Ruby On Rails

By:
Sahar Jambi

Object Oriented A&D

Spring 2011

University of Colorado at Boulder

Outline

Ruby on **Rails**

- What is Ruby
- Why Ruby
- Ruby Features
- OOP in Ruby
- Dynamic Ruby
- What is Rails
- Rails History
- Why Rails
- Rails Design Paradigm
- Strength of Rails
- Rails Framework
- Rails Architecture
- Rails and MVC
- Rails Scripts
- Rails Models
- Rails Controllers
- Rails Views
- Routing in Rails
- Real Applications
- Conclusion
- References
- Available Resources

What is Ruby?

A High Level Programming Language that provides an elegant, natural syntax.

Created by Yukihiro Matsumoto (Matz), Japan, 1995.

What is Ruby?

- Object-Oriented like Java
- Interpreted Scripting like Perl and Python
- Introspection
- Domain-Specific Languages

Why Ruby?

More understandable code in less lines

(Less code means fewer bugs)

Open Source with rich libraries (open license)

Extensible

Scripted language

Ruby easy to learn

Ruby Features

Line Orientation

Getter, Settter and attr_ methods

Naming Conventions

Modules vs Classes

Method Missing

Iterators and Blocks

Method Chaining

OOP in Ruby

Ruby is purely object-oriented

Everything is an object, even strings and numbers!

e.g.

```
- # Output "Beautiful Ruby!" 10 times
  10.times do
      puts “Beautiful Ruby!”
  end

- “sahar”.capitalize
- 3.methods
- 2.hours.ago
```

Dynamic Ruby

Ruby is highly dynamic

Ruby supports many dynamic features

- Duck Typing

"If it quacks like a duck, it must be a duck."

What is Rails?

Ruby on Rails (Rails or RoR) is

An open source application framework composed of several libraries; working together to supply a complete framework for building Web applications.

Written in Ruby by David Heinemeier Hansson.

Rails History

In 2003, while Hansson was building “Basecamp”, a web application for “37signals” company, he created a core of functionality to reuse on his other applications, then turned it into an open-source project that became *Ruby on Rails*.

In Jun 2004, Rails 0.5 was presented to the public as the first time (at IRC).

In Dec 2005, Rails 1.0 was the first public release with an incredibly proud.

In Dec 2007, Rails 2.0 was a fantastic release with great new features, loads of fixes and an incredible amount of polish.

In Aug 2010, Rails 3.0 is the third generation of Rails with the work of more than 1,600 contributors to make Rails better, faster, cleaner, and more beautiful.

Why Rails?

- It's an extremely productive database-driven web application framework.
- It's all about productivity that allows the developer to focus more on the business logic of the application, rather than the technical, behind-the-scenes stuff.
- It makes working with a database extremely simple.
- It makes development fun and easy for the developer.
- Almost all of the development is in Ruby, one language!.
- The creator of the Ruby (Matz) has referred to Rails as *Ruby's Killer App*.

Rails Design Paradigm

DRY

Don’t Repeat Yourself (DRY) is a philosophy that can be seen throughout the Rails framework.

DRY philosophy means that you should not have to repeat yourself in code, configuration, or in many cases even documentation, within a single Web application.

Rails Design Paradigm

Convention Over Configuration

Rails relies on accepted convention over configuration.

Rails asks developers to write code not configuration files.

Rails Design Paradigm

Opinionated Software

Rails contains built-in opinions of how Web applications should be developed and designed.
This aspect has been a large contributor to its ease of use and its overall success.

Strength of Rails

- Creates denser code that's is easy to work with and maintain.
- Uses metaprogramming techniques that use programs to write programs.
- Scaffolding that builds code skeletons.
- Migration that manages the database tables and columns.
- Embraces test-driven development.
- Supports three environments: development, testing, and production.
- Supports many database back-ends, such as MySQL, PostgreSQL, SQLite, SQL Server, Oracle, DB2

Rails Scripts

Rails provides some excellent tools that assist in the developing process (run from the command-line).

- Rails Console
  A command-line utility

- WEBrick Web server
  Local developer testing web server

- Generators
  Generates class files for framework layers

- Migrations
  Routines help to manage the database

Rails Framework

Rails provides a **full-stack** framework. This means that Rails provides all the pieces needed to build a complete Web application in one package.

Using the analogy of building a house, Rails supplies the complete plumbing, electrical, and framework already built. You just have to add the functionality and features specific to your application.

The basic functions provided by Rails are:

- HTML templating
- Database storage and retrieval
- Handling of Web request and response
- HTML form handlinger.

Rails Architecture

Rails implements the **Model-View-Controller** (MVC) architecture to create a full-stack framework in which all layers are built to work together.

The implementation of Rails is divided into libraries based around each layer of the MVC pattern.

Rails and MVC

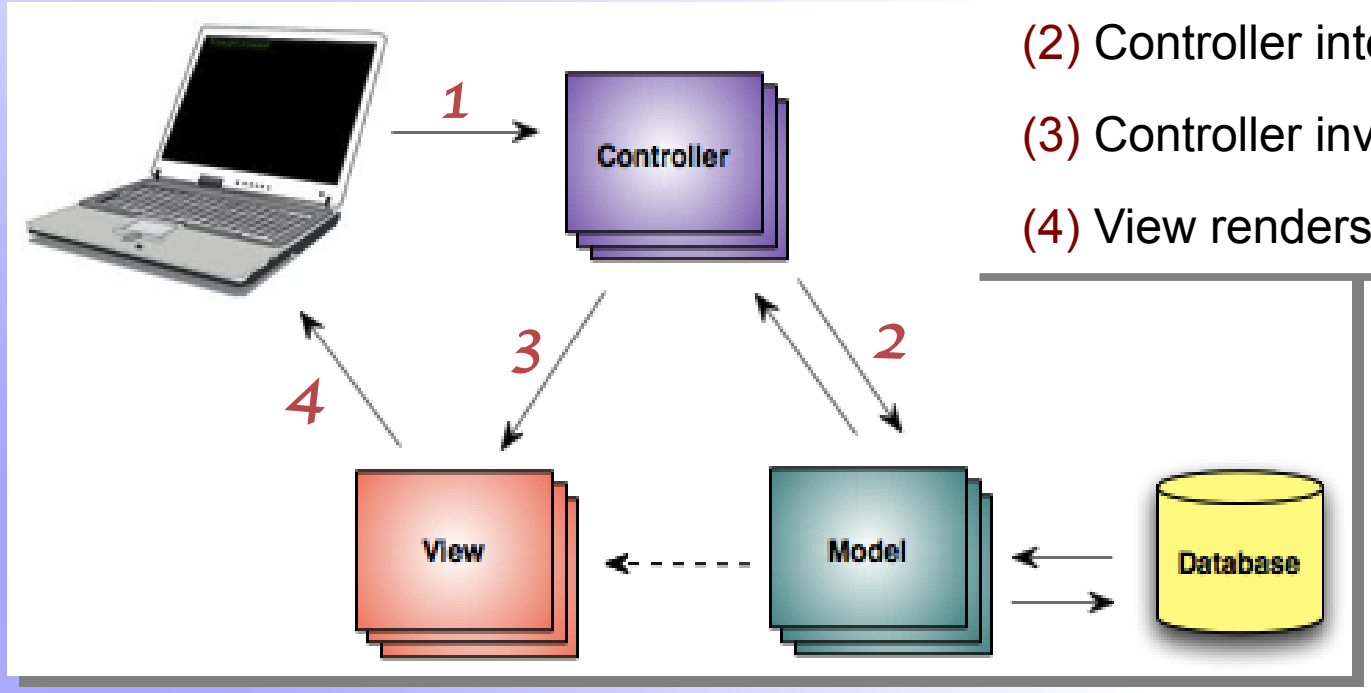
The **MVC** design pattern is at the core of the Rails framework.

The **Model** layer communicates with a database and performs necessary business logic required to manipulate the data.

The **View** layer represents the Web pages that make up the user interface of a Web application.

The **Controller** layer handles the HTTP requests and communicates with the model layer.

Rails and MVC



The Rails request cycle

(1) Browser sends request.

(2) Controller interacts with model.

(3) Controller invokes view.

(4) View renders next browser screen.

Rails Models

Rails applications use model layer to:

- Implement an object relational mapping (ORM) to map relational database tables to object-oriented classes.
- Perform create, read, update, and delete (CRUD) operations.
- Define relationships in the data models using domain-specific language (DSL).
- Dynamically generate accessors, finders, and validators.

ActiveRecord

Rails implements the Model layer using a component called ActiveRecord, that provides a powerful abstraction layer.

ActiveRecord is implemented as a set of base classes from which the model objects extend. The model objects inherit a wealth of functionality.

Active Record uses Migration methods to manipulate tables, columns, and indexes.

ActiveRecord

An example of using ActiveRecord to create a model object:

To create the User model for users database table, use the generate script:

```
ruby script/generate model User
```

The model file user.rb will be created in app/models/ folder.

In this new model file, you will have:

```
class User < ActiveRecord::Base

end
```

Rails Controllers

Rails use controllers to:

- Handle incoming browser requests.
- Call appropriate functions on model objects.
- Render a view templates into pure HTML.

Each of the application's model classes will typically have a controller class for working with that model.

ActionController

Rails controllers are implemented using the ActionController component.

Action Methods are methods contained in controllers that have requests routed to them.

The ActionController provides functionality related to:

- Routing
- Interfacing with the Web server
- Using sessions
- Cache management
- Rendering view templates.

ActionController

An example of using ActionController to create a controller:

To create a controller for User model, use the generate script:

```
ruby script/generate controller User
```

The controller file user_controller.rb will be created in app/controllers/ folder.

In this new controller file, you will have:

```
class UserController < ApplicationController::Base

end
```

To add an action method called show, write:

```
def show
  user_id = params[:id]
  @user = User.find(user_id)
  render :template => "show"
end
```

Rails Views

Rails uses view layer to present the application to the end users.

The view layer shouldn't handle any business or processing logic, but it is as important to the success of an application as the model and controller layers.

The view layer is implemented in ERB template files. These files contain a mixture of HTML and embedded Ruby code (Erb).

ActionView

The Rails component that manages the view layer is called ActionView that provides:

- Embedded Ruby (ERb)
- Layout templates
- Partial templates
- Helper methods

ActionView

In a Rails application, each controller has its own subdirectory for views templates relating to actions in that controller.

All instance variables in ActionController are copied into the ActionView instance before view invocation.

An example of creating a View template:

The show action in UserController will create the show template, such that show.html.erb. is created in the app/views/user folder.

Routing in Rails

The Rails routing mechanism is very flexible and can be adapted to meet any special requirements.

The basic routing functionality is created immediately without having to write a single line of configuration code.

The URL http://host:port/:controller/:action/:id determines

- Which controller
- Which action
- Which object

Real Applications

Commercial Rails Applications:

- Basecamp: http://www.basecamphq.com
- 43 Things: http://43things.com
- Ta-Da Lists: http://tadalists.com
- Snowdevil: http://www.snowdevil.ca
- Bellybutton: http://www.bellybutton.de
- Backpack: http://www.backpackit.com/

Conclusion

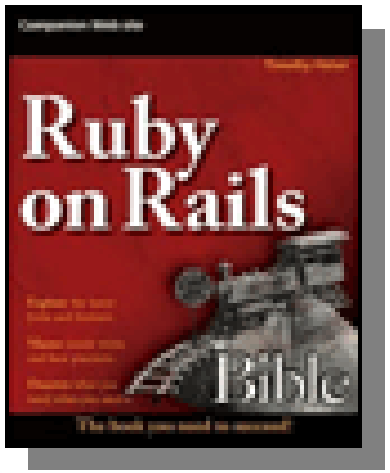
Ruby on Rails is an extensive open-source framework for developing database-backed web applications that provides high productivity and robust applications in very competitive development time, including super beautiful and maintainable Ruby code.

References

Books

Ruby on Rails Bible, by Timothy Fisher

John Wiley & Sons © 2008



Internet resources

http://rubyonrails.org/

http://en.wikipedia.org/wiki/Ruby_on_Rails

http://weblog.rubyonrails.org/

Available Resources

Books

- Agile Web Development With Rails
- Rails Recipes
- Programming With Rails
- Ruby on Rails: Up and Running
- Active and growing community

Internet resources

- wiki.rubyonrails.com
- api.rubyonrails.com
- www.ruby-doc.org/