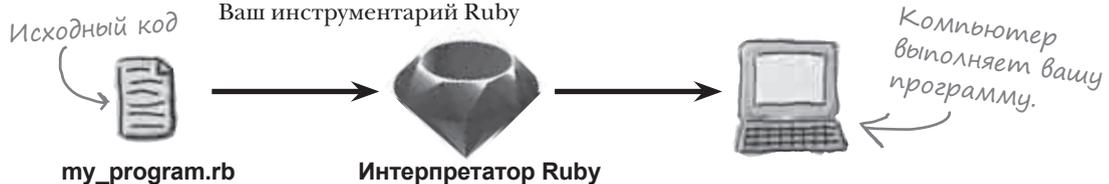


1 Как сделать больше меньшими усилиями

Программируйте так, как вам удобно

Вас интересует, почему вокруг столько говорят о языке Ruby и подойдет ли он вам? Ответьте на простой вопрос: **вам нравится работать эффективно?** Кажется ли вам, что с многочисленными компиляторами, библиотеками, файлами классов и комбинациями клавиш других языков вы действительно становитесь ближе к **готовому продукту, восхищению коллег и толпе счастливых заказчиков?** Вы хотите, чтобы язык программирования **занимался техническими подробностями** за вас? Если вам хочется перестать писать рутинный код и *работать над задачей*, то язык Ruby — для вас. Ruby позволит вам **сделать больше с меньшим объемом кода.**

Философия Ruby	32
Интерактивное использование Ruby	35
Ваши первые выражения Ruby	36
Математические операторы и сравнения	36
Строки	36
Переменные	37
Вызов метода для объекта	38
Ввод, сохранение и вывод	42
Запуск сценариев	43
Комментарии	44
Аргументы методов	45
Интерполяция строк	46
Анализ объектов методами «inspect» и «p»	48
Служебные последовательности в строках	49
Вызов «chomp» для объекта строки	50
Генерирование случайного числа	52
Преобразование числа в строку	53
Преобразование строк в числа	55
Условные команды	56
«unless» как противоположность «if»	59
Циклы	60
Попробуем запустить игру!	63
Ваш инструментарий Ruby	64



Методы и классы

2

Наводим порядок

В вашей работе кое-чего не хватало. Да, вы вызывали методы и создавали объекты, как настоящий профессионал. Но при этом вы могли вызывать только те методы и создавать только те виды объектов, которые были определены за вас в Ruby. Теперь ваша очередь. В этой главе вы научитесь создавать *свои* методы, а также свои **классы** — своего рода «шаблоны» для создания новых объектов. *Вы сами решаете*, как будут выглядеть объекты, созданные на базе вашего класса. **Переменные экземпляра** определяют, какая информация *хранится* в ваших объектах, а **методы экземпляра** определяют, что эти объекты *делают*. А самое главное — вы узнаете, как определение собственных классов *упрощает чтение и сопровождение* вашего кода.

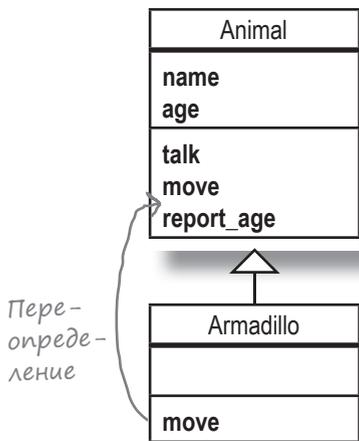


Определение методов	66
Вызов методов	67
Имена методов	68
Параметры	68
Возвращаемые значения	72
Раннее возвращение из метода	73
Беспорядок в методах	74
Слишком много аргументов	75
Слишком много команд «if»	75
Проектирование класса	76
Чем класс отличается от объекта	77
Первый класс	78
Создание новых экземпляров (объектов)	78
Разбиение больших методов на классы	79
Создание экземпляров новых классов животных	80
Диаграмма классов с методами экземпляра	81
Локальные переменные существуют до завершения метода	85
Инкапсуляция	88
Методы доступа	89
Использование методов доступа	91
Методы чтения и записи атрибутов	92
Ошибки и «аварийная остановка»	100
Использование «raise» в методах записи атрибутов	101
Ваш инструментарий Ruby	103

3 Наследование С родительской помощью

Столько повторений! Новые классы, представляющие разные типы машин или животных, удобны — это правда. Но ведь вам придется *копировать методы экземпляра из класса в класс*. И эти копии будут вести самостоятельную жизнь — одни будут работать нормально, в других могут появиться ошибки. Разве классы создавались не для того, чтобы *упростить* сопровождение кода?

В этой главе вы научитесь применять **наследование**, чтобы ваши классы могли использовать *одни и те же* методы. Меньше дубликатов — меньше проблем с сопровождением!



Копировать, вставлять... Столько проблем!	106
Код Майка для приложения виртуального тест-драйва	107
На помощь приходит наследование!	108
Определение суперкласса (в общем-то ничего особенного!)	110
Определение subclasses (совсем просто)	111
Добавление методов в subclasses	112
Subclasses содержат как новые, так и унаследованные методы	113
Переменные экземпляра принадлежат объекту, а не классу!	114
Переопределение методов	116
Включение наследования в иерархию классов животных	121
Проектирование иерархий классов животных	122
Код класса Animal и его subclasses	123
Переопределение метода в subclasses Animal	124
Как добраться до переопределяемого метода?	125
Ключевое слово «super»	126
Subclass обретает суперсилу	128
Трудности с выводом Dog	131
Класс Object	132
Почему все классы наследуют от Object	133
Переопределение унаследованного метода	134
Ваш инструментарий Ruby	135

Инициализация экземпляров

4

Хороший старт — половина дела

Пока что ваш класс напоминает бомбу с часовым механизмом. Каждый экземпляр, созданный вами, начинается «с пустого места». Если вы вызовете методы экземпляра до того, как в нем появятся данные, то может произойти ошибка, приводящая к аварийному завершению программы. В этой главе мы представим пару способов создания объектов, которые можно сразу безопасно использовать. Начнем с метода `initialize`, который позволяет передать набор аргументов для заполнения данных объекта *на момент его создания*. Затем мы покажем, как написать **методы класса**, которые позволяют **еще** проще создавать и инициализировать объекты.



Зарплата в Chargetmore	138
Класс Employee	139
Создание новых экземпляров Employee	140
Деление с использованием класса Ruby Fixnum	142
Деление с классом Ruby Float	143
Исправление ошибки округления в Employee	144
Форматирование чисел для вывода	145
Форматные последовательности	146
Применение метода «format» для исправления вывода	149
Если атрибуты объекта не заданы	150
«nil» означает «ничто»	151
«/» — это метод	152
Метод «initialize»	153
Безопасность использования Employee и «initialize»	154
Аргументы «initialize»	155
Необязательные параметры и «initialize»	156
«initialize» и проверка данных	160
«self» и вызов других методов для того же экземпляра	161
Когда ключевое слово «self» не обязательно	165
Реализация почасовой оплаты на основе наследования	167
Восстановление методов «initialize»	170
Наследование и метод «initialize»	171
«super» и «initialize»	172
Методы класса	179
Полный исходный код класса	182
Ваш инструментарий Ruby	184

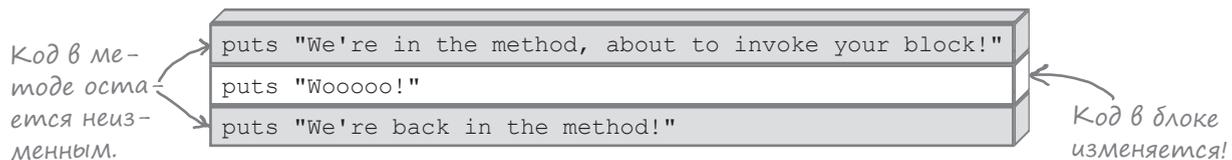
5

Массивы и блоки

Лучше, чем цикл

Очень многие задачи из области программирования связаны с обработкой списков. Списки адресов. Списки телефонных номеров. Списки продуктов. Мац, создатель языка Ruby, знал об этом. Поэтому он *основательно* потрудились над тем, чтобы работать со списками в Ruby было *действительно просто*. Сначала он поработал над тем, чтобы **массивы**, используемые для работы со списками в Ruby, обладали множеством *мощных методов* для выполнения практически любых операций. Затем он осознал, что написание кода для *перебора всех элементов списка* для выполнения некоторой операции с каждым элементом — утомительная рутинная работа, которую программистам приходится выполнять *очень часто*. Поэтому он добавил в язык **блоки**, благодаря которым код перебора стал лишним. Что же это такое — блок? Сейчас узнаете...

Массивы	186
Работа с массивами	187
Перебор элементов массива	191
Устранение дубликатов — НЕПРАВИЛЬНЫЙ способ	195
Блоки	197
Определение метода, получающего блок	198
Ваш первый блок	199
Передача управления между методом и блоком	200
Вызов одного метода с разными блоками	201
Множественный вызов блока	202
Параметры блоков	203
Ключевое слово «yield»	204
Форматы блоков	205
Метод «each»	209
Устранение повторов из кода при помощи «each» и блоков	211
Блоки и область видимости переменных	214
Использование «each» с методом «refund»	216
Использование «each» в последнем методе	217
Полный код методов	218
Ваш инструментарий Ruby	222

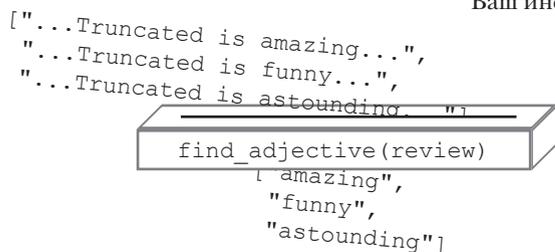
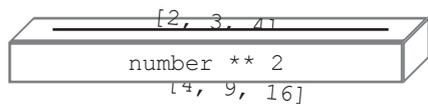


Возвращаемые значения блоков

6 Как будем обрабатывать?

Вы видели лишь малую часть возможностей блоков. До настоящего момента *методы* передавали данные блоку и ожидали, что блок сделает все необходимое. Однако *блок* также может возвращать данные *методу*. Эта возможность позволяет методу получать *команды* от блока, позволяя ему выполнять более содержательную работу. Методы, рассмотренные в этой главе, получают *большие и сложные* коллекции и используют **возвращаемые значения блоков** для сокращения их размера.

Поиск в больших коллекциях слов	224
Открытие файла	226
Безопасное закрытие файла	226
Безопасное закрытие файла в блоке	227
Не забывайте про область видимости!	228
Поиск элементов в блоке	231
Долгий способ поиска с использованием «each»	232
А теперь короткий способ...	233
Блоки тоже возвращают значение	234
Как метод использует возвращаемое значение блока	239
Все вместе	240
Возвращаемые значения блоков: присмотримся повнимательнее	241
Исключение лишних элементов с использованием блока	242
Возвращаемые значения для «reject»	243
Преобразование строки в массив слов	244
Определение индекса элемента массива	245
Создание массива на базе другого массива (сложный способ)	246
Создание массива на базе другого массива с использованием «map»	247
Дополнительная логика в теле блока «map»	249
Работа закончена	250
Ваш инструментарий Ruby	253



7

Хеши

Пометка данных

Хранить данные в одной большой куче удобно... До тех пор, пока вам не потребуется в них что-нибудь найти. Вы уже видели, как создать коллекцию объектов с использованием *массива*. Вы уже видели, как обработать *каждый элемент* массива и как *найти* нужные элементы. В обоих случаях мы начинаем от начала массива и *проверяем каждый отдельный объект*. Вы уже видели методы, получающие большие коллекции в параметрах. Вам уже известно, какие проблемы это создаст: вызовы методов требуют передачи большой, *запутанной коллекции аргументов*, для которой вам нужно помнить точный порядок. А не существует ли коллекции, у которой *все данные* уже снабжены *метками*? Тогда вы могли бы *быстро найти* нужные элементы! В этой главе рассматриваются **хеши** Ruby, предназначенные именно для этого.



Массив



Хеш

Подсчет голосов	256
Массив массивов — не идеальное решение	257
Хеши	258
Хеши — тоже объекты	260
Хеши возвращают «nil» по умолчанию	263
Значение nil (и только nil) интерпретируется как ложное	264
Возвращение по умолчанию другого значения вместо «nil»	266
Нормализация ключей хеша	268
Хеши и «each»	270
Путаница с аргументами	272
Передача хешей в параметрах методов	273
Параметр-хеш в классе Candidate	274
Фигурные скобки не нужны!	275
И стрелки не нужны!	275
Сам хеш тоже может быть необязательным	276
Опасные опечатки в аргументах-хешах	278
Ключевые слова в аргументах	279
Использование ключевых слов в аргументах в классе Candidate	280
Обязательные ключевые слова в аргументах	281
Ваш инструментарий Ruby	285



Ссылки

Путаница с сообщениями

Вы когда-нибудь отправляли сообщения не тому контакту?

Наверное, вам пришлось изрядно потрудиться, чтобы разобраться с возникшей путаницей. Объекты *Ruby* очень похожи на контакты в адресной книге, и вызов методов для них напоминает отправку сообщений. Если содержимое вашей адресной книги будет перепутано, вы рискуете отправить сообщение не тому объекту. В этой главе вы научитесь распознавать такие ситуации и узнаете, как снова наладить работу ваших программ.

Загадочные ошибки	288
Куча	289
Ссылки	290
Путаница со ссылками	291
Наложение имен	292
Исправление ошибки в программе астронома	294
Быстрая идентификация объектов методом «inspect»	296
Проблемы с объектом по умолчанию для хеша	297
Объект по умолчанию для хеша: близкое знакомство	300
Возвращаемся к хешу с планетами и спутниками	301
Чего мы хотим от объектов по умолчанию для хеша	302
Блоки по умолчанию для хешей	303
Блоки по умолчанию для хеша: присваивание	304
Блоки по умолчанию для хеша: возвращаемое значение блока	305
Блоки по умолчанию для хешей: сокращенная запись	306
Хеш астронома: окончательная версия кода	309
Безопасное использование объектов по умолчанию	310
Объекты по умолчанию для хешей: простое правило	313
Ваш инструментарий Ruby	314



Западная улица в реальности



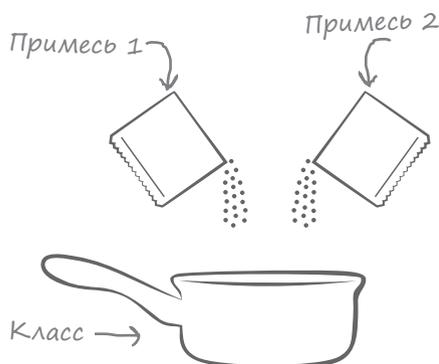
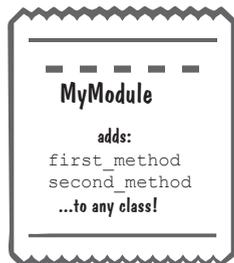
Западная улица по адресной книге Артура

9 Примеси

Аккуратный выбор

У наследования есть свои ограничения. Наследовать методы можно только от одного класса. Но что, если вам понадобилось использовать *некоторое поведение* в совершенно разных классах? Представьте методы для запуска цикла зарядки и получения информации о текущем уровне заряда — такие методы могут использоваться телефонами, электродрелями и электромобилями. И вы готовы создать *один* суперкласс для всех *этих* устройств? (Не пытайтесь, ничем хорошим это не кончится.) Или методы запуска и остановки двигателя. Конечно, эти методы могут пригодиться дрели и автомобилю, но не телефону!

В этой главе рассматриваются **модули и примеси** — мощный механизм *группировки методов* и их последующего включения *только в те классы, для которых они актуальны*.



Мультимедийное приложение	316
Приложение с наследованием	317
Один из этих классов не похож на другие	318
Вариант 1: Определение Photo как subclasses Clip	318
Вариант 2: Копирование нужных методов в класс Photo	319
Не вариант: Множественное наследование	320
Модули как примеси	321
Примеси: как это работает	323
Создание примеси	327
Использование примеси	328
Об изменениях в методе «comments»	329
Почему не следует добавлять «initialize» в примесь	330
Примеси и переопределение методов	332
Старайтесь не использовать методы «initialize» в модулях	333
Логический оператор «или» при присваивании	335
Оператор условного присваивания	336
Полный код	339
Ваш инструментарий Ruby	340

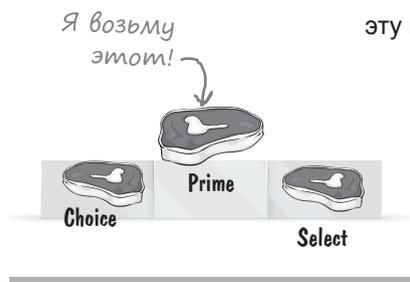
10

Comparable и enumerable

Готовые примеси

Из предыдущей главы вы уже знаете, что примеси полезны.

Но вы еще не видели их истинную мощь. В стандартную библиотеку Ruby входят две совершенно *потрясающие* примеси. Первая, Comparable, используется для сравнения объектов. Мы уже использовали такие операторы, как <, > и ==, с числами и строками, а с Comparable сможете использовать их с *вашими* классами. Вторая примесь, Enumerable, применяется при работе с коллекциями. Помните замечательные методы find_all, reject и map, которые мы использовали с массивами? Так вот, они были предоставлены Enumerable. Но это лишь малая часть возможностей Enumerable! И разумеется, вы можете включать эту примесь в свои классы. Хотите узнать, как это делается? Читайте дальше!



Встроенные примеси в Ruby	342
Знакомство с примесью Comparable	343
Выбор стейка	344
Реализация метода «больше» в классе Steak	345
Константы	346
Работа только начинается...	347
Примесь Comparable	348
Оператор <=>	349
Реализация оператора <=> в классе Steak	350
Включение Comparable в Steak	351
Как работают методы Comparable	352
Следующая примесь	355
Модуль Enumerable	356
Класс для включения Enumerable	357
Включение Enumerable в класс	358
Внутри модуля Enumerable	359
Ваш инструментарий Ruby	362

11

Документация

Читайте документацию

В книге не хватит места, чтобы рассказать все о Ruby. Есть старая поговорка: «Дайте человеку рыбу, и он будет сыт один день. Научите его ловить рыбу, и он будет сыт всю жизнь». До сих пор мы *давали вам рыбу*: показали, как использовать некоторые классы и модули Ruby. Однако существуют десятки других классов и модулей, которые могут пригодиться в вашей работе, но нам не хватит места, чтобы описать их в книге. Пришло время *научить вас ловить рыбу*. Существует превосходная бесплатная **документация** по всем классам, модулям и методам Ruby. Вам просто нужно знать, где ее найти и как ее интерпретировать. Именно об этом пойдет речь в этой главе.

Потрясающе! Похоже, из этой документации я смогу узнать все о классах и модулях Ruby. Но как насчет **моего** кода? Как другие люди научатся пользоваться им?



Документация метода класса «today»

```
.today([start = Date::ITALY]) => Object
Date.today #=> #<Date: 2011-06-11 ...>
Creates a date object denoting the present day.
```

Как узнать больше?	364
Базовые классы и модули Ruby	365
Документация	365
Документация в формате HTML	366
Список доступных классов и модулей	367
Поиск информации о методах экземпляра	368
Методы экземпляра обозначены в документации символом #	369
Документация по методам экземпляров	370
Аргументы в сигнатурах вызова	371
Блоки в сигнатурах вызова	372
Описания методов класса	376
Документация методов класса	378
Документация несуществующих классов?!	379
Стандартная библиотека Ruby	380
Поиск классов и модулей стандартной библиотеки	382
Откуда берется документация Ruby: rdoc	383
Какую информацию может вывести rdoc	385
Добавление комментариев для создания документации	386
Ваш инструментарий Ruby	388

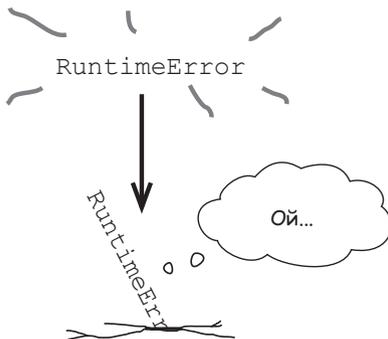
```
#year => Integer
Returns the year.
```

Документация метода экземпляра «year».

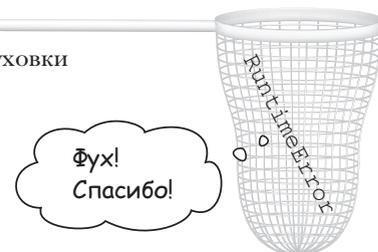
12 Исключения

Непредвиденное препятствие

В реальном мире порой происходит нечто неожиданное. Кто-то удаляет файл, который пытается загрузить ваша программа, или сервер, с которым программа пытается связаться, выходит из строя. Вы можете проверять такие исключительные ситуации в своем коде, но такие проверки перемешиваются с кодом, выполняемым в ходе нормальной работы. (И в результате возникает жуткая мешанина, в которой невозможно разобраться.) Эта глава посвящена средствам обработки исключений в Ruby, которые позволяют писать код для обработки аварийных ситуаций и размещать его отдельно от нормального кода.



Не используйте возвращаемые значения методов для передачи информации об ошибках	390
Использование «raise» для передачи информации об ошибках	392
Использование «raise» создает новые проблемы	393
Исключения: когда происходит что-то не то	394
Условия rescue: пытаемся исправить проблему	395
Как Ruby ищет условие rescue	396
Использование условия rescue с классом SmallOven	398
Описание проблемы от источника	399
Сообщения об исключениях	400
Разная логика rescue для разных исключений	405
Классы исключений	407
Назначение класса исключения для условия rescue	409
Несколько условий rescue в одном блоке begin/end	410
Переход на пользовательские классы исключений в классе SmallOven	411
Перезапуск после исключения	412
Обновление кода с «retry»	413
Код, который должен выполняться в любом случае	415
Условие ensure	416
Гарантированное выключение духовки	417
Ваш инструментарий Ruby	418



13

Модульное тестирование

Контроль качества кода

А вы уверены, что ваша программа работает правильно? Действительно уверены? Прежде чем передавать новую версию пользователям, вы, вероятно, опробовали новые функции и убедились в том, что они работают. Но проверили ли вы *старые* функции, чтобы убедиться, что они не перестали работать? *Все* старые функции? Если этот вопрос не дает вам покоя, значит, вашей программе необходимо автоматизированное тестирование. Автоматизированные тесты гарантируют, что основные компоненты программы продолжают правильно работать даже после изменения кода. **Модульные тесты** — самая распространенная, самая важная разновидность автоматизированных тестов. В поставку Ruby входит библиотека **MiniTest**, предназначенная для модульного тестирования. В этой главе вы узнаете все, что действительно необходимо знать об этой библиотеке!

ListWithCommas
items
join



Автоматизированные тесты помогают найти ошибки до того, как их найдут другие	420
Программа, для которой <u>необходимы</u> автоматизированные тесты	421
Типы автоматизированных тестов	423
MiniTest: стандартная библиотека модульного тестирования Ruby	424
Выполнение теста	425
Тестирование класса	426
Подробнее о тестовом коде	428
Красный, зеленый, рефакторинг	430
Тесты для класса ListWithCommas	431
Исправление ошибки	434
Еще одна ошибка	436
Сообщения об ошибках	437
Другой способ проверки равенства двух значений	438
Другие методы проверки условий	440
Устранение дублирования кода из тестов	443
Метод «setup»	444
Метод «teardown»	445
Обновление кода для использования метода «setup»	446
Ваш инструментарий Ruby	449

14

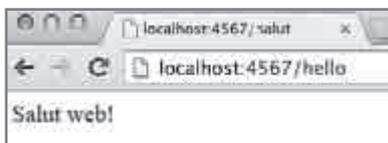
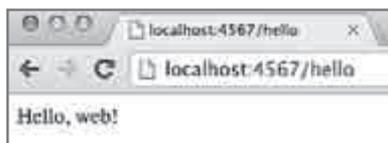
Веб-приложения

На раздаче HTML

На дворе XXI век. Пользователи требуют веб-приложения.

И Ruby поможет вам в этом! Существуют библиотеки, которые помогут вам развернуть собственные веб-приложения и обеспечить доступ к ним из любого браузера. Итак, в двух последних главах этой книги мы покажем вам, как построить полноценное веб-приложение.

Прежде всего вам понадобится Sinatra — независимая библиотека для создания веб-приложений. Но не беспокойтесь, мы научим вас использовать программу RubyGems (включенную в поставку Ruby) для автоматизации загрузки и установки библиотек! Затем будет рассмотрена разметка HTML — ровно столько, сколько необходимо для создания собственных веб-страниц. И конечно, мы покажем, как предоставить эти страницы браузеру!



Написание веб-приложений на Ruby	452
Список задач	453
Структура каталогов проекта	454
Браузеры, запросы, серверы и ответы	455
Загрузка и установка библиотек с использованием RubyGems	456
Установка библиотеки Sinatra	457
Простое приложение Sinatra	458
Ваш компьютер разговаривает сам с собой	459
Тип запроса	460
Маршруты Sinatra	462
Создание списка фильмов в формате HTML	465
Обращение к разметке HTML из Sinatra	466
Класс для хранения данных фильмов	468
Создание объекта Movie в приложении Sinatra	469
Теги внедрения в ERB	470
Тег внедрения вывода в ERB	471
Внедрение названия фильма в разметку HTML	472
Нормальный тег внедрения	475
Перебор названий фильмов в разметке HTML	476
Ввод данных на форме HTML	479
Получение формы HTML для добавления фильма	480
Таблицы HTML	481
Размещение компонентов формы в таблице HTML	482
Ваш инструментарий Ruby	484

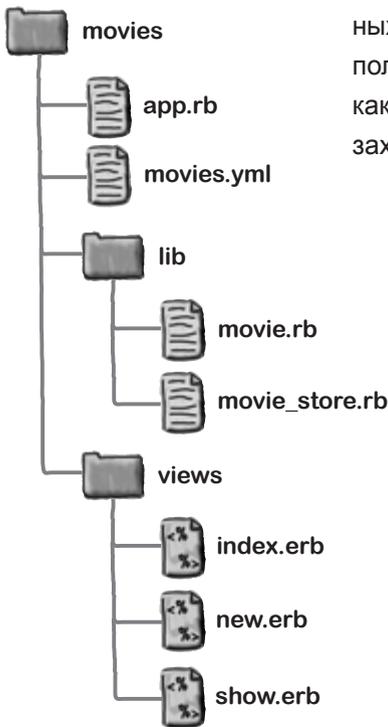
15

Сохранение и загрузка данных

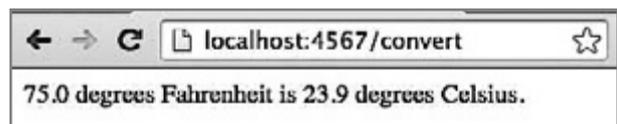
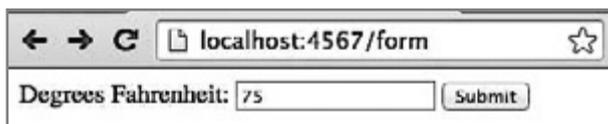
Сбережения пользователя

Сейчас наше веб-приложение просто выбрасывает данные, введенные пользователем. Мы создали форму, на которой пользователь *вводит* данные. Пользователь ожидает, что данные будут *сохранены*, чтобы их можно было *прочитать* и *вывести* позднее. Но сейчас ничего подобного не происходит! Все введенные данные просто *пропадают*.

В последней главе книги приложение будет подготовлено к сохранению данных, введенных пользователем. Мы покажем, как настроить приложение для получения данных формы, как преобразовать эти данные в объекты Ruby, как сохранить их в файле и как загрузить нужный объект, когда пользователь захочет увидеть его. Готовы? Так доделаем наше приложение!



Сохранение и загрузка данных формы	486
Настройка формы HTML для отправки запроса POST	490
Создание маршрута Sinatra для запроса POST	491
Преобразование объектов в строки и YAML	495
YAML::Store и сохранение объектов в файле	496
YAML::Store и сохранение описаний фильмов в файле	497
Поиск объектов Movie в YAML::Store	501
Числовые идентификаторы	502
Класс для управления YAML::Store	505
Использование класса MovieStore в приложении Sinatra	506
Тестирование MovieStore	507
Загрузка всех фильмов из MovieStore	508
Загрузка всех фильмов в приложении Sinatra	510
Построение ссылок HTML на фильмы	511
Именованные параметры в маршрутах Sinatra	514
Поиск объекта Movie в YAML::Store	519
Шаблон ERB для отдельного фильма	520
Полный код приложения	523
Ваш инструментарий Ruby	527



Приложение. Оставшиеся темы

Десять основных тем (не рассмотренных в книге)

Мы прошли долгий путь, и книга почти закончена. Мы будем скучать по вам, но было бы неправильно расставаться и отпускать вас в самостоятельное путешествие без еще *нескольких* напутственных слов. Нам при всем желании не удалось бы уместить все, что вам еще нужно знать о Ruby, на этих страницах... (Вообще-то сначала мы *уместили* все необходимое, уменьшив размер шрифта в 25 000 раз. Все поместилось, но текст было не прочитать без микроскопа, поэтому материал пришлось изрядно сократить.) Но мы оставили все самое лучшее для этого приложения.

И это уже *действительно* конец книги!

1. Другие полезные библиотеки	530
2. Компактные версии if и unless	532
3. Приватные методы	533
4. Аргументы командной строки	535
5. Регулярные выражения	536
6. Синглетные методы	538
7. Вызов любых (даже неопределенных) методов	539
8. Rake и автоматизация задач	541
9. Bundler	542
10. Литература	543

↙ файл в формате CSV

```
Associate,Sale Count,Sales Total
"Boone, Agnes",127,1710.26
"Howell, Marvin",196,2245.19
"Rodgers, Tonya",400,3032.48
```



sales.csv

```
p ARGV[0]
p ARGV[1]
```



args_test.rb

```
File Edit Window Help
$ ruby args_test.rb hello terminal
"hello"
"terminal"
```