Effective Ruby on Rails Development Using CodeGear's Ruby IDE

Shelby Sanders
Principal Engineer
CodeGear

Web Application Expectations

Dynamic

- Static web pages and images are not enough

Interactive

- Rich-client functionality is becoming the norm

Persistent

- Data and preferences integrated throughout

Flexible

- Retarget content to mobile devices, feeds, web services

Ruby Background

Object-Oriented

- Everything is an Object
- Everything is extensible

```ruby
"abc".slice(1, 2)

1.2.finite?()

96.modulo(7)

true.instance_of?(TrueClass)

nil.nil?()
```

```ruby
class Object
    def print_greeting
        puts "Hello World!"
    end
end
96.print_greeting()


module Helper
    def do_work
        puts "..."
    end
end
message = "abc"
message.extend(Helper)
message.do_work()
```


Ruby Background

"Duck" Typing (Dynamic Typing)

- If something walks like a duck and quacks like a duck, then it can be treated as if it is a duck

```ruby
number = 96
puts number.modulo(7)


count = 10
count.times do
    puts "."
end


letters = ["a","b","c"]
letters.each {|i| puts "#{i}" }
```

Ruby Background

Blocks (Closures)

- Arbitrary sections of code which can be passed around while retaining original context

```ruby
block1 = lambda { puts "." }
block2 = lambda do
    1 + 1
end
10.times { block1.call }
puts block2.call

def perform(items, &action)
    items.each do |item|
        yield(item)
    end
end
letters = ["a","b","c"]
perform(letters) { |letter| puts letter }
```


Ruby Background

Modules

- Collections of functionality which can be attached to Classes or Objects
- Equivalent of Interfaces, Multiple Inheritance, and Aspects

```ruby
module Logging
    def log(message)
        puts message
    end
end

module Validation
    def valid?()
        self.validate.nil?
    end
end
```

```ruby
class Item
    include Logging
    include Validation
    def validate()
        log("Validating Item...")
        nil
    end
end
i = Item.new()
i.valid?
```

Ruby Background

Method Missing Handler

- Allows dynamic instrumentation of functionality at runtime

```ruby
module LogMissing
    def method_missing(method_id, *args)
        arity = !args.nil? && args.length > 0 ? args.length.to_s : ''
        puts "Attempt to call #{self.class.name}.#{method_id.id2name}(#{arity})"
    end
end

class Item
    include LogMissing
end
i = Item.new
i.missing() # Results in "Attempt to call Item.missing()"
i.testing(1, 2, 3) # Results in "Attempt to call Item.testing(3)"
```


Ruby Background

Lenient Syntax

- Most punctuation is optional, including parentheses around method parameters and semicolons as line endings
- Enables natural creation of a Domain Specific Language (DSL) for the task at hand

```ruby
var1 = "abc";
var2 = "abc"
index = var1.index "b"


var1.slice(1, 2)
var1.slice 1, 2
var1.slice 1, index


10.times do
    puts "."
end
10.times { puts "." }
```

Rails Background

Extracted from Reality

- All features have been used in multiple production applications before being added to the framework
- Features are evolved based on the experience of the development community
- Deprecation is used to support legacy applications, while allowing the framework and community to move forward

Rails Background

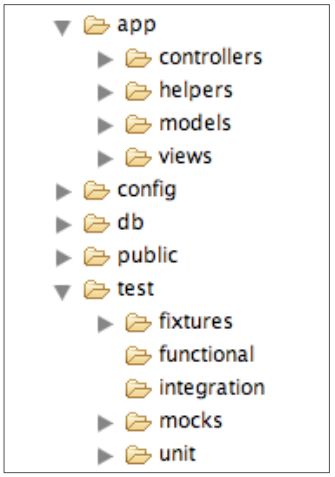
Opinionated Software

- Designed around the most common scenarios, instead of trying to be everything for everyone
- Chooses a default configuration which satisfies the most common scenarios
- Enforces best practices by default, but allows deviation when necessary
- For instance, the Model-View-Controller pattern is naturally enforced by the application generators, layout, and lifecycle

Rails Background

Convention over Configuration

- Every Rails application has the same layout on disk
- All common components have predefined locations
- Accepting the default layout and configuration, enables most routing and mapping to be done automatically



Rails Background

Don't Repeat Yourself (DRY)

- All configuration and logic should be defined once
- For instance, database schema and structure of model classes doesn't overlap, instead they depend on each other to define the whole picture

```ruby
class Post < ActiveRecord::Base

  has_many :comments


  validates_presence_of :title
  validates_uniqueness_of :title

  validates_presence_of :content

end
```

```ruby
class CreatePosts < ActiveRecord::Migration
  def self.up
    create_table :posts do |t|
      t.column :title, :string
      t.column :content, :text
    end
  end

  def self.down
    drop_table :posts
  end
end
```

Rails Background

Full Stack Framework

- All common functionality for database-driven web applications
  - Object/Relational mapping
  - Model-View-Controller support
  - AJAX libraries and helpers
  - Web services support
  - Embedded development server
  - Test management
- No need to evaluate, learn, and maintain dependency on multiple third-party frameworks

Rails Development

Component Generation

- All major components are generated via built-in or easily acquired third-party plugins, including:
  - Models
  - Migrations
  - Controllers
  - Scaffolds (Views)
  - Web services
  - Tests
  - Plugins

```
>script/generate model Comment author:string ...
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/comment.rb
      create  test/unit/comment_test.rb
      create  test/fixtures/comments.yml
```

Scaffolding

- Provides default implementation of Create, Read, Update, and Delete (CRUD) Views
- Provided via dynamic instrumentation or static code generation
- Supports incremental replacement at the Action granularity

```
1  class PostsController < ApplicationController
2
3    scaffold :post
4
5  end
```

```
>script/generate scaffold Post Post
      exists  app/controllers/
      exists  app/helpers/
      exists  app/views/post
      exists  app/views/layouts/
      exists  test/functional/
  dependency  model
      exists    app/models/
      exists    test/unit/
```


Change Recognition

- Automatic recognition of artifact changes and update of application at runtime
- The following changes are integrated without a server restart:
  - Creation of application components
  - Modification of database schema including relationships
  - Changes in Model, View, and Controller logic
- Routing changes are the exception

Rails Development

"Automagic" Object/Relational Mapping

- All Model schema, relationship, and validation information is defined only once
- Most schema information is defined using database-agnostic syntax
- Special cases are defined using the same artifacts

```ruby
class Post < ActiveRecord::Base

  has_many :comments


  validates_presence_of :title
  validates_uniqueness_of :title

  validates_presence_of :content

end
```

```ruby
class CreatePosts < ActiveRecord::Migration
  def self.up
    create_table :posts do |t|
      t.column :title, :string
      t.column :content, :text
    end
  end

  def self.down
    drop_table :posts
  end
end
```


Rails Development

"Automagic" Object/Relational Mapping (Continued)

- Most model client API is dynamically instrumented at runtime

```ruby
def test_post_client
  @post = Post.new(:title => 'Testing 1, 2, 3',
                   :content => 'This is a test.',
                   :views => 0,
                   :created_at => Time.now)
  @post.save

  @current_title = @post.title
  @post.content = 'New Content'

  @posts = Post.find(:all)
  @post = Post.find_by_id(3)
  @post = Post.find_by_title('Hello World!')

  @comment = @post.comments.first
  @parent_post = @comment.post
end
```

Context Propagation

- Request and session data are easily transformed into database objects
- Appropriate contextual objects defined in Controller are automatically propagated to Views

```
class PostsController < ApplicationController
    def show
      @post = Post.find(params[:id])
    end
end
```

```
<% for column in Post.content_columns %>
<p>
  <b><%= column.human_name %>:</b> <%=h @post.send(column.name) %>
</p>
<% end %>

<%= link_to 'Edit', :action => 'edit', :id => @post %> |
<%= link_to 'Back', :action => 'list' %>
```


Code Generation Support

- Defined via server-side Ruby code
- Generates client-consumed JavaScript, XML, etc

```
page.select("div#notice" ).each { |div| div.hide }

page.replace_html("post" , :partial => "post" , :object => @post)

page[:post].visual_effect :blind_down if @post.comments.length > 0
```

```
xml.posts do
    for post in @posts
        xml.post do
            xml.title(post.title)
            xml.content(post.content)
            xml.views(post.views)
            xml.created_at(post.created_at)
        end
    end
end
```

Testing Support

- Integrated generation of functional, integration, and unit tests when creating application components
- Automation and reusability via mocks, stubs, and fixtures
- Leverages standard Ruby testing framework (Test::Unit)
- Helper functionality for common web application assertions

```
one:
  id: 1
  title: Title1
  content: Content1
  views: 1
two:
  id: 2
  title: Title2
  content: Content2
  views: 2
```

```
def test_blog_index
    get "/blog/index"
    assert_response :success
    assert_template "index"
end
```


Web Services

- Built-in support for REST, SOAP, and XML-RPC
- Automatic dispatch and marshalling, just like any other view

```
class BlogInfoApi < ActionWebService::API::Base
  api_method :find_all_posts
  api_method :find_comments_by_post
end
```

```
class BlogInfoController < ApplicationController
  wsdl_service_name 'BlogInfo'

  def find_all_posts
    @posts = Post.find(:all)
  end

  def find_comments_by_post
    @post = Post.find(params[:id])
    @comments = @post.comments
  end
end
```

Product Overview

Initiation

- Complete runtime environment provided during install
  - Includes Ruby, Rails, gems, plugins, and databases
- Wizards
  - Provide logical creation of application components via composite code generation
- Cheat sheets and instructional videos
  - Teach usage of Rails and IDE at the same time

Product Overview

Code Generation and Manipulation

- Integrated command line environment
  - Provides natural access to Rails helper scripts
  - Enhanced by contextual information from IDE
- Ruby and Rails specific code completion and refactoring
  - Leverages type information based on framework semantics and conventions
  - Includes information for dynamically instrumented functionality

Product Overview

Archaeology

- Analysis of artifact dependencies regardless of location and format
- Visualization of inbound and outbound dependencies of the current selection

Navigation

- Actions enabling seamless movement between collaborating artifacts
- Correlation of URLs to associated actions and vice-versa

Product Overview

Running

- Embedded development server runs application in place
- Embedded Mozilla web browser enables realistic inspection of DOM, request lifecycle, and contextual information

Debugging

- Multiple language support including Ruby and JavaScript
- Integrated display of Rails-specific contextual information

References

Release Information

- Available in the second half of 2007
- Windows, Mac OS X, RedHat Linux
- Pricing has not yet been set

Product Site

- http://www.codegear.com/products/rubyide

Beta Test Signup

- Contact CodeGear staff!