Extracted from:

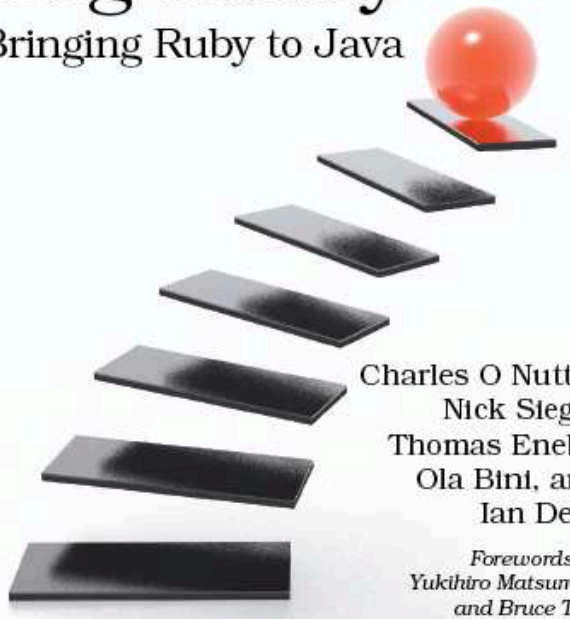
Using JRuby

Bringing Ruby to Java

The Pragmatic Programmers

Using JRuby

Bringing Ruby to Java



Charles O Nutter,
Nick Sieger,
Thomas Enebo,
Ola Bini, and
Ian Dees

Forewords by
Yukihiro Matsumoto
and Bruce Tate

Edited by Jacquelyn Carter

The Facets of Ruby Series

responding to the actions they perform. It consists of three parts: ActionView and ActionController correspond to the View and Controller parts of MVC, while ActionDispatch is responsible for connecting a request to a controller.

A controller in Rails is just a regular class that inherits from ActionController; each public method is an action triggered by something the user does. ActionView is there in the background, but your Ruby classes don't interact directly with it. Views in Rails are templates with names ending in .html.erb by default (the exact suffix varies with the templating system).

ActionPack and ActiveRecord do most of the work in Rails.

ActiveSupport

Rails includes a large number of extensions to the Ruby core classes. It also includes libraries to handle internationalized text, helpers for working with times and dates, and lots of other things.

A lot of these smaller features aren't necessarily tied to web development. For example, date/time math crops up in a lot of applications, on the Web or elsewhere. With ActiveSupport, you can express a time difference as easily as (2.months + 1.day + 3.hours + 15.minutes).ago. Compare that to old-school time arithmetic: Time.now - (2*30*24*3600 + 24*3600 + 3*3600 + 15*60)—and that doesn't even take into account that different months have different lengths.

ActiveSupport is chock full of nice things like this. As you become familiar with it, you'll often find that some utility function you've been wishing for is already included.

ActiveResource

In the bad old days, we tended to think of web apps as little computer programs churning out HTML tag soup. You can write a program like this with Rails, of course. But you'll find it far easier to "cut with the grain" and think in terms of *resources* instead of pages or scripts. This style is known as Representational State Transfer (REST).

Rails makes it easy for you fit your app into this structure. ActionPack helps you create REST services, and ActiveResource helps you consume them—both under similar APIs.

ActionMailer

ActionMailer is a small package that helps you create uniform mail

templates. You can send mail from your controllers and use .erb files as templates for your messages.

ActiveModel

ActiveModel is a new component created in Rails 3 that is basically an extraction of the best bits of ActiveRecord, such as data validations and callbacks. With ActiveModel, you can easily make any Ruby object (not just database classes) at home in Rails.

Bundler

Although not part of Rails, Bundler is a utility developed in parallel with the Rails 3 release to aid in gem dependency management in any Ruby project (even a non-Rails one). Bundler locks down your dependencies to make sure you can repeatably deploy the same configuration across different environments and machines. You'll get comfortable with Bundler in the tutorial shortly.

Most parts of Rails work fine on their own, even in non-Rails applications. For instance, ActiveRecord is widely used in other frameworks and applications. That said, some components are more reusable than others.

There's a lot functionality just in the Rails core. Thanks to the plug-in architecture, there's also a universe of extensions available to take Rails in more directions (or sometimes fewer directions—the Rails team will often spin off a seldom-used feature into a plug-in).

What About JRuby on Rails?

The previous sections described how Rails is put together, and we will soon take a look at how to actually create an application using JRuby on Rails. But first, why would you want to use JRuby together with Rails? The short answer: for exactly the same reasons you would want to use JRuby on *any* project—speed, stability, infrastructure, and so on.

The slightly longer answer is that Rails in its current incarnation is very good at many things but not absolutely everything. JRuby can smooth over some of the remaining rough spots. Deployment is probably the most interesting of these. Deploying a Rails application is fairly well documented, but getting everything right can still be difficult. With

Ian Says. . .

A First Look at REST

A RESTful web service provides a set of discoverable, uniquely named documents (*resources*). Client code—which may or may not be a browser—can read and modify resources by using the HTTP protocol's four simple verbs: POST, GET, PUT, and DELETE.*

For example, suppose you're creating a photo-editing site. With a traditional approach, you might send a GET request to http://example.com/show.php to display an image or send POST requests to new.php, edit.php, or delete.php to upload, modify, or remove an image.

With REST, you'd present each photo as a resource with a unique ID, such as http://example.com/photos/12345. All operations—viewing, modifying, and so on—would take place through GET, POST, PUT, and DELETE requests to that same address.

Think of it as "convention over configuration" applied to your API design.

*. This is not the same thing as the four CRUD (create, read, update, destroy) operations performed by many web apps; see http://jcalcote.wordpress.com/2008/10/16/put-or-post-the-rest-of-the-story.

JRuby, you can package your Rails application as a standard .war file and deploy it to any compliant Java web container.[4]

Rails supports several different databases, but in practice, most shops use either MySQL or PostgreSQL. Since JRuby on Rails allows you to use any database that has a JDBC driver, you have access to a wider range of databases, plus features such as data sources and connection pooling. JRuby on Rails also works very well with JavaDB, the in-memory database that is distributed with Java.

The best way to think of JRuby on Rails is like regular Rails with a few intriguing new possibilities.

4. Web application archives, or .war files, are a standard way of deploying web applications on Java servers.

5.2 Going Rouge

It's time to get started with some code. Through the rest of this chapter, we'll build Rouge, a simple web-based restaurant guide. By the time we've finished, you should be able to build your own JRuby on Rails application. We can't cover all or even most of the functionality that Rails provides—there are other books that can teach you this.[5]

Getting Started

Before starting the tutorial, we need to install Bundler and Rails. The example code in this chapter was written using Rails 3.0.1, Bundler 1.0.2, and activerecord-jdbc-adapter 1.0.1.

To install Bundler and Rails, just type this command:

Download introduction_to_rails/output/gem-install.txt

```
$ jruby -S gem install bundler rails
Successfully installed bundler-1.0.2
Successfully installed activesupport-3.0.1
Successfully installed builder-2.1.2
Successfully installed i18n-0.4.1
Successfully installed activemodel-3.0.1
Successfully installed rack-1.2.1
Successfully installed rack-test-0.5.6
Successfully installed rack-mount-0.6.13
Successfully installed tzinfo-0.3.23
Successfully installed abstract-1.0.0
Successfully installed erubis-2.6.6
Successfully installed actionpack-3.0.1
Successfully installed arel-1.0.1
Successfully installed activerecord-3.0.1
Successfully installed activeresource-3.0.1
Successfully installed mime-types-1.16
Successfully installed polyglot-0.3.1
Successfully installed treetop-1.4.8
Successfully installed mail-2.2.7
Successfully installed actionmailer-3.0.1
Successfully installed rake-0.8.7
Successfully installed thor-0.14.3
Successfully installed railties-3.0.1
Successfully installed rails-3.0.1
24 gems installed
```

Our restaurant guide will make it easy for someone who's considering a restaurant to find reviews for it. They'll want to search restaurants, read

5. See *Agile Web Development with Rails* [RTH08].

reviews, and comment on either a review or a restaurant. Visitors will be generating most of this content, but we'll also need an administrator account for creating restaurants. You'll see later how to offer these two different views of the same data.

Deciding on Our Models

From the previous short description, we can deduce some potential models:

- Restaurant
- Administrator
- Reviewer
- Review
- Comment (attached to a Restaurant)
- Comment (attached to a Review)

We will use a common Comment model for both restaurant comments and review comments—it seems unnecessary to have two different models for essentially the same idea.

Establishing Structure

Rails emphasizes a particular structure for your code. The first step in creating a new application is to generate this structure. The rails new command will build a minimal (but well-organized!) app from scratch, using the directory name you provide on the command line. We'll choose the name rouge for our project directory.

Download introduction_to_rails/output/rails-rouge.txt

```
$ jruby -S rails new rouge --template http://jruby.org
      create
      create  README
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
      create  app
      create  app/controllers/application_controller.rb
      create  app/helpers/application_helper.rb
      create  app/views/layouts/application.html.erb
      create  app/mailers
      create  app/models
      create  config
      create  config/routes.rb
```

Nick Says...

You Have Options

The rails new command supports a number of options. A couple of the more interesting ones are --database mysql for setting up an application for use with MySQL, or --skip-active-record for avoiding using ActiveRecord or databases at all. See jruby -S rails help new for more information.

Rails tells you exactly which directories and files get created. Reproducing the entire list here would take more than two pages; for the trees' sake, we've truncated the output. As you can see by the directory names, there is one specific place for each piece of functionality you would want to add to your application.

The directories you will spend most of your time in from now on are the following:

- app: Contains most of the application's functionality—models, controllers, and views.
- config: Holds configuration settings, such as the database server location.
- test: Go on, guess!

You'll notice we passed an extra --template http://jruby.org option when we generated the application. This flag tells Rails to apply some extra JRuby-specific configuration to the new application.

If you are following along and ran the command yourself, you might have noticed a couple of extra lines at the bottom of the rails new command output:

Download introduction_to_rails/output/rails-rouge.txt

```
apply  http://jruby.org
apply    http://jruby.org/templates/default.rb
 gsub      Gemfile
```

JRuby needs to use the activerecord-jdbc-adapter gem to connect to databases via Java's JDBC API, so JRuby has made some small mod-

ifications to the default Rails application's Gemfile. What goes in the Gemfile, you say? We're glad you asked!

Installing Dependencies with Bundler

Bundler's stated goal is to "manage an application's dependencies through its entire life across many machines systematically and repeatably."[6] In more pragmatic terms, it helps prevent conflicting or missing gems. Although you can use Bundler with any Ruby application, the integration story is particularly good with Rails 3.

As we hinted in the previous section, one of the files the rails new command creates is called Gemfile. Let's take a look inside:

Download introduction_to_rails/output/Gemfile

```
source 'http://rubygems.org'

gem 'rails', '3.0.1'

# Bundle edge Rails instead:
# gem 'rails', :git => 'git://github.com/rails/rails.git'

if defined?(JRUBY_VERSION)
  gem 'activerecord-jdbc-adapter'
  gem 'jdbc-sqlite3', :require => false
else
  gem 'sqlite3-ruby', :require => 'sqlite3'
end
```

The Gemfile is just a place to declare the gems and libraries your application needs. Bundler shines when it's time to configure those dependencies at install time and runtime. To make Bundler install the dependencies, run the bundle install command:

Download introduction_to_rails/output/bundle-install.txt

```
$ jruby -S bundle install
Fetching source index for http://rubygems.org/
Using rake (0.8.7)
Using abstract (1.0.0)
Using activesupport (3.0.1)
Using builder (2.1.2)
Using i18n (0.4.1)
Using activemodel (3.0.1)
Using erubis (2.6.6)
Using rack (1.2.1)
Using rack-mount (0.6.13)
```

6. http://gembundler.com/

```
Using rack-test (0.5.6)
Using tzinfo (0.3.23)
Using actionpack (3.0.1)
Using mime-types (1.16)
Using polyglot (0.3.1)
Using treetop (1.4.8)
Using mail (2.2.7)
Using actionmailer (3.0.1)
Using arel (1.0.1)
Using activerecord (3.0.1)
Installing activerecord-jdbc-adapter (1.0.1)
Using activeresource (3.0.1)
Using bundler (1.0.2)
Installing jdbc-sqlite3 (3.6.14.2.056)
Using thor (0.14.3)
Using railties (3.0.1)
Using rails (3.0.1)
Your bundle is complete! Use `bundle show [gemname]`
to see where a bundled gem is installed.
```

The beauty of having the dependencies stored in Gemfile is that you can ensure that anyone else working on your application has the same set of libraries. Everyone simply needs to remember to run bundle install (the first time) or bundle update (when someone changes the Gemfile).

Configuring the Database

The next step after creating a new Rails application is to configure your database. Open config/database.yml. It will consist of three sections named after the three standard environments Rails creates for you: test, development, and production. Here's the setup for the development database, which is the one you'll use during most of this chapter:[7]

Download introduction_to_rails/rouge/config/database.yml

```
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000
```

If you were developing the application with a database server such as MySQL or PostgreSQL, you'd edit this file to change the connection information. Since we'll be using the embedded SQLite database, there's no need to change anything here for now.

7. Your automated tests will use the test database instead. It's important to keep this one separate, since Rails destroys and re-creates it every time you run the tests.

Ola Says...

Whitespace in Config Files

Be very careful when editing YAML files (files that end in .yml or .yaml)—one single tab character in these files will render them unreadable to Ruby. If you see strange errors after editing database.yml, check your whitespace for tabs.

Before you start the application, we should point out there is another step you'd need to perform had we started with MySQL: creating the databases. Rails provides a handy command that will create a separate database for each environment. As with many maintenance tasks, you run it using Rake, the Ruby build and maintenance tool.[8]. This step is unnecessary with SQLite, which will create the files for us the first time our Rails app hits the database. If you're really curious, you can safely run the command anyway:

Download introduction_to_rails/output/rake-db-create.txt

```
$ jruby -S rake db:create:all
(in code/introduction_to_rails/rouge)
```

The Rails application is now ready to start:

Download introduction_to_rails/output/script-server.txt

```
$ jruby script/rails server
=> Booting WEBrick
=> Rails 3.0.1 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2010-10-15 11:08:40] INFO  WEBrick 1.3.1
[2010-10-15 11:08:40] INFO  ruby 1.8.7 (2010-10-13) [java]
[2010-10-15 11:08:40] INFO  WEBrick::HTTPServer#start: pid=6137 port=3000
```

You should be able to visit http://localhost:3000 and see the standard Rails welcome page.

8. We'll cover Rake in more detail in Chapter 7, *Building Software for Deployment*, on page 163.

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Home Page for Using JRuby
http://pragprog.com/titles/jruby
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/jruby.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)