Rails Overview

Ryan Timmons

April 14, 2008

Preliminaries

• Recommended resources:
  – “Agile Web Development with Rails” (Thomas/Hansson)
  – AJAX Version of Rails API: [http://railsbrain.com](http://railsbrain.com)

• Rails Dev
  – Mac/Linux: Built in (update using `sudo gem update rails` first though!)
  – Windows: InstantRails (includes MySQL server)
  – IDE: Aptana Studio (with RadRails plugin) has Ruby, etc. byte-compiled in Java built-in

Rolling With Rails

```
rails _2.0.2_ blogr
cd blogr
```

Rails Directory Structure

Contents of the new `blogr` directory:

```
app/   | public/
   |   controllers/
   |   images/
   |   helpers/
   |   javascripts/
   |   models/
   |   stylesheets/
   |   views/
   |   Rakefile
config/ | README
   |   database.yml  svn:ignore
   |   script/
   |   console
   |   generate
   |   plugin
   |   server
   |   test/
   |   tmp/  svn:ignore *
   |   vendor/
```
Rails Conventions

- **Model**, Story (SINGULAR) → blogr/app/models/story.rb
- **Controller**, StoriesController (PLURAL) → blogr/app/controllers/stories_controller.rb
- **Views**, blogr/app/views/stories/action.html.erb

Rails does a lot of “nice” string and variable renaming operations for you. If you have a model Person, for example, it will look for the PeopleController because the plural of “person” is “people.” Pluralizations are configured via Rails’s Inflections.

Rails uses **reflection** a lot, especially when dealing with the View.

Database Conventions

- Exactly one lower-cased plural table per Object: even polymorphic objects use the same table (but with an added type (string) field)
- id (integer) is primary key
- foreign key (used in the case of “belongs to”) is object_id, e.g., the stories table will have author_id to reference the Author to whom the Story belongs
- Some plugins, e.g., Acts as List, Acts as Tree, Acts as Nested Set, Acts as Finite State Machine, etc., will use their own conventional column names (e.g., Acts as Tree gives the model a parent_id field).

Entity Relationships

Our blog application has three objects: Authors, Stories, and Categories. Rails would say

- An Author “has many” Stories,
- A Story “belongs to” an Author,
- A Story “has and belongs to many” Categories, and
- A Category “has and belongs to many” Stories.
This looks like

and results in the following Database schema:

Convention is to put table alphabetically in join tables (e.g., `categories_stories` not `stories_categories`). Order of fields within the table doesn’t matter.

Model/Controller/View ("MVC")

---

Figure 2.1: The Model-View-Controller Architecture
Generators, Migrations, and Scaffolding

When using Rails 2.0.x, use the following to produce a scaffold for an Object. This creates, using Rails’s *generators*, the Object’s Model, Controller, and View files/folders and provides some basic common functionality (e.g., the ability to list/view/add/edit model instances). It also generates a Migration to create the necessary tables in the database.

```ruby
ruby ./script/generate scaffold Story \
  title:string \
  text:text \
  created_at:datetime \
  updated_at:datetime \
  author_id:integer
```

Note that id is implicit. The *scaffold* generator expects the model name to be singular. **Things go crazy if you don’t use the correct pluralization, especially in the generators.**

Generators do not automatically assume the Object relationships. You also need to create a separate migration manually for the *has_and_belong_to_many* relationships.

Other data types:

- **string** (corresponds to CHAR/VARCHAR types in SQL),
- **text** (corresponds to TEXT type in SQL),
- **binary** (BLOB type in SQL—probably don’t want this)
- **integer, decimal**
- **boolean**
- **date, datetime, and time**

The migration at *blogr/db/migrate/001_create_story.rb* with some sample data added in:

```ruby
class CreateStories < ActiveRecord::Migration
  # this is how to perform this database change
  def self.up
    create_table :stories do |t|
      t.string :title
      t.text :text
      t.datetime :created_at
      t.datetime :updated_at
      t.integer :author_id
      t.timestamps # gives you created_at and updated_at
    end
    # (Can also use Fixtures for adding sample data.)
    Story.create(:title => "The Titanic sank today.")
    Story.create(:title => "Wait, um, that was a while ago.")
  end
  # this is how to "undo" this database migration
  def self.down
    drop_table :stories
  end
end
```

Use `rake db:migrate` to migrate your database to the most current migration.

Can run `ruby ./script/server` to set up a local server running your Rails project at *http://localhost:3000.* Run `ruby ./script/console` to interact with your app interactively.
Sample Model

class Author < ActiveRecord::Base
  has_many :stories # now have access to @author.stories in controller
  validates_presence_of :fname, message: "Must enter first name!"
  validates_presence_of :lname # can also validate with regexp
    # or with Proc blocks for
  # Automatically have "for free" # complex/conditional
  # Author.find_by_field("Data") # validations.
  # for arbitrary field (e.g., Author.find_by_title("some title"))
  # as well as crazy things like even
  # Author.find_or_create_by_fname_and_lname("Oscar","Wilde")

  def name # virtual property getter
    self.fname + " " + self.lname
  end

  def name=(str) # virtual property setter
    self.fname, self.lname = str.split(" ")
  end
end

Sample Controllers

class StoriesController < ApplicationController
  # [...] def new # page submits to the create action
  # populate a list of categories for, e.g., a dropdown
  @categories = Category.find(:all)
  # instance variables (@) are available in the view
  @story = Story.new # creates the model in memory but not in db
  respond_to do |format|
    format.html # new.html.erb
    format.xml { render :xml => @story }
  end end
class CategoriesController < ApplicationController
  def find_by_title_like
    title = params[:title] # params object gets user-submitted data
    @category = Category.find( :first,
      :conditions => [ 'title LIKE \'%?\%\', title ])
    # ? is like %s in C’s printf
  end end

View Misc.

- Actions get rendered in a layout. You can change this by calling the render method in the controller. Rails assumes you want blogr/views/layouts/stories.html.erb as the layout for your Story model. You can create blogr/views/layouts/application.html.erb and delete model-specific layouts if they aren’t necessary.
- the h method used often in views such as <%=h str %> simply escapes HTML characters (e.g., < ⇒ &lt;)
- Partialis allow you to refactor views
Useful Plugins


- **ActiveScaffold** gives you instant/live AJAX-enabled views of your db. Great for instantly setting up an admin interface.
- **will_paginate** (used to be part of Rails 1.x) allows you to easily paginate your controllers/views
- **acts_as_versioned** automatically keeps track of old versions of a model instance in the database every time it’s changed and gives you access to, e.g., `@author.versions`.

Upcoming Web Workshops


- **HTML Day 2** Covers HTML tables and forms. Assumes basic HTML knowledge
  
  *Thu. 17 April, 5:30-7:30pm, OUGL 102*

- **CSS Day 1** Covers basic selectors/properties for fonts and colors. Assumes knowledge of HTML.
  
  *Tue. 22 April, 5:30-7:30pm, OUGL 102*

- **CSS Day 2** Covers advanced selectors and properties for page layout
  
  *Thu. 24 April, 5:30-7:30pm, OUGL 102*

- **Web Site Creation Day** Covers the “best” of HTML, CSS, and Adobe ImageReady.
  
  *Sat. 19 April 9:00am-5:00pm, MGH 030*

- **JavaScript** Covers the jQuery JavaScript+AJAX library. Assumes knowledge of HTML, CSS, and Computer Programming but little assumption of prior JavaScript experience. 2-Parts.
  
  *Tue. 13 May and Thu. 15 May, 5:30-7:30pm, MGH 058*