Development with Ruby-on-Rails

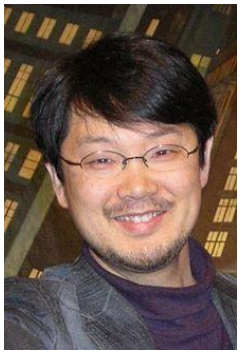
April 4th, 2013
SENG 403
Tyler Gibb
Alberto Saavedra
Justin Milanovic
Garrick Van Der Lee

Agenda

- Background
- Principles of Ruby-on-Rails
- Model-View-Control Architecture Pattern
- Agile Development with Rails
- Pitfalls of Rails
- Live Demo

Background Ruby

Ruby: Ruby was developed By Yukihiro Matsumoto in 1995. It was developed because he wanted a language that was as powerful as the Perl scripting language but was object oriented like Python.



Background Rails

Rails: The rails gem was developed by David Hansson in 2003 as part of a project he was working on called Basecamp. He extracted the code and released it as the rails framework in 2004.

Model-View-Control

Model:

- Responsible for maintaining the state of the application
- Used to represent forms of data used by the application
- Contains logic to manipulate and retrieve data
- Enforces all of the business rules that apply to the data

Model-View-Control

View:

- Responsible for generating the user interface.
- Usually this interface is based on the data in the model.
- Presents the user with various ways of inputting data.
- Never handles incoming data
- Once the data is displayed, the View's work is done.
- There can be many views that access the same model data, usually for different purposes.

Model-View-Control

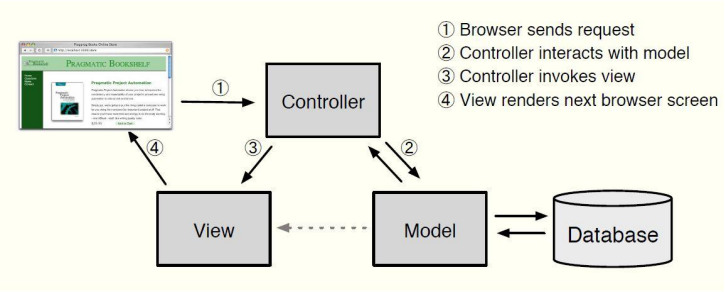
Control:
- Provides the logic that binds together models and views.
- Coordinates the interaction between user and views
- Process Input
- Calls methods made available by models
- Delivers data to the views

Model-View-Control

Benefits of MVC:
- Code will become easier to work with and the application becomes easier to extend and maintain.
- Ability to have multiple views that rely on the same data
- Could switch the database from MySQL to Oracle, only the model would need editing
- Scaffolding

Litte bit of history:
1979, Trygve Reenskaug came up with MVC

MVC



Testing With Ruby Rails

- Rails was designed with testing in mind
- When a new project is made, Rails automatically generates a folder specifically for testing
- When new projects are created, three databases are made, a test database, development database and production database, this will allow for tests to be done without effecting the data in the production database

Test Database

- This database is specifically used for unit, functional and integration testing
- When running tests, the fixtures in the, fixture directory will migrate into the test database when the tests are being executed

Development database

- Used to test the software without making modifications to the production database
-rake db:create will create a new development database
-rake db:migrate will create the schema for the database
-rake db:seed will seed thje database with values predetermined for testing the code
-This helps with continuous integration.

Production Database

- This database is used for a working project
- Obviously not used for testing
- Benefits from having the other two database to run test on

Testing With Ruby on Rails

- the test database is generated with test data stored in fixtures
- Rails automatically loads the fixtures into the test database when implementing tests

Unit/Functional Testing

- When a new model is generated in rails, templates for unit and functional tests will automatically be generated in their corresponding folders
- Rails does not automatically implement test code for you it simply creates a template for your tests
- There is a large library of assertions that rails provides for your tests

Integration Testing

- Integration Testing is completely implemented by the user within the integration folder
- the integration test folder is generated when a new rails project is generated

Running tests

- Once all tests have been correctly set, a developer simply needs to use "rake test" and all unit,functional and integration tests will automatically run

Agile Development

-Ruby on rails was developed for agile development and achieves this from
-Transparency and
-Customer Collaboration

Agile Development: Transparency

- When the code is changed, everyone involved in the project should automatically see the changes being made
-This is often done using a continuous integration host such as github
-The code being implemented is working code

Agile Development: Customer Collaboration

- Due to transparency customers can see the project as it develops and give more timely feedback
-Seeing the project develop will also likely build the trust of customers

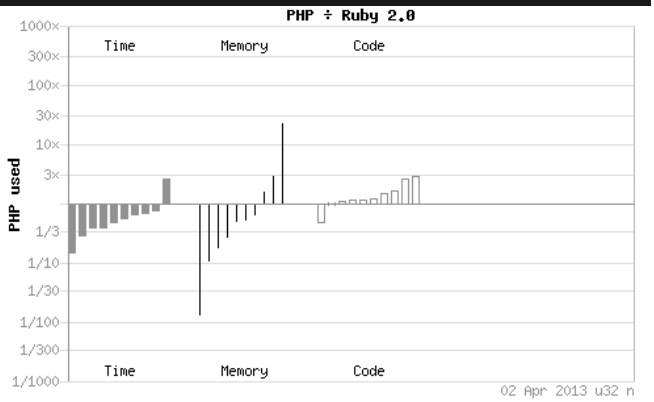
Pitfall #1

SLOW!!

-Ten times slower than Java and two times slower than PHP
-Several benchmark websites that run test and response times of different programming languages describe Ruby as one of the slower languages.
-Twitter, after two years of high profile scaling problems, replaced Ruby on Rails as the front end for their search engine with a Java based solution, which purportedly executes queries three times faster and dramatically reduced CPU utilization.

PHP ÷ Ruby 2.0

http://benchmarksgame.alioth.debian.org



Pitfall #2

Learning Curve!

-Because it integrates all components, Ruby on Rails requires that all developers have some knowledge of a high level language (Ruby), front-end technologies (HTML, CSS, Javascript), and an understanding of relational databases (MySQL).
-Ruby itself is a very minimalist language, making it hard to figure out what is going on in a given block of code; sacrificing readability for “tightness.”
-With Rails strict “convention over configuration” mantra, everything must be done in a specific way making it difficult to separate the MVC layers. Sufficed to say, it can be difficult to find competent Ruby on Rails developers.

Coding Example

-Rails is thought to be easy for development because many things in the framework have been automated; this does not make it easy to understand.

Here's a bit of an extreme example:

```
(0...8).map{65.+(rand(25)).chr}.join
```

It's very simple actually - generate an 8 character random string.

Pitfall #3

Lack of Documentation

-Most of the documentation is scattered around the Internet on blog sites, with very little consolidation.
-There are a lot of online video tutorials; only problem being that once you deviate from the specific project being shown it is very easy to quickly get lost and confused when trying to apply a modification.
-Constant, rapid changes of the framework combined with Ruby on Rails complexity itself that requires enormous amounts of peripheral knowledge, making it difficult for developers to catch up with the latest changes.

Pitfall #4

Error Messages

-Compared to PHP, Rails is also unfriendly when it comes to errors. With PHP, typically a page will render including the part with the error, showing which line the error occurred and a message that is helpful. In Rails, typically the whole app blows up!

ExecJS::RuntimeError in Active_admin/devise/sessions#new

Showing */usr/lib/ruby/gems/1.9.1/gems/activeadmin-0.5.1/app/views/layouts/active_admin_logged_out.html.erb* where line **#12** raised:

```
活興•讚瀚欏明歧湘港凍殘凍   浴   碳腕瀧      仰□提坐□璃伏儀□瀧•勿掬晚敦□□
  (in /usr/lib/ruby/gems/1.9.1/gems/activeadmin-0.5.1/app/assets/javascripts/active_admin/lib/namespace.js.coffee)
```

Pitfalls #5

Framework development is community driven

-Ruby on Rails is developed by its community. The downside of this is that there is no formal institution that takes the responsibility for any changes introduced into the framework.
-Rails is a young framework that has been changing rapidly over the past few years.
-No long term roadmap of updates and there are no guarantees when it comes to the support.
-Ruby on Rails has been rife with a tremendous amount of deprecation between versions; making it hard to keep up with the changes and find reliable information sources.

Pitfall #6

Active Record

-Active Record insulates the programmer from the need to use SQL in most instances, but this comes at the cost of a foreign key constraints and support for triggers and stored procedures.
-It makes atomic operations tricky. If a group of objects must be saved in an all-or-nothing fashion either one object must know about all these other objects and control their persistence or the control over the entire transaction must be handled from outside the domain

Pitfall #7

Applications typically not suitable for Ruby on Rails development

-Blogs: better solutions have been developed specifically for this purpose such as WordPress
-Content Management Systems: better alternatives such as Drupal or Joomla
-Brochure Pages: A simple site with information about your company should be built with lighter technologies; a simple combo of HTML and CSS.

Pitfall #7, continued

Generic e-commerce applications: It is less complex and less expensive to build a typical e-commerce service using such tools as Shopify.

Highly secure financial applications: Financial transactions often rely on procedures stored in the software environment and databases which are independent from the web interface. Ruby on Rails is optimized for simple CRUD operations rather than for enforcing complex transactional rules.

Live Demo

Brief Introduction to Ruby
Rails App

Brief Introduction to Ruby

Ruby Installation

- Compiling Ruby - Source Code
- Use RVM (Ruby Version Manager) for Mac OS X, Linux, or any UNIX-like operating system:
  `$ \curl -L https://get.rvm.io | bash -s stable --ruby`
- RubyInstaller for Windows

Interactive Ruby Shell (irb)

```
2.0.0-p0 :018 > message = "Hello class!"
2.0.0-p0 :019 > puts message
 Hello class!
```

Brief Introduction to Ruby
(continued)

Comments

```
# This whole line is a comment
x = 100  # x is assigned the value 100, and comment is ignored.
```

Variables

```
@myResult = 5 #instance variable, scope confined to whatever
object # self refers to.
x = 4 # local variable, only available within a block of code
```

Arrays

```
> myArray =  Array.new() => []
> myArray =  Array.new(3) => [nil, nil, nil]
> myArray =  Array.new(3,true) => [true, true, true]
> myArray = [1,2,3,4] => [1, 2, 3, 4]
```

Brief Introduction to Ruby
(continued)

Hashes

```
> hash = {'one' => 1} => {"one"=>1}
> hash['one'] => 1
```

Symbols

```
:courses # symbols are immutable
myName = :jim
myName :jim
```

Closures

```
> def twice
>   yield
>   yield
>   end
```

```
> name = 'class'
> twice { puts "Hello #{name}" }
Hello class
Hello class
```

Brief Introduction to Ruby
(continued)

Classes and Methods

```
> class MyTest
>   def set_foo(n)
>     @foo = n
>   end
> end
> i = MyTest.new()
 => #<MyTest:0x000000028f7a30>
 i.set_foo(100)
 => 100
> i
 => #<MyTest:0x000000028f7a30 @foo=100>
```

Ruby-on-Rails App

Installation

- RubyGems (standard Ruby package manager)
  `gem install rails`

PostgreSQL

Bootstrap

heroku

http://sheltered-taiga-3895.herokuapp.com/products

[redacted]

Questions?