.4. Windows Win32 API.....	438
12.7. Заключение	438

Глава 13. Потоки и параллелизм439

13.1. Создание потоков и манипулирование ими.....	440
13.1.1. Создание потоков.....	441
13.1.2. Доступ к поточно-локальным переменным	442
13.1.3. Опрос и изменение состояния потока.....	443
13.1.4. Назначение рандеву (и получение возвращенного значения).....	446
13.1.5. Обработка исключений	447
13.1.6. Группы потоков	449
13.2. Синхронизация потоков	450
13.2.1. Простая синхронизация.....	451
13.2.2. Синхронизация доступа с помощью мьютекса	452
13.2.3. Встроенные классы очередей.....	454
13.2.4. Условные переменные	456
13.2.5. Другие способы синхронизации	457
13.2.6. Таймаут при выполнении операций.....	460
13.2.7. Ожидание события	462
13.2.8. Параллельный поиск в коллекции	463
13.2.9. Параллельное рекурсивное удаление	464
13.3. Волокна и кооперативная многозадачность.....	465
13.4. Заключение	467

Глава 14. Скрипты и системное администрирование468

14.1. Запуск внешних программ	468
14.1.1. Методы system и exec	469
14.1.2. Запоминание вывода программы	469
14.1.3. Манипулирование процессами.....	470
14.1.4. Стандартный ввод и вывод	473
14.2. Флаги и аргументы в командной строке	474
14.2.1. Константа ARGV	474

14.2.2. Константа ARGF	474
14.2.3. Разбор флагов в командной строке	475
14.3. Библиотека Shell.....	476
14.3.1. Использование библиотеки Shell для перенаправления ввода-вывода	477
14.3.2. Дополнительные замечания по поводу библиотеки Shell	478
14.4. Переменные окружения	479
14.4.1. Чтение и установка переменных окружения	479
14.4.2. Хранение переменных окружения в виде массива или хэша	480
14.5. Работа с файлами, каталогами и деревьями	481
14.5.1. Несколько слов о текстовых фильтрах.....	481
14.5.2. Копирование дерева каталогов (с символическими ссылками)	481
14.5.3. Удаление файлов по времени модификации и другим критериям	483
14.5.4. Вычисление свободного места на диске	484
14.6. Различные скрипты.....	484
14.6.1. Распространение программ на Ruby.....	484
14.6.2. Подача входных данных Ruby по конвейеру.....	485
14.6.3. Работает ли Ruby в интерактивном режиме?.....	486
14.6.4. Определение текущей платформы или операционной системы	486
14.6.5. Модуль Etc.....	487
14.7. Заключение	487
Глава 15. Ruby и форматы данных	489
15.1. Разбор JSON	490
15.1.1. Обход JSON-данных	490
15.1.2. Типы данных, не представимые в JSON.....	491
15.1.3. Другие библиотеки для работы с JSON	492
15.2. Разбор XML (и HTML)	492
15.2.1. Разбор документа.....	493
15.2.2. Поточковый разбор	495
15.3. RSS и Atom.....	497
15.3.1. Разбор новостной ленты	497
15.3.2. Создание новостных лент	499
15.4. Обработка изображений с помощью RMagick	499
15.4.1. Типичные графические задачи.....	500
15.4.2. Специальные эффекты и преобразования	503



15.4.3. API рисования	505
15.5. Создание документов в формате PDF с помощью библиотеки Prawn	508
15.5.1. Основные концепции и приемы	509
15.5.2. Пример документа	509
15.6. Заключение	513

Глава 16. Тестирование и отладка

514

16.1. Тестирование с помощью RSpec	515
16.2. Тестирование с помощью Minitest	517
16.3. Тестирование с помощью Cucumber	522
16.4. Работа с отладчиком byebug	524
16.5. Отладка с помощью pry	527
16.6. Измерение производительности	528
16.7. Объекты форматированной печати	532
16.8. О том, что осталось за кадром	534
16.9. Заключение	534

Глава 17. Создание пакетов и распространение программ

535

17.1. Библиотеки и система Rubygems	535
17.1.1. Работа с RubyGems	536
17.1.2. Создание gem-пакетов	536
17.2. Управление зависимостями с помощью Bundler	537
17.2.1. Семантическое версионирование	539
17.2.2. Загрузка зависимостей из Git	539
17.2.3. Создание gem-пакетов с помощью Bundler	540
17.2.4. Частные gem-пакеты	540
17.3. Программа RDoc	541
17.3.1. Простая разметка	543
17.3.2. Создание улучшенной документации с помощью Yard	545
17.4. Заключение	546

Глава 18. Сетевое программирование

547

18.1. Сетевые серверы	549
18.1.1. Простой сервер: время дня	549
18.1.2. Реализация многопоточного сервера	550
18.1.3. Пример: сервер для игры в шахматы по сети	551
18.2. Сетевые клиенты	558
18.2.1. Получение истинно случайных чисел из веб	558

18.2.2. Запрос к официальному серверу времени	561
18.2.3. Взаимодействие с POP-сервером	562
18.2.4. Отправка почты по протоколу SMTP	563
18.2.5. Взаимодействие с IMAP-сервером	566
18.2.6. Кодирование и декодирование вложений	568
18.2.7. Пример: шлюз между почтой и конференциями	570
18.2.8. Получение веб-страницы с известным URL	575
18.2.9. Библиотека Open-URI	575
18.3. Заключение	576

Глава 19. Ruby и веб-приложения577

19.1. HTTP-серверы	577
19.1.1. Простой HTTP-сервер	577
19.1.2. Rack и веб-серверы	579
19.2. Каркасы приложений	581
19.2.1. Маршрутизация в Sinatra	582
19.2.2. Маршрутизация в Rails	583
19.2.3. Параметры в Sinatra	585
19.2.4. Параметры в Rails	586
19.3. Хранение данных	586
19.3.1. Базы данных	587
19.3.2. Хранилища данных	589
19.4. Генерация HTML	589
19.4.1. ERB	590
19.4.2. Haml	592
19.4.3. Другие шаблонные системы	592
19.5. Конвейер активов	593
19.5.1. CSS и Sass	593
19.5.2. JavaScript и CoffeeScript	595
19.6. Предоставление веб-служб по протоколу HTTP	597
19.6.1. Использование JSON в API	597
19.6.2. REST API и его вариации	598
19.7. Генерация статических сайтов	599
19.7.1. Middleman	599
19.7.2. Другие генераторы статических сайтов	600
19.8. Заключение	601

Глава 20. Распределенный Ruby602

20.1. Обзор: библиотека drb	602
20.2. Пример: эмуляция биржевой ленты	605
20.3. Rinda: пространство кортежей в Ruby	608

20.4. Обнаружение служб в распределенном Ruby	611
20.5. Заключение	613
Глава 21. Инструменты разработки для Ruby	614
21.1. Программа Rake	614
21.2. Оболочка irb.....	618
21.3. Основы pry.....	621
21.4. Утилита ri	623
21.5. Поддержка со стороны редакторов.....	623
21.5.1. Vim.....	624
21.5.2. Emacs	624
21.6. Менеджеры версий Ruby	625
21.6.1. Работа с rvm	625
21.6.2. Работа с rbenv.....	625
21.6.3. Работа с chruby.....	626
21.7. Заключение	627
Глава 22. Сообщество пользователей Ruby	628
22.1. Ресурсы в веб	628
22.2. Новостные группы и списки рассылки.....	628
22.3. Извещения об ошибках и предложения новых функций ...	629
22.4. Каналы IRC	629
22.5. Конференции по Ruby.....	630
22.6. Локальные группы пользователей Ruby	631
22.7. Заключение	631
Предметный указатель	632



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 ДНН 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 избытком 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 Арко¹. 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 Хотите 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 Тао, трудно. Я чувствую, но не могу подобрать 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 Тао 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.

*Юкихира «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, «Тао Те Чинг»

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 вобилии.

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 почти 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 разнообразия 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 пре-словутых “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 особенностью 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 Райт¹ (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 APЛ 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. Масив 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 масиве. 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.