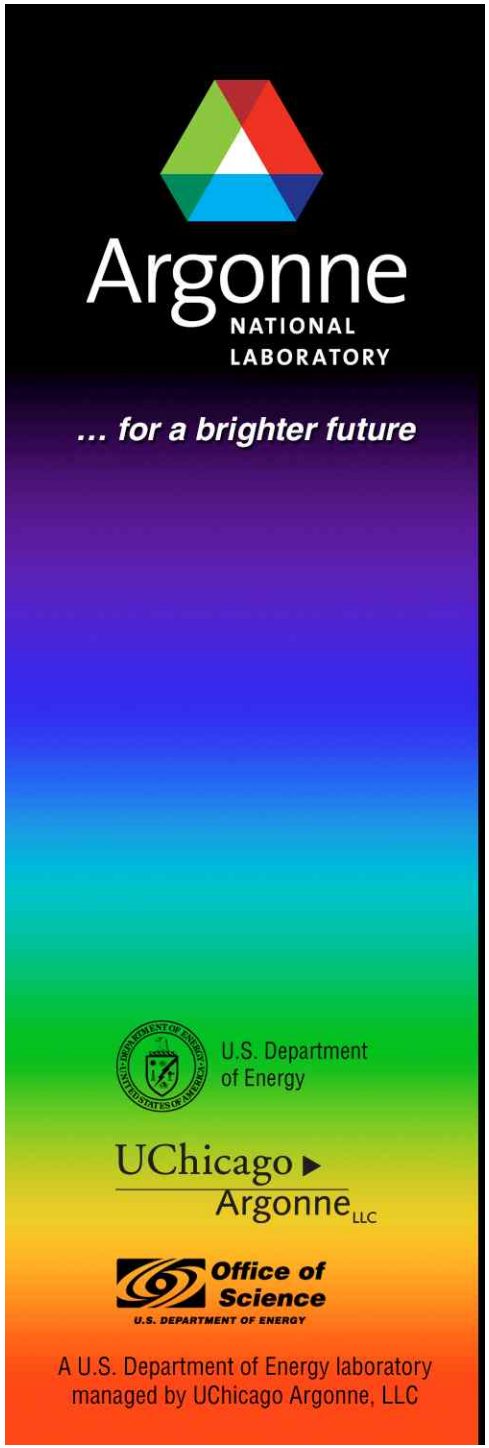


Ruby on Rails

::The New Gem of Web Development

Ross Pallan

IT Project Manager

Argonne National Laboratory

rpallan@anl.gov

Ruby on Rails

- Ruby on Rails is a web application framework written in Ruby, a dynamically typed programming language. The amazing productivity claims of Rails is the current buzz in the web development community. If your job is to create or manage web applications that capture and manipulate relational database from a web-based user interface, then Ruby on Rails may be the solution you've been looking for to make your web development life easier.
- In this presentation:
  - Get an introduction to Ruby on Rails
  - Find out what's behind the hype
  - See it in action by building a fully functional application in minutes

What is Ruby?

- Ruby is a pure object-oriented programming language with a super clean syntax that makes programming elegant and fun.
  - In Ruby, everything is an object
- Ruby is an interpreted scripting language, just like Perl, Python and PHP.
- Ruby successfully combines Smalltalk's conceptual elegance, Python's ease of use and learning and Perl's pragmatism.
- Ruby originated in Japan in 1993 by Yukihiro “matz” Matsumoto, and has started to become popular worldwide in the past few years as more English language books and documentation have become available.
- Ruby is a metaprogramming language. Metaprogramming is a means of writing software programs that write or manipulate other programs thereby making coding faster and more reliable.

What is Rails?
Ruby on Rails or just Rails (RoR)

- Rails is an open source Ruby framework for developing database-backed web applications.
- Created by David Heinemeier Hansson – DHH Partner, 37Signals
  http://money.cnn.com/magazines/business2/peoplewhomatter/
- The Rails framework was extracted from real-world web applications. That is, Rails comes from real need, not anticipating what might be needed. The result is an easy to use and cohesive framework that's rich in functionality, and at the same time it does its best to stay out of your way.
- All layers in Rails are built to work together so you Don't Repeat Yourself and can use a single language from top to bottom.
- Everything in Rails (templates to control flow to business logic) is written in Ruby
  - Except for configuration files - YAML

Rails Strengths – It's all about Productivity

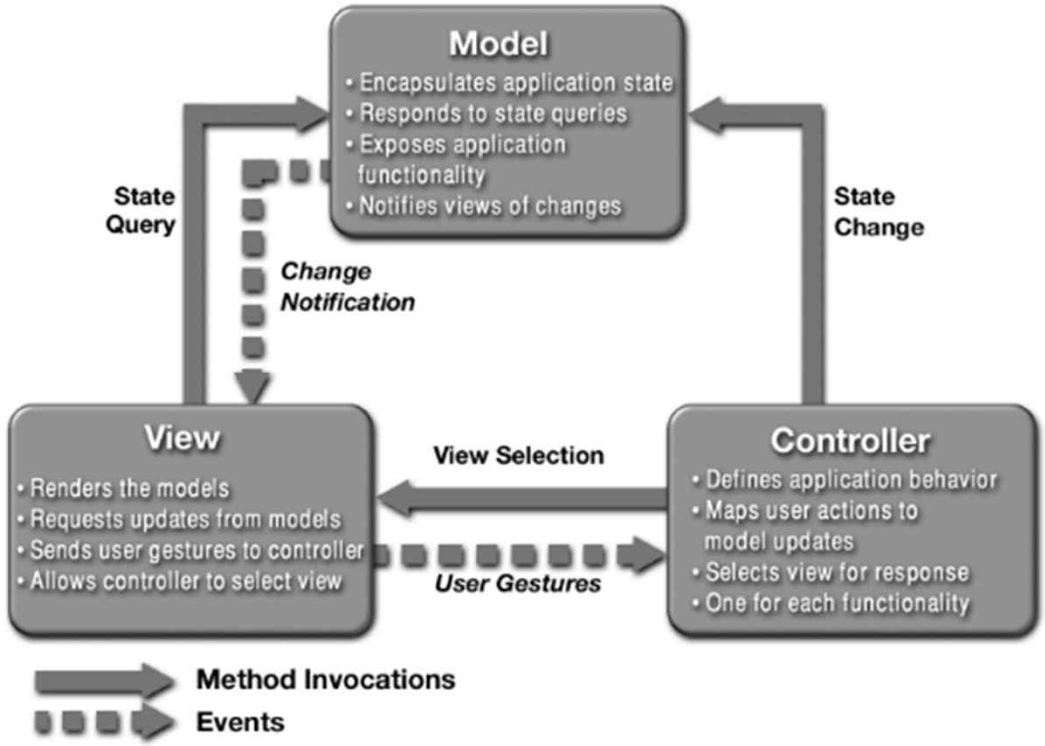
- Metaprogramming techniques use programs to write programs. Other frameworks use extensive code generation, which gives users a one-time productivity boost but little else, and customization scripts let the user add customization code in only a small number of carefully selected points
  - Metaprogramming replaces these two primitive techniques and eliminates their disadvantages.
  - Ruby is one of the best languages for metaprogramming, and Rails uses this capability well.
- Scaffolding
  - You often create temporary code in the early stages of development to help get an application up quickly and see how major components work together. Rails automatically creates much of the scaffolding you'll need.

Rails Strengths – Write Code not Configuration

- *Convention over configuration*
  - Most Web development frameworks for .NET or Java force you to write pages of configuration code. If you follow suggested naming conventions, Rails doesn't need much configuration. In fact, you can often cut your total configuration code by a factor of five or more over similar Java frameworks just by following common conventions.
    - *Naming your data model class with the same name as the corresponding database table*
    - *'id' as the primary key name*
- Rails introduces the Active Record framework, which saves objects to the database.
  - Based on a design pattern cataloged by Martin Fowler, the Rails version of Active Record discovers the columns in a database schema and automatically attaches them to your domain objects using metaprogramming.
  - This approach to wrapping database tables is simple, elegant, and powerful.

Rails Strengths – Full-Stack Web Framework

- Rails implements the model-view-controller (MVC) architecture. *The MVC design pattern separates the component parts of an application*
  - Model encapsulates data that the application manipulates, plus domain-specific logic
  - View is a rendering of the model into the user interface
  - Controller responds to events from the interface and causes actions to be performed on the model.
  - *MVC pattern allows rapid change and evolution of the user interface and controller separate from the data model*



Rails Strengths

- Rails embraces test-driven development.
  - *Unit testing: testing individual pieces of code*
  - *Functional testing: testing how individual pieces of code interact*
  - *Integration testing: testing the whole system*
- Three environments: development, testing, and production
- Database Support: Oracle, DB2, SQL Server, MySQL, PostgreSQL, SQLite
- Action Mailer
- Action Web Service
- Prototype for AJAX

Rails Environment and Installing the Software

- Rails will run on many different Web servers. Most of your development will be done using WEBrick, but you'll probably want to run production code on one of the alternative servers.
  - Apache, Lighttpd (Lighty),Mongrel
- Development Environment
  - Windows, Linux and OS X
  - No IDE needed although there a few available like Eclipse, RadRails
- Installing Ruby for Windows
  - Download the “One-Click Ruby Installer from http://rubyinstaller.rubyforge.org
- Installing Ruby for Mac
  - It’s already there!
- RubyGems is a package manager that provides a standard format for distributing Ruby programs and libraries
- Installing Rails
  - >gem install rails –include-dependencies

Rails Tutorial

- Create the Rails Application
  - Execute the script that creates a new Web application project

  >Rails projectname
  - This command executes an already provided Rails script that creates the entire Web application directory structure and necessary configuration files.

Rails Application Directory Structure

- **App>** contains the core of the application
  - */models> Contains the models, which encapsulate application business logic*
  - */views/layouts> Contains master templates for each controller*
  - */views/controllername> Contains templates for controller actions*
  - */helpers> Contains helpers, which you can write to provide more functionality to templates.*
- **Config>** contains application configuration, plus per-environment configurability - contains the database.yml file which provides details of the database to be used with the application
- **Db>** contains a snapshot of the database schema and migrations
- **Log>** application specific logs, contains a log for each environment
- **Public>** contains all static files, such as images, javascripts, and style sheets
- **Script>** contains Rails utility commands
- **Test>** contains test fixtures and code
- **Vendor>** contains add-in modules.

Hello Rails!

- Need a controller and a view

  >ruby script/generate controller Greeting
- Edit app/controllers/greeting_controller.rb
- Add an index method to your controller class

```
class GreetingController < ApplicationController
  def index
    render :text => "<h1>Hello Rails World!</h1>"
  end
end
```

  - Renders the content that will be returned to the browser as the response body.
- Start the WEBrick server

  >ruby script/server

  http://localhost:3000

Hello Rails!

- Add another method to the controller

```
def hello
end
```

- Add a template app/views/greeting>hello.rhtml

```
<html>
  <head>
    <title>Hello Rails World!</title>
  </head>
  <body>
    <h1>Hello from the Rails View!</h1>
  </body>
</html>
```

Hello Rails!

- ERb - Embedded Ruby. Embedding the Ruby programming language into HTML document. An erb file ends with *.rhtml file extension.*
  - Similar to ASP, JSP and PHP, requires an interpreter to execute and replace it with designated HTML code and content.

- Making it Dynamic

  ** Date/Time: <%= Time.now %> **

- Making it Better by using an instance variable to the controller

  @time = Time.now.to_s
  - Reference it in .rhtml **<%= @time %>**

- Linking Pages using the helper method link_to()

  ** Time to say <%= link_to "Goodbye!", :action => "goodbye" %>**

Building a Simple Event Calendar

- Generate the Model (need a database and table)
- Generate the Application
- Configure Database Access
- Create the Scaffolding
- Build the User Interface
- Include a Calendar Helper
- Export to iCal (iCalendar is a standard for calendar data exchange)

Create the Event Calendar Database

- Create a Database
  - Naming Conventions
    - *Tables should have names that are English plurals. For example, the people database holds person objects.*
- Use object identifiers named id.
    - *Foreign keys should be named object_id. In Active Record, a row named person_id would point to a row in the people database.*
    - *ID*
    - *Created_on*
    - *Updated_on*
  - Edit the config/database.yml
- Create a Model
- Create a Controller

Rails Scaffolding

- Scaffold
  - Building blocks of web based database application
  - A Rails scaffold is an auto generated framework for manipulating a model
- CRUD Create, Read, Update, Delete

  >ruby script/generate model Event

  >ruby script/generate controller Event

  - *Instantiate scaffolding by inserting*
  - ***scaffold :event*** *into the EventController*
  - *The resulting CRUD controllers and view templates were created on the fly and not visible for inspection*

Rails Scaffolding

- Generating Scaffolding Code
- Usage: script/generate scaffold ModelName [ControllerName] [action,…]

  >ruby script/generate scaffold Event Admin

  – Generates both the model and the controller, plus it creates scaffold code and view templates for all CRUD operations.

  – This allows you to see the scaffold code and modify it to meet your needs.

    - ***Index***
    - ***List***
    - ***Show***
    - ***New***
    - ***Create***
    - ***Edit***
    - ***Update***
    - ***Destroy***

Create the User Interface

- While functional, these templates are barely usable and only intended to provide you with a starting point
  - RHTML files use embedded Ruby mixed with HTML ERb
    - *<% …%> # executes the Ruby code*
    - *<%= … %> # executes the Ruby code and displays the result*
  - Helpers
  - Layouts
  - Partials
  - Error Messages
  - CSS
  - AJAX

- Form Helpers
  - Start, submit, and end forms

User Interface with Style

- Layouts

  /standard.rhtml

  Add **layout "layouts/standard"** to controller

- Partials

  /_header.rhtml

  /_footer.rhtml

- Stylesheets
  - Publis/stylesheets/*.css

  - *NOTE: The old notation for rendering the view from a layout was to expose the magic @content_for_layout instance variable. The preferred notation now is to use **yield***

Enhance the Model

- Enhancing the Model
  - The model is where all the data-related rules are stored
  - Including data validation and relational integrity.
  - This means you can define a rule once and Rails will automatically apply them wherever the data is accessed.
- Validations - Creating Data Validation Rules in the Model

  validates_presence_of :name

  validates_uniqueness_of :name

  validates_length_of :name :maximum =>10
- Add another Model
- Migrations
  - Rake migrate

Create the Relationship

- Assigning a Category to an Event
  - Add a field to Event table to hold the category id for each event
  - Provide a drop-down list of categories
- Create Relationship
  - Edit models\event.rb

```
Class Event < ActiveRecord::Base
  belongs_to :category
end
```

  - Edit models\category.rb

```
Class Category < ActiveRecord::Base
  has_many :events
end
```

  - An Event belongs to a single category and that a category can be attached to many events

Rails Relationships

- Model Relations
  - Has_one => One to One relationship
  - Belongs_to => Many to One relationship (Many)
  - Has_many => Many to One relationship (One)
  - Has_and_belongs_to_many =>Many to Many relationships

Associate Categories to Events

- Edit Event Action and Template to assign categories

```
def new
  @event = Event.new
  @categories = Category.find(:all)
end

def edit
  @event = Event.find(@params[:id])
  @categories = Category.find(:all)
end
```

- Edit _form.rhtml

```
<%= select "event", "category_id", @categories.collect {|c| [c.name, c.id]} %>
```

Calendar Helper

- This calendar helper allows you to draw a databound calendar with fine-grained CSS formatting without introducing any new requirements for your models, routing schemes, etc.
- **Installation:**
  - script/plugin install http://topfunky.net/svn/plugins/calendar_helper/
  - List plugins in the specified repository:

    plugin list --source=http://dev.rubyonrails.com/svn/rails/plugins/
  - To copy the CSS files, use

    >ruby script/generate calendar_styles
- **Usage:**

```
<%= stylesheet_link_tag 'calendar/blue/style' %>
<%= calendar(:year => Time.now.year, :month => Time.now.month) %>
```

iCalendar and Rails

- gem install icalendar
- Add a method to generate the event:

```
require 'icalendar'
class IcalController < ApplicationController
 def iCalEvent
     @myevent = Event.find(params[:id])
     @cal = Icalendar::Calendar.new
      event = Icalendar::Event.new
      event.start = @myevent.starts_at.strftime("%Y%m%dT%H%M%S")
      event.end = @myevent.ends_at.strftime("%Y%m%dT%H%M%S")
      event.summary = @myevent.name
      event.description = @myevent.description
      event.location = @myevent.location
      @cal.add event
      @cal.publish
      headers['Content-Type'] = "text/calendar; charset=UTF-8"
   render_without_layout :text => @cal.to_ical
  end
end
```

RSS and Rails

- Create a new feed controller

```
def rssfeed
  conditions = ['MONTH(starts_at) = ?', Time.now.month]
  @events = Event.find(:all, :conditions => conditions, :order =>
   "starts_at", :limit =>15)
  @headers["Content-Type"] = "application/rss+xml"
 end
```

- Create a rssfeed.rxml view

- Add a link tag to standard.rhtml

```
<%= auto_discovery_link_tag(:rss, {:controller => 'feed', :action =>
   'rssfeed'}) %>
```

AJAX and Rails

- Add javascript include to standard.rhtml

  <%= javascript_include_tag :defaults %>

- Add to Event Controller

  auto_complete_for :event, :location

- Form Helper

  <%= text_field_with_auto_complete :event, :location%>

Summary

- Rail's two guiding principles:
  - Less software (Don't Repeat Yourself - DRY)
  - Convention over Configuration (Write code not configuration files)
- High Productivity and Reduced Development Time
  - How long did it take?
  - How many lines of code?

    >rake stats
  - Don't forget documentation…

    >rake appdoc

- Our experience so far.

- Unless you try to do something beyond what you have already mastered, you will never grow. – Ralph Waldo Emerson

Rails Resources

- Books
  - Agile Web Development with Rails –Thomas/DHH
  - Programming Ruby - Thomas
- Web sites
  - Ruby Language

    http://www.ruby-lang.org/en/
  - Ruby on Rails

    http://www.rubyonrails.org/
  - Rails API
    - *Start the Gem Documentation Server*

      Gem_server http://localhost:8808
  - Top Tutorials

    http://www.digitalmediaminute.com/article/1816/top-ruby-on-rails-tutorials
  - MVC architectural paradigm

    http://en.wikipedia.org/wiki/Model-view-controller

    http://java.sun.com/blueprints/patterns/MVC-detailed.html

David Heinemeier Hansson (Ruby on Rails creator) explained, "Once you've tried developing a substantial application in Java or PHP or C# or whatever," he says, "the difference in Rails will be readily apparent. You gotta feel the hurt before you can appreciate the cure."