Instant Websites using Ruby on Rails

Tyler Kovacs
tyler.kovacs@zvents.com

CoolTech Club
Jan 25, 2005

Tonight's Discussion

- What is Ruby On Rails?
- Why would you use Ruby on Rails?
- What class of applications is is appropriate for?
- How does it differ from other approaches?

Zvents

- Concept to market 6 months/one programmer
- No performance issues
- Highly satisfied
- We’re hiring!

What is Ruby On Rails?

- What is Ruby?
  - Programming language
  - “successful combination of SmallTalk’s conceptual elegance, Python’s ease of use and learning, and Perls’ pragmatism”
    - Curtis Hibbs
- What is Ruby on Rails
  - A framework for developing web applications
  - Written in Ruby

Why Ruby?

- Interpreted, dynamic, flexibly typed language
- Easy to learn and maintain
- Single Inheritance + Mixins
- “Duck Typing”
- Language stays out of the way

Duck Typing

- “If it walks like a duck, and quacks like a duck, it’s duck”
- If an object responds to a message, it’s of an appropriate type.
- Method calls are viewed as messages
- Undefined methods can be handled by the object

Inheritance/Extending

- Single Inheritance
- Can mixin modules
  - Effectively multiple inheritance
- You can add methods to any class
  - String, Date, Thread
  - This is not extending the class, this modifies the original class for everyone!

Why Not Ruby?

- Performance
  - Similar to Perl and Python
  - This is not an issue for many websites.
- Threading model
  - Does not use native threads

Ruby on Rails

- Extremely productive web application framework developed by David Heinemeier Hansson
- MVC (Model / View / Controller)
- Open-source
- Low learning curve
  - A working app in literally minutes
- Version 1.0

Framework Means…

- Rails builds a skeleton, and you flesh it out.
- All layers designed to work together
- Many decisions are made for you.
  - Convention over Configuration
  - This is a good thing for new applications, but can be a bit of a burden if you need to adapt to legacy systems.

Convention over Configuration

- Manual configuration replaced with convention and reflection
  - Your data schema and code is the configuration
- Less configuration files, no compilation – changes take effect immediately (in development – restart required to pick up some changes in production)

What does Rails Do?

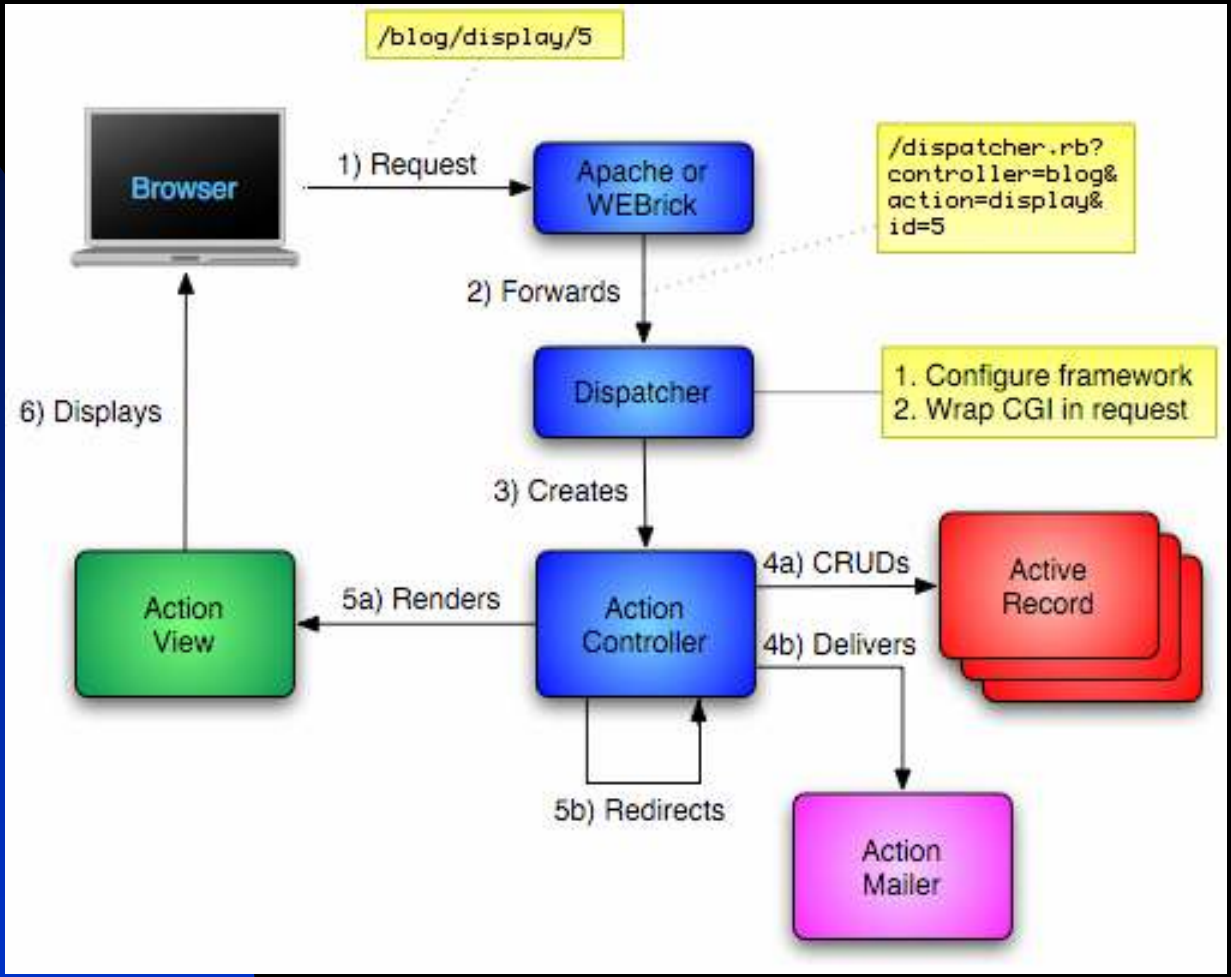
- It covers almost all of what you need to do for a typical web app, from creating the app, through deployment.

Some of what Rails does

- Builds The MVC Application Skeleton
- Code Generation
- ORM
- UI
  - Templates
  - Javascript/Ajax Helpers
- Logging
- Deployment
- Testing/Benchmarking, etc.

MVC Application

- Rails generates a nicely laid out MVC application
- Saves wasting time on deciding how to build the app.
- Generates code/scaffolding for Models/Views/Controllers


/blog/display/5
Browser
1) Request
Apache or WEBrick
/dispatcher.rb?
controller=blog&
action=display&
id=5
2) Forwards
Dispatcher
1. Configure framework
2. Wrap CGI in request
6) Displays
3) Creates
Action View
5a) Renders
Action Controller
4a) CRUDs
Active Record
4b) Delivers
5b) Redirects
Action Mailer

Getting started

- Generate an application skeleton with one command:

```
$rails ctc
```

```
create  app/controllers
create  app/helpers
create  app/models
create  app/views/layouts

…
```

- Start the web server (Rails comes with one bundled)

```
$ ruby script/server
```

Getting started

- Connect with your browser http://localhost:3000/
- Create a 'Hello World' page:

```
$ rm public/index.html
$ ruby script/generate controller welcome
$ vi app/views/welcome/index.html
```

- Make the welcome page the default route

```
$ vi config/routes
```

- Set up your databases

```
$ vi config/database.yml
```

Rails Environments

- Rails has 3 environments
  - Development
  - Test
  - Production
- Always running within the context of one of these environments
- Determines database
- Affects behavior of framework (e.g., mail delivery and caching disabled in test, etc.)

Database/ORM

- ActiveRecord
  - Dynamically mapped from the database
  - Maps in-memory Ruby objects to persistent database store
  - Unlike hibernate, schema definition is not duplicated in code.

Creating a Model Object

```
$ ruby script/generate model Commercial
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/commercial.rb
      create  test/unit/user_test.rb
      create  test/fixtures/commercials.yml
```

What do we get?

- Empty class or surprising functionality?

```
class Commercial < ActiveRecord::Base
end
```

- Automatically mapped to users table using pluralization rules

```
CREATE TABLE commercials (
  id INT(32) UNSIGNED NOT NULL AUTO_INCREMENT,
  firstname CHAR(64)  DEFAULT “”,
  lastname CHAR(64)  DEFAULT “”,
  PRIMARY KEY(id)
);
```

```
$ ruby script/console
> c = Commercial.new
=> #<Commercial:0x392d660
   @attributes={“id"=>nil, “firstname"=>"“,
   “lastname"=>""}, @new_record=true>
>> quit
```

- script/console allows you to access Rails objects from the command line
- Same mechanism can be used to scripts within the Rails environment

Extending Model Objects

- Add methods to model class

```
class Commercial < ActiveRecord::Base
    def full_name
      return self.firstname + " " + self.lastname
    end
end
```

- Model methods determine how you (programmer) interact with model objects

Assocations

- Define the relationship between objects

```
class Commercial < ActiveRecord::Base
  has_many :comments
end

class Comment < ActiveRecord::Base
  belongs_to :commercial
end
```

Assocations

- Adds methods to access associated objects

```
commercial = Commercial.find(id)
commercial.comments.each{|c|
  puts c.name
}
```

- Can model one-to-one, many-to-one and many-to-many relationships

Validations

- Specify constraints on the object
- Object can't be saved if not valid
- Built-in validations
  :validates_presence_of
  :validates_length_of
  :validates_uniqueness_of
  :validates_format_of
  …
- Define your own named validations
- Arbitrary code in validation method

Callbacks

- Automatically execute code on object lifecycle events (create, validate, save, update, destroy)
- Before and After hooks

  before_save

  after_save

  before_destroy

  after_destroy

  …

Callbacks

```
class Commercial < ActiveRecord::Base
    def before_create
            self.created_at ||= Time.now
    end

    def before_update
            self.updated_at ||= Time.now
    end
end
```

Transaction Support

- Database transaction support

```
Account.transaction do
        david.withdrawal(100)
        mary.deposit(100)
end
```

- Database and model object transactions

```
 Account.transaction(david,mary) do
     david.withdrawal(100)
     mary.deposit(100)
 end
```

Finding Objects

- Find by id

  Commercial.find(id)

- Dynamic finders

  Commercial.find_by_name(“monster.com”)

  Commercial.find_all_by_name(“monster.com”)

- Find By SQL

  Commercial.find_by_sql(“SELECT * FROM commercials WHERE name = ‘monster.com’”)

Finding Objects

- Complex finders with eager loading

```
name = some_dynamic_value
Commercial.find(:all, :conditions => [“name = ?”,
    name], :include => :comments, :order =>
    “created_at DESC”)
```

- Finds can be scoped to an association

```
c = Commercial.find(1)
c.comments.find(:all, :conditions =>
    [“created_at > ?”, Time.now – 1.day])
```

Find works on associations too

- Same finder can be used to search for objects scoped to a specific association

```
c = Commercial.find(1)
c.comments.find(:all, :conditions =>
    [“created_at > ?”, Time.now – 1.day])
```

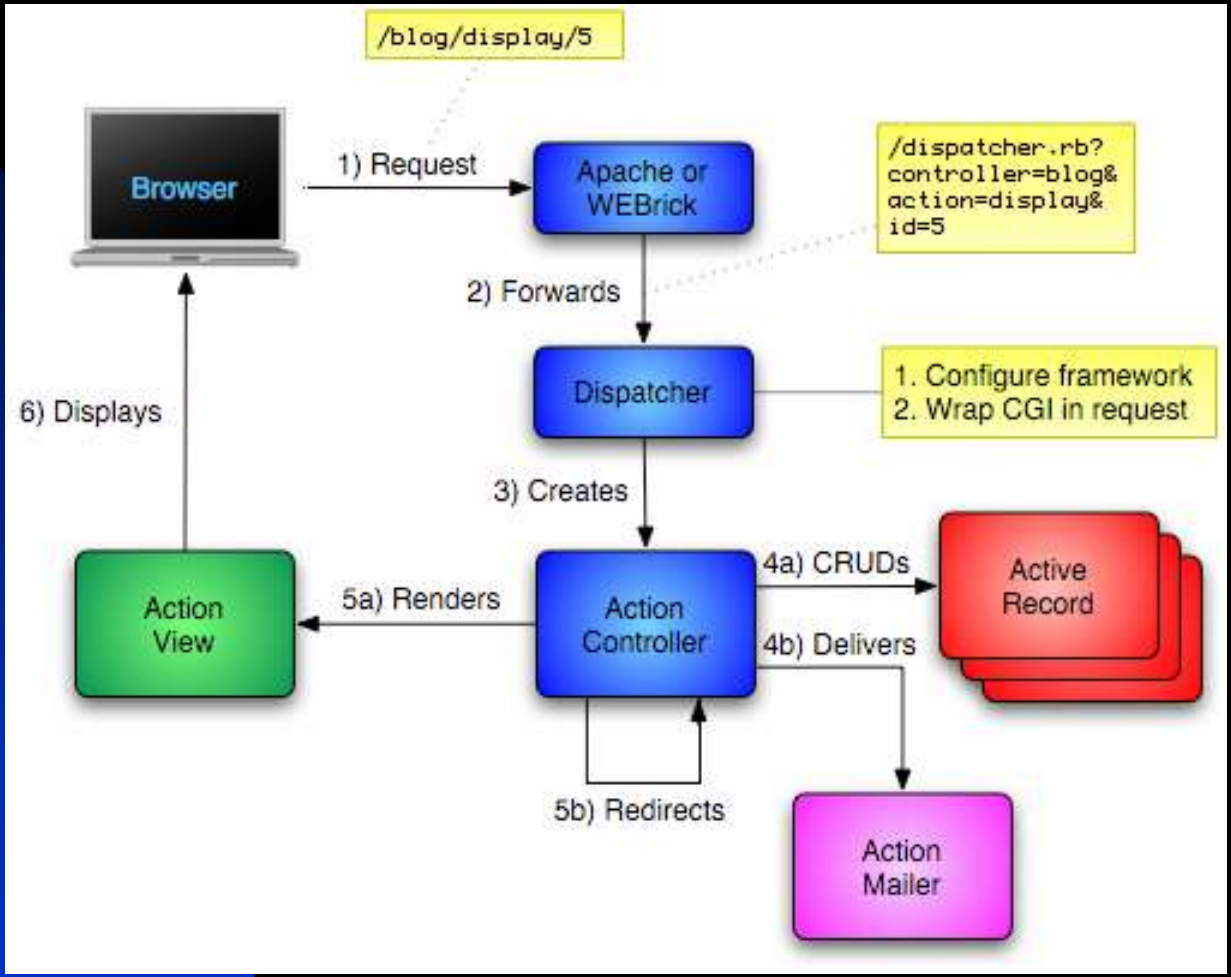
Controller

- Handles incoming requests to web application
- May interact with model object(s) if necessary to satisfy this request
- Renders a view which is sent back as a response
- Controller methods define how a user interacts with application

Controller

```
class CommercialController < ActionController::Base
    def show
        @commercial = Commercial.find(@params['id'])
    end

    def delete
        @commercial = Commercial.find(@params['id'])
        @commercial.destroy
    end
end
```


/blog/display/5
Browser
1) Request
Apache or WEBrick
/dispatcher.rb?
controller=blog&
action=display&
id=5
2) Forwards
Dispatcher
1. Configure framework
2. Wrap CGI in request
6) Displays
3) Creates
Action View
5a) Renders
Action Controller
4a) CRUDs
Active Record
4b) Delivers
5b) Redirects
Action Mailer

Creating Controller and View

- `ruby script/generate scaffold Meeting`
- Builds a controller and several views
- Basic CRUD Application

Views

- Generates HTML/XML/Email that is sent back to in response to a request
- RHTML templates very similar is concept to ASP and JSP
- Helpers make Javascript and AJAX easy to use
- Rails comes bundled with prototype.js and script.aculo.us libraries

Views

app/views/commercial/show.rhtml:

```
<h1>Commercial: <%= @commercial.name %></h1>
<% for comment in @commercial.comments %>
    <%= comment.text %><br/>
<% end %>
```

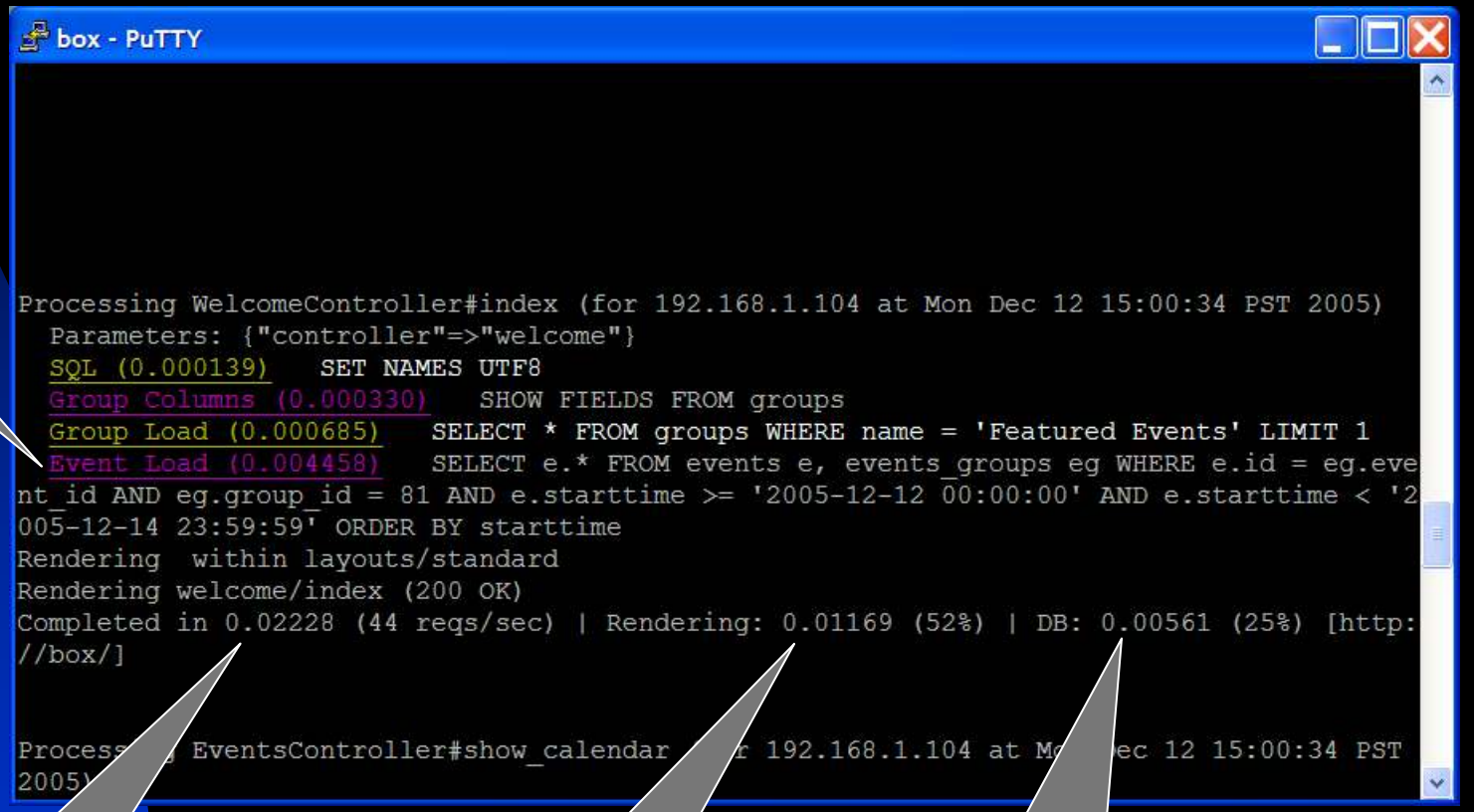
- Where's the rest of the page with the <HTML> and <BODY> tags?
  - Layouts wrap the view

Logging

- Rails logs provide a wealth of information out of the box
- Log levels adjusted depending on current Rails Environment

Log File

Execution times for all SQL.



Total page time

Render time

Total DB time

Breakpoints

- Breakpoints can be inserted anywhere in your code path
- Connect to the breakpoint by running:

```
$ ruby script/breakpointer
```

Testing

- Rails includes test framework
- Rails generates test code skeleton
- Unit and functional tests
- Simulated HTTP
  - get :index, post :update_password
  - assert_response :success
- Fixtures
- Mock Objects

Mock Objects

- Mock objects can be very easy, using Ruby

```
require 'models/purchase_order'
class PurchaseOrder
  def accept
    return true
  end
end
```

Use the real object

Override only the methods you need to.

Simple Code Statistics

- rake stats
- Delivers simple code stats including:
  - LOC
  - Class and Method counts
  - Test to code ratio

Web Services

- Built-in mechanism to deliver SOAP and XML/RPC web services
- Basically as easy as writing a controller
- Roll your own REST API using XML builder templates instead of the standard RHTML templates

Automated Deployment

- Switchtower is Rails' Deployment Tool
- Deploys code directly out of your repository
- Database migrations
- Transactional
- Can roll back changes, if necessary

Is that it?

- There's MUCH more, but we don't have time for it tonight:
  - Cookies and session management
  - Caching
  - Pagination
  - Form helpers
  - ActionMailer
  - AJAX helpers
  - Etc.

Why Use Ruby on Rails?

- Agile
  - Fast Time to Market
  - Quick changes
  - Quick to Learn
- Complete solution
  - Don’t have to worry about
    - What will I do for logging, deployment, testing….

Why Not Use Rails?

- CPU speed
  - Ruby is currently not a fast language
  - Neither was Java when it started
  - Bottleneck in many web apps is database or IO bandwidth
  - If your app is CPU bound Ruby may not be the right choice
- If your app is not a good fit with the framework

Documentation

- Buy The Books

- There is much less good online documentation than there is with Java.

Try it yourself!

- Download and install
  - http://rubyonrails.org/
- Watch the videos
  - http://rubyonrails.org/screencasts

Reminder

- Did I mention that we're hiring?