My First Three Weeks on Rails

Aaron Mulder
Chariot Solutions

Background

- J2EE developer
- Mac/Linux platform
- Have used dynamic languages before (mostly Perl, some P/Jython)
- Never used Ruby
- Suddenly found myself helping out on a real Rails project!

Let’s get something out of the way right now...

- Documentation motto: “Grr.... Arrrgh!”
- Dear RDoc, why can’t you be like JavaDoc?
- If you combine the book and the RDoc and the online documentation and Google, you get nearly 80% of what you need
- Try “ri [ClassName]” if you really want to be annoyed, or http://api.rubyonrails.com/ and http://www.ruby-doc.org/

Tools & Setup

Gem & Rails

- Start on rails 0.14.3 (easy upgrades)
  - Though older version has more explicit scripts and configuration files
- Try “gem list”, “gem list rails”, “gem update rails”
- gem will actually store more than one copy of the package (“gem uninstall -v 2.3 foo”)

MySQL

- Any installation process is OK for MySQL
  - But MySQL 4.1 is safest
- Ruby native MySQL drivers are bad (esp. if client or server is OS X, etc.)
- Install the C drivers, it’s quick and easy and your Rails config/code doesn’t change

Source Control

- Can put entire rails generated directory under source control
- May need to register new extensions (yml, rhtml, rb)
- Main problem is that I keep accidentally checking in config/database.yml -- I want some way to leave it out of source control but generate it on the first build...

Build Scripts

- Ha ha ha...
- Actually a little annoying not to be able to do certain things during a “build”
- Can put code in config/environment.rb?
- Use “rake” for certain setup tasks -- can add own rake tasks (more on this later)
- Use “rake --tasks” to see available tasks

IDE

- Kate on Linux has the **best** Ruby syntax highlighting!
- Need to have lots of files at your fingertips
- Editor with embedded tree view is nice, otherwise a nearby Finder/Explorer/Konq
- Eclipse environment seems to be evolving rapidly (I haven’t tried the latest)

Misc

- DB Browser
  - I still use DBVisualizer
- Graphics
- HTML/CSS Editor
- ...

Database Stuff

Instances

- Ruby builds-in support for development, test, production databases/environments (including “init” scripts...)
- Can leverage this in your own batch/tool/client code by setting appropriate vars
- All tests run against test by default, everything else against development. Production has “special properties”.

Scripts

- Must have column definitions handy for every table! I refer to the scripts a lot, but a DB browser is handy too
- Rake can copy your dev database structure (tables, keys, etc.) to test, but you have to keep dev up to date and remember to run the Rake task before testing
- No need to have test data scripts...

Fixtures

- Rails uses test data in “fixtures” (named hashes, where each has the data for a row)
- Make sure every entry uses a unique name!
- Fixtures don’t work automatically with foreign keys (gee, who would use those?)
- Very handy for immortalizing test data

A Fixture

test/fixtures/user.yml:

```
test_user:
  id: 1
  username: aaron
  password: secret
  first_name: Aaron
  last_name: Mulder
  created_at: 2005-11-16
  updated_at: <%= Time.now.strftime(
                        "%Y-%m-%d %H:%M:%S") %>
another:
  id: 2
  ...
```

A Custom Rake Task

lib/tasks/load_my_fixtures.rake:

```
desc "Load fixtures in correct order"
task :load_my_fixtures => :environment do
  require 'active_record/fixtures'
  ActiveRecord::Base.establish_connection(
                                  RAILS_ENV.to_sym)
  tables = ["parent","child","three","four",...]
  Fixtures.create_fixtures('test/fixtures', tables)
end
```

Typical DB Procedures

- mysql ... < db/drop_tables.sql
- mysql ... < db/create_tables.sql
- rake load_my_fixtures
- script/server

- rake clone_structure_to_test
- ruby test/unit/some_test.rb

DDL

- Columns named “type” will cause problems
- If you don’t name your foreign keys and have problems, try “show create table table_name” in MySQL
- Ideal to give every table an “id” primary key (set to auto_increment)
- Rails prefers one table with a type and many extra fields to “inheritance” tables

Ruby Is Not Java

Some Differences

- ClassPath
- Imports
- Main
- Interfaces
- Mixins / Mult. Inheritance / Op. Overloading
- Unspecified fields/methods

Class Path & Import

- There are certain default search locations
- Try putting one of these in environment.rb:

```
ADDITIONAL_LOAD_PATHS.each { |file| puts "#{file}" }

config.load_paths.each { |file| puts "#{file}" }
```

- For child dirs off any of those, use:

```
require 'batch/upload_job'      # for upload_job.rb
```

- Require takes a *filesystem case* String

main(args/argc,argv)

- There is none
- Whatever Ruby file you run can have statements in the file but outside of the classes it contains, and those will be run when the file is run
- There's a global variable ARGV for the args
- But, "ruby foo.rb" doesn't always load rails...

script/runner

- script/runner does actually load rails (and you can give it a DB environment too) and then executes whatever you pass it
  - Try script/runner “require ‘foo’”
- Might want to alter script/runner to do “ARGV.shift” to take the execution command out of the argument list

Interfaces

- I don’t know what you can do about this?
- I wanted a server connectivity class with a real back end and a mock back end for when the server is not available
- I have to use my eyeballs to make sure they have the same methods/params?
- Dear gurus, any suggestions?

Mixins

- People talk about using this for multiple inheritance, but I don’t think of it that way
- I use it to access utility functions that I couldn’t get otherwise (date helpers for a controller, when they’re in a view class)
- At the top of the controller file:

```
include ActionView::Helpers::DateHelper
```

Unspecified Properties

- The model objects get methods for all the database columns
- But that's not visible anywhere in the source code
- When creating a new model object, what fields do you have to set?
- A lot of time spent referring to create SQL

Give Me Type Safety...

- Some of the most frustrating errors for me were where I ended up with the wrong object in my variable
- Try debugging with: puts "#{foo.type}"
- Why can't it auto-convert String for math?
- Errors may be caused by incorrect method arguments (more on this in a moment...)

Coding Stuff

Models: Find vs. Relate

- In some cases, can put pretty much the same thing in a relationship as you could in a finder (SQL, criteria, ordering...)
- Why not just declare a method that uses a find call under the covers?
- Relationships add multiple methods, etc.
- Find can do parameters/substitution

Learning from Rails

- Notice how all the special Rails stuff takes a Hash as an argument?

```
belongs_to :parent, :class_name =>
"ParentType", :foreign_key => "parent_id"
```

- That's because the method would have 12 arguments and it would be impossible to consistently get them in the right order
- Not a bad idea... Not bad at all... :)

Speaking of belongs_to

- Relationship specifiers are a little goofy
- belongs_to vs has_one -- works alright for "child" tables but not "subclass" tables
- What properties do you use if the foreign key column name doesn't match the remote table name? (e.g. parent_id points to a table not named "parents")

Relationship Example

```
belongs_to :parent, :class_name =>
"MyParent", :foreign_key => "parent_id"
```

- The first bit (parent) is the name of the property you'll use to access/navigate this
- The next bit (class_name) is the type of object on the other end
- The last bit (foreign_key) is the name of the foreign key column

That Evil 'Type'

- Object.type is the type (class) of an object
- Rails somehow uses type to manage a class hierarchy based on rows in a single table with different type codes
- Now your table has a type field too?!?
- Try providing manual accessor methods with a different name:

Avoiding Type Conflicts

```
class Event < ActiveRecord::Base
  has_one :playout_event
  has_one :switch_event
  has_one :overlay_event

  # "type" attribute is defined by Object!
  def event_type
    read_attribute("type")
  end

  def event_type=(type)
    write_attribute("type", type)
  end
end
```

Die, Middle Tier, Die!

- There's no reason to put business logic in a standalone object or in the controller
- Just put it in the model! (often as a class method)
- Event.schedule(...) and Event.cancel() rather than EventManager.scheduleEvent(...) and EventManager.cancelEvent(eventID)

Controllers

- A view (.rhtml) has access to any instance variables of the controller (extends it??)
- There doesn’t seem to be anything like request.setAttribute(“foo”, bar);
- May end up with a number of instance variables, only some of which are valid for any given view... But at least this works a little better in a dynamic language.

Error Handling

- Seems pretty convenient -- if a model produces a validation error it can pop right into specific messages on the view
- Not as clear how to do it if the form doesn’t directly correspond to a model object

View Cleverness

- The rails “form tags” are pretty good at reading data from a model object to populate a form, and reading data from a form submission into a hash for you
- Then you can use `model.update_attributes (@params[:hash_name])` to copy the changed data into the model -- nice!

View Weirdness

- I want to have a small form on a page that doesn’t quite correspond to a model object
- Reading out of the hash after a form submission works well!
- There’s no way to pre-populate the form (e.g. by sending the view a hash instead of an object that contains the default values) -- need to use a non-Rails input widget

Speaking of Hashes

- Don’t be try to use Strings as Hash keys -- it may or may not work depending on how the Hash was populated
- Bad: `foo[“key”] = value`   `bar = foo[“key”]`
- Good: `foo[:key] = value`   `bar = foo[:key]`
- Perhaps somebody else can explain this... :)

Utility Weridness

- Some classes that are not associated with model objects don’t automatically reload when changed
- Need to stop and start WebBrick
- There’s got to be some way around this! I feel like.... a J2EE developer! :)

Testing

- Built in unit tests (create a model, query for models, etc.)
- Built in functional tests (call controller methods with fake data as if submitted from web form)
- “rake test_units” & “rake test_functional”
- Also “ruby test/unit/foo.rb” to run just one

Test Example

```
require File.dirname(__FILE__) + '/../test_helper'

class EventTest < Test::Unit::TestCase
  fixtures :parent, :event, :child, ...

  def setup
    @event = Event.find(1)
  end

  def test_something
    assert ...
  end
end
```

More Issues...

- Some data seems cached?
- No obvious way to bulk insert data?
- Date/time manipulations are a little painful
  - Cool stuff, just scattered & incomplete
  - Time, DateHelper, Rails Number utils
  - if event.date + 6.hours < 3.days.ago ....

Properties & Initializers

```
class MyObject
  attr_reader :id, :name

  def init(id, name)
    @id = id
    @name = name
  end
end

obj = MyObject.new(1, “Hello”)
obj.name = “Goodbye #{obj.id}”
```

Method Call Parens

- It seems that it’s best to use them unless it’s quite obvious that they’re not necessary
- Makes it clearer what total expression a certain bit of logic applies to
- Makes it clear what’s a method argument on a return line, vs a separate value

```
return do_something “foo”, “bar” if baz
```

Summary

Final Thoughts

- A little more solid documentation would go a long way
- Biggest pain point is not knowing what properties a model class has
  - Next: getting method args correct
- Can develop plumbing very quickly; mostly it comes down to writing business logic and the UI (all the better to AJAX you with!)

Discussion / Q&A