Lecture 18: Ruby on Rails

Wendy Liu
CSC309F – Fall 2007

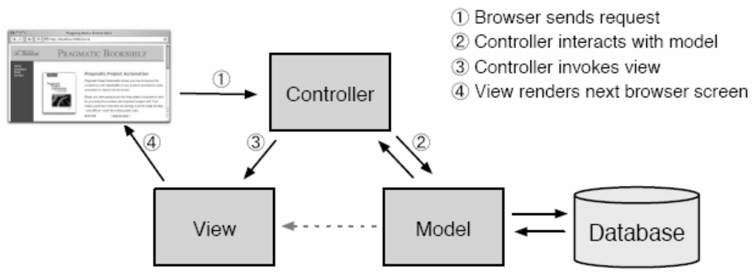
Outline

- Introduction to Rails
- Rails Principles
- Inside Rails
- Hello World
- Rails with Ajax
- Other Framework

Introduction to Rails

"Agile Web Development with Rails"
2nd Ed, D. Thomas et al.

MVC


① Browser sends request
② Controller interacts with model
③ Controller invokes view
④ View renders next browser screen
Controller
View
Model
Database
Pragmatic Bookshelf

Rails

- Rails enforces a structure for the application
  - MVC architecture
  - Models, views, and controllers are developed as separate chunks of functionality and knitted together by Rails
  - Little external configuration metadata is needed
    - Convention over configuration philosophy

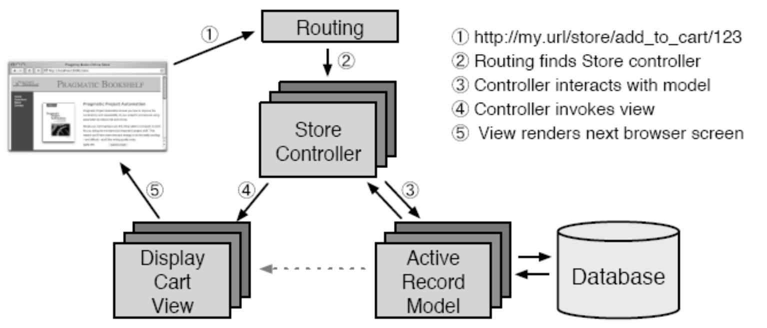
5

A Typical Rails Application

- Incoming requests are first sent to a <u>router</u>, which determines where to send and how to parse the request
- A particular method, or <u>action</u>, is called in some controller
- The action might look at data in the request, interact with the <u>model</u>, or cause other <u>actions</u> to be invoked
- The action prepares information for the <u>view</u>, which renders the output to the user

6

Rails and MVC


Routing
Store Controller
Display Cart View
Active Record Model
Database
① http://my.url/store/add_to_cart/123
② Routing finds Store controller
③ Controller interacts with model
④ Controller invokes view
⑤ View renders next browser screen

- Controller: store
- Action: add_to_cart
- Model: (shopping cart) params[:id] 123

7

Why Rails

- MVC seems to do the trick, why do we need Rails?
  - Rails handles all of the <u>low-level housekeeping</u> for you—all those messy details that take so long to handle by yourself—and lets you concentrate on your application's core functionality

8

Rails Principles

Painstaking Mappings

- Mappings we have seen and used
  - Deployment descriptors
    - Servlet mappings (URL to Servlet class)
  - Persistence data binding
    - JDBC data mappings (Class to table)
  - Transport data binding
    - WSDL and JAX-B data binding (Class/Field to XML element/attribute)
  - MVC mapping
    - Model to View to Controller (usually hard coded)
- All stored and performed differently
  - Learning curve, maintenance cost

Top Two Principles

- Convention over configuration
  - No explicit mapping needed
    - Such as deployment descriptors
  - Use naming conventions to automatically perform the mapping
    - E.g. Store controller vs. store view
  - To fight back the proliferation of configurations seen in J2EE
- DRY (Don't Repeat Yourself)
  - Information is located in a single, unambiguous place

Inside Rails

Components of Rails

- Active Record
  - Model
- Action Pack
  - View and Controller

OO vs RDB

- Applications are often Object Oriented, they also keep information in a relational database
  - Relational databases are designed around mathematical set theory
    - All about sets of values
  - Objects are all about data and operations
- Operations that are easy to express in relational terms are sometimes difficult to code in an OO system, and vice versa

Object-Relational Mapping (ORM)

- ORM libraries map database tables to classes
  - If a database has a table called orders, we will have a corresponding class named Order
  - Rows in this table correspond to objects (instances) of the class
    - A particular order is represented as an object of class Order
  - Within that object, attributes are used to get and set the individual columns
    - Our Order object has methods to get and set the amount, the sales tax, and so on

ORM (cont'd)

- Rails (model) classes provide a set of class-level methods that perform table-level operations
  - For example, we might need to find the order with a particular id
    - This is implemented as a class method that returns the corresponding Order object

Active Record

- Model support in Rails
- The ORM layer supplied with Rails
  - tables map to classes, rows to objects, and columns to object attributes

17

Example: Active Record

- SQL

```
CREATE TABLE people (
  id INT(11) NOT NULL auto_increment,
  name VARCHAR(100),
  PRIMARY KEY (id)
)
```

- Active Record

```
class Person < ActiveRecord::Base; end

Person.create(:name => "Lucas Carlson")
lucas = Person.find_by_name("Lucas Carlson")
lucas.name = "John Doe"
lucas.save
```

18

Action Pack

- Bundles both views and controllers
  - The view and controller parts of MVC are pretty intimate
    - The controller supplies data to the view
    - The controller receives events from the pages generated by the views
- Rails provides a clear separation for control and presentation logic

19

Action Pack: View Support

- Creating either all or part of a page to be displayed in a browser
- Dynamic content is generated by templates
  - rhtml
    - Embeds snippets of Ruby code within the view's HTML
  - rxml
    - Lets you construct XML documents using Ruby code
    - The structure of the generated XML will automatically follow that of the code
  - rjs
    - Allows you to create JavaScript fragments on the server which are to be executed on the browser
    - Great for creating dynamic Ajax interfaces

20

Example: Rails View

```
<h1>Hello world!</h1>
<p>The count is <%= @some_number %></p>

<ul>
<% for i in 0..@some_number do %>
  <li><%= i %></li>
<% end %>
</ul>
```

Action Pack: Controller

- Coordinates the interaction between the user, the views, and the model
  - Rails handles most of this interaction behind the scenes
    - You only need to add the application-level functionality
- Other responsibilities
  - Routing external requests to internal actions
  - Managing caching
    - Give applications orders-of-magnitude performance boosts
  - Managing helper modules
    - Extend the capabilities of the view templates without bulking up their code
  - Managing sessions
    - Giving users the impression of ongoing interaction with the applications

Example: Rails Controller

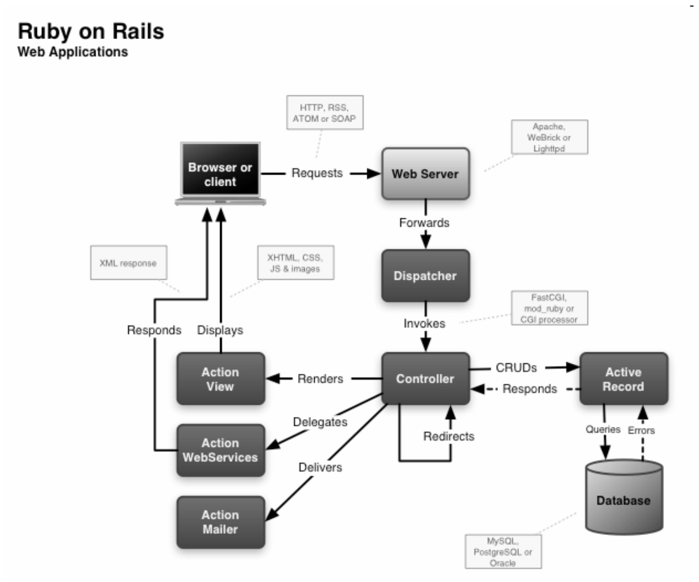
```
class PersonController < ApplicationController
  def index
    # local to the method ONLY
    some_number = 5
  end

  def count
    # local to the method AND the view
    @some_number = 5
  end
end
```

How to Interpret URLs in Rails

- Example:

http://localhost:3000/person/count/123

  - person translates to the PersonController class
  - count translates to the count method
  - 123 translates to the value of params[:id]



That's Not All

- ActionWebServices
  - Create SOAP and XML-RPC web services in minutes
- XML views
  - Create RSS in seconds
- Easy, well integrated unit testing
- Automated documentation generation
- Automated benchmarking and integrated logging
- Interactive debugger
- Easy custom routes
- Plug-ins

Hello World

Build a Rails App:

Illustrating Convention over Configuration

Create Application: demo

```
 work> rails demo
 create
 create   app/controllers
 create   app/helpers
 create   app/models
   :      :      :
create   log/development.log
create   log/test.log
work>
```

Directory Listing: demo

```
work> cd demo
demo> ls -p
README           components/     doc/
Rakefile         config/         lib/
app/             db/             log/
public/          tmp/
script/          vendor/
test/
```

29

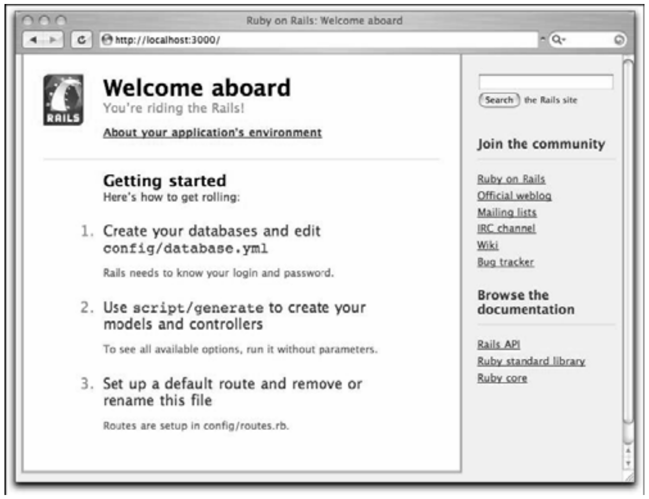
Starts Web Server

```
demo> ruby script/server
=> Booting WEBrick...
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
[2006-01-08 21:44:10] INFO  WEBrick 1.3.1
[2006-01-08 21:44:10] INFO  ruby 1.8.2 (2004-12-30) [powerpc-darwin8.2.0]
[2006-01-08 21:44:11] INFO  WEBrick::HTTPServer#start: pid=10138 port=3000
```

- Starts a stand-alone web server that can run our newly created Rails application under WEBrick (default)

30

Newly Created Rails Application



Welcome aboard
You're riding the Rails!
About your application's environment

Getting started
Here's how to get rolling:

1. Create your databases and edit config/database.yml
   Rails needs to know your login and password.
2. Use script/generate to create your models and controllers
   To see all available options, run it without parameters.
3. Set up a default route and remove or rename this file
   Routes are setup in config/routes.rb.

31

Create New Controller: Say

```
demo> ruby script/generate controller Say
exists   app/controllers/
exists   app/helpers/
create   app/views/say
exists   test/functional/
create   app/controllers/say_controller.rb
create   test/functional/say_controller_test.rb
create   app/helpers/say_helper.rb
```

32

Inside Say Controller

app/controllers/say_controller.rb

```
class SayController < ApplicationController
end
```

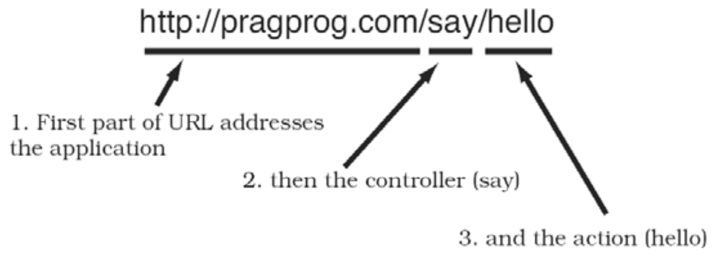
33

Add Method to Say Controller

```
class SayController < ApplicationController
  def hello
  end
end
```

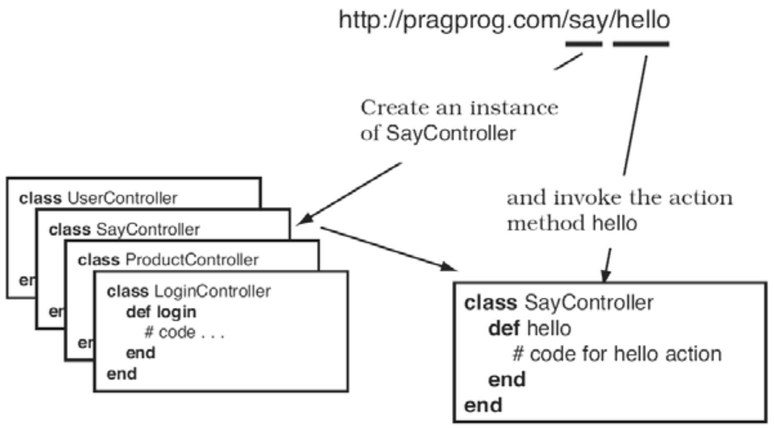
34

URLs: Mapped to Controllers and Actions


http://pragprog.com/say/hello
1. First part of URL addresses the application
2. then the controller (say)
3. and the action (hello)

35

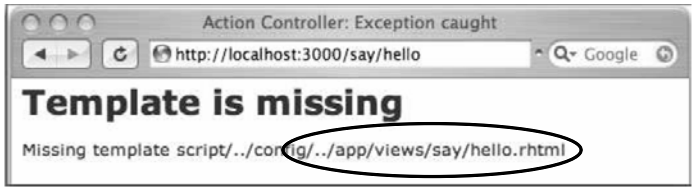
Rails Routes to Controllers and Actions



36

Go to the Link

- http://localhost:3000/say/hello


Action Controller: Exception caught
http://localhost:3000/say/hello
Google

Template is missing

Missing template script/../config/../app/views/say/hello.rhtml

37

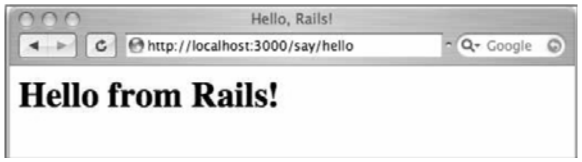
Create a Template (View)

- By default, Rails looks for templates in a file with the same name as the action it's handling
- We need to create a file called hello.rhtml in the directory app/views/say/

```
<html>
  <head>
    <title>Hello, Rails!</title>
  </head>
  <body>
    <h1>Hello from Rails!</h1>
  </body>
</html>
```

38

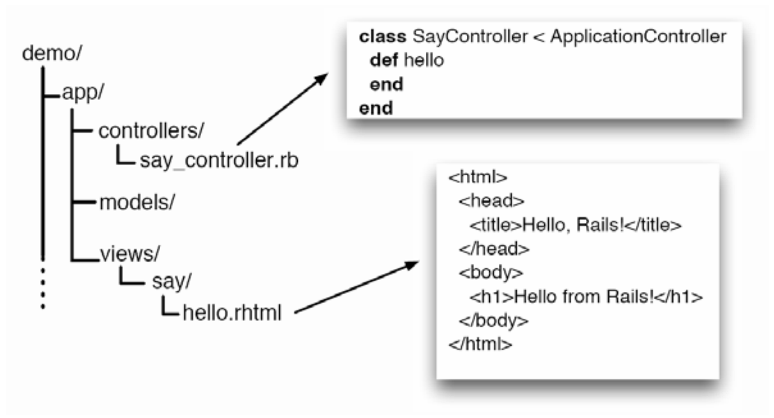
Hello!



Hello from Rails!

39

Standard Locations for Controllers and Views



```
class SayController < ApplicationController
  def hello
  end
end
```

```
<html>
  <head>
    <title>Hello, Rails!</title>
  </head>
  <body>
    <h1>Hello from Rails!</h1>
  </body>
</html>
```

40

Adding Dynamic Content

```
class SayController < ApplicationController
  def hello
    @time = Time.now
  end

  def goodbye
  end
end
```

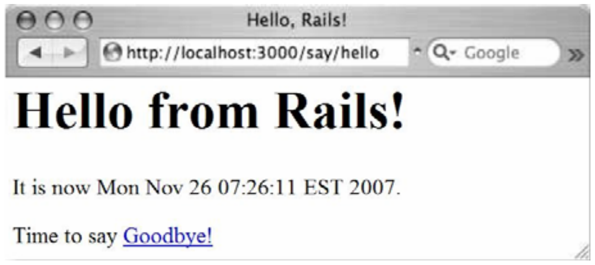
```
<html>
  <head>
    <title>Hello, Rails!</title>
  </head>
  <body>
    <h1>Hello from Rails!</h1>
    <p>
      It is now <%= @time %>.
    </p>
    <p>
      Time to say
      <%= link_to "Goodbye!", :action => "goodbye" %>
    </p>
  </body>
</html>
```

work/demo/app/views/say/goodbye.rhtml

```
<html>
  <head>
    <title>See You Later!</title>
  </head>
  <body>
    <h1>Goodbye!</h1>
    <p>
      It was nice having you here.
    </p>
  </body>
</html>
```

41

Hello and Goodbye!



Hello from Rails!

It is now Mon Nov 26 07:26:11 EST 2007.

Time to say Goodbye!

42

Story So Far

1. The user navigates to our application
   - In our case, we do that using a local URL such as http://localhost:3000/say/hello
2. Rails analyzes the URL
   - The say part is taken to be the name of a controller, so Rails creates a new instance of the Ruby class SayController (which it finds in app/controllers/say_controller.rb)
3. The next part of the URL path, hello, identifies an action
   - Rails invokes a method of that name in the controller
   - This action method creates a new Time object holding the current time and tucks it away in the @time instance variable

43

Story So Far (cont'd)

3. Rails looks for a template to display the result
   - It searches the directory app/views for a subdirectory with the same name as the controller (say) and in that subdirectory for a file named after the action (hello.rhtml)
4. Rails processes this template through ERb (Embedded Ruby), executing any embedded Ruby and substituting in values set up by the controller
5. The result is returned to the browser, and Rails finishes processing this request

44

Rails with Ajax

45

RJS Template

- Idea: Your XHR calls can return JavaScript to execute in the browser
  - Solving problems that otherwise require a great deal of complex JavaScript on the client
- A RJS template is a file in app/views/ with an .rjs extension
- When a request comes in from XHR, the dispatcher will preferentially look for an .rjs template
  - The template is parsed, JavaScript is generated and returned to the browser, where it is finally executed

46

Other Framework

47

Related Framework

- Django for Python
  - Features:
    - MVC-based
    - Object-relational mapping
    - DRY (Don't Repeat Yourself) Principle
  - Ease the creation of complex, DB-driven web development
  - Emphasize reusability and pluggability
    - Automate as much as possible
  - Support for rapid development
  - http://www.djangoproject.com/

48