

Why is Ruby such a good language for Rails?

Keith Pitty

keith@keithpitty.org

<http://keithpitty.org>

Sydney Rails Meetup, 15 August 2007

About Keith

About Keith

- 24 years experience since graduating with B.Sc.

About Keith

- 24 years experience since graduating with B.Sc.
- Earlier years on mainframes

About Keith

- 24 years experience since graduating with B.Sc.
- Earlier years on mainframes
- First OO programming experience in Smalltalk

About Keith

- 24 years experience since graduating with B.Sc.
- Earlier years on mainframes
- First OO programming experience in Smalltalk
- Employed by a Java consultancy since 2000

About Keith

- 24 years experience since graduating with B.Sc.
- Earlier years on mainframes
- First OO programming experience in Smalltalk
- Employed by a Java consultancy since 2000
- Fascinated by Ruby and Rails since 2004

About Keith

- 24 years experience since graduating with B.Sc.
- Earlier years on mainframes
- First OO programming experience in Smalltalk
- Employed by a Java consultancy since 2000
- Fascinated by Ruby and Rails since 2004
- Life long Cricket tragic

Why ask the Question?

Why ask the Question?

- One lunch-time in May 2006 I presented a "Rails Experience Report" to my colleagues at Cirrus Technologies

Why ask the Question?

- One lunch-time in May 2006 I presented a "Rails Experience Report" to my colleagues at Cirrus Technologies
- My boss was curious about Ruby's role in the Rails framework

Why ask the Question?

- One lunch-time in May 2006 I presented a "Rails Experience Report" to my colleagues at Cirrus Technologies
- My boss was curious about Ruby's role in the Rails framework
- "Is there something special about Ruby?"

Why ask the Question?

- One lunch-time in May 2006 I presented a "Rails Experience Report" to my colleagues at Cirrus Technologies
- My boss was curious about Ruby's role in the Rails framework
- "Is there something special about Ruby?"
- At the time, I didn't have a good answer

So, why is Ruby such a good language
for the Rails framework?

Let's hear an opinion or two...

"Ruby is such an incredibly rich and expressive language that it's hard to appreciate its beauty by simply relating it to past experiences with other languages."

“Tasks that would have made my eyes roll in PHP or Java made my smile light up as Ruby time and time again showed that programming could be simple, fun, and outright beautiful.”

David Heinemeier Hansson
in his foreword to
"Ruby for Rails"
by David A. Black

Inspiring words,
but let's dig a little deeper...

Question One

How does Ruby enable `ActiveRecord` to infer the attribute names of a model object?

ActiveRecord Inference

ActiveRecord Inference

Suppose we have a model class...

ActiveRecord Inference

Suppose we have a model class...

```
class Programmer < ActiveRecord::Base
  has_many :skills

  def full_name
    first_name + " " + last_name
  end
end
```

15 August, 2007

Sydney Rails Meetup: "Why is Ruby such a good language for Rails?"

Keith Pitty

Based on the table...

Based on the table...

```
create table programmers (  
  id integer not null,  
  first_name varchar(255) not null,  
  last_name varchar(255),  
  favourite_language varchar(255),  
  primary key (id)  
);
```

15 August, 2007

Sydney Rails Meetup: "Why is Ruby such a good language for Rails?"

Keith Pitty

And now we code...

And now we code...

```
joe = Programmer.new  
joe.first_name = "Joe"  
joe.last_name = "Bloggs"  
joe.favourite_language = "Ruby"  
joe.save
```

And now we code...

```
joe = Programmer.new  
joe.first_name = "Joe"  
joe.last_name = "Bloggs"  
joe.favourite_language = "Ruby"  
joe.save
```

How does Ruby enable Rails to infer the attribute names?

15 August, 2007

Sydney Rails Meetup: "Why is Ruby such a good language for Rails?"

Keith Pitty

Let's look at the source code..

Let's look at the source code..

For `ActiveRecord::Base`, it's in
`activerecord/lib/active_record/base.rb`

Let's look at the source code..

For `ActiveRecord::Base`, it's in
`activerecord/lib/active_record/base.rb`

Let's focus on a couple of methods...

```
def initialize(attributes = nil)
  @attributes = attributes_from_column_definition
  @new_record = true
  ensure_proper_type
  self.attributes = attributes unless attributes.nil?
  yield self if block_given?
end
```

```
def initialize(attributes = nil)
  @attributes = attributes_from_column_definition
  @new_record = true
  ensure_proper_type
  self.attributes = attributes unless attributes.nil?
  yield self if block_given?
end
```

```

def method_missing(method_id, *args, &block)
  method_name = method_id.to_s
  if @attributes.include?(method_name) or
      (md = /\?$/ .match(method_name) and
       @attributes.include?(query_method_name = md.pre_match) and
       method_name = query_method_name)
    define_read_methods if self.class.read_methods.empty? &&
      self.class.generate_read_methods
      md ? query_attribute(method_name) : read_attribute(method_name)
  elsif self.class.primary_key.to_s == method_name
    id
  elsif md = self.class.match_attribute_method?(method_name)
    attribute_name, method_type = md.pre_match, md.to_s
    if @attributes.include?(attribute_name)
      __send__("attribute#{method_type}", attribute_name, *args, &block)
    else
      super
    end
  else
    super
  end
end
end

```

```
def method_missing(method_id, *args, &block)
  method_name = method_id.to_s
  if @attributes.include?(method_name) or
    (md = /\?$/ .match(method_name) and
     @attributes.include?(query_method_name = md.pre_match) and
     method_name = query_method_name)
    define_read_methods if self.class.read_methods.empty? &&
      self.class.generate_read_methods
    md ? query_attribute(method_name) : read_attribute(method_name)
  elsif self.class.primary_key.to_s == method_name
    id
  elsif md = self.class.match_attribute_method?(method_name)
    attribute_name, method_type = md.pre_match, md.to_s
    if @attributes.include?(attribute_name)
      __send__("attribute#{method_type}", attribute_name, *args, &block)
    else
      super
    end
  else
    super
  end
end
```

method_missing

method_missing

- Responds to arbitrary method calls

method_missing

- Responds to arbitrary method calls
- Dynamically handles missing methods

method_missing

- Responds to arbitrary method calls
- Dynamically handles missing methods
- As the name implies, "if the method is missing, I'll handle it"

method_missing

- Responds to arbitrary method calls
- Dynamically handles missing methods
- As the name implies, "if the method is missing, I'll handle it"
- An example of Ruby metaprogramming

__send__

- An alias for `send`
- `__send__ :symbol, *args`
- Invokes the method identified by the symbol, passing any arguments specified
- In our example, `__send_` invokes `attribute=`

So when we code:

So when we code:

```
joe = Programmer.new  
joe.first_name = "Joe"  
joe.last_name = "Bloggs"  
joe.favourite_language = "Ruby"
```

So when we code:

```
joe = Programmer.new
joe.first_name = "Joe"
joe.last_name = "Bloggs"
joe.favourite_language = "Ruby"
```

`method_missing` uses `__send__` to dynamically handle the setter methods for the `Programmer` object from the `@attributes` loaded in `initializer`

Also worth examining methods in
`active_record/attribute_methods.rb` such as:

Also worth examining methods in
`active_record/attribute_methods.rb` such as:

```
match_attribute_method?(method_name)  
attribute?(attribute_name)  
attribute=(attribute_name)
```

Also worth examining methods in
`active_record/attribute_methods.rb` such as:

```
match_attribute_method?(method_name)  
attribute?(attribute_name)  
attribute=(attribute_name)
```

as well as methods in `ActiveRecord::Base...`

Also worth examining methods in `active_record/attribute_methods.rb` such as:

```
match_attribute_method?(method_name)
attribute?(attribute_name)
attribute=(attribute_name)
```

as well as methods in `ActiveRecord::Base...`

```
define_read_methods
read_methods
generate_read_methods
query_attribute(attr_name)
read_attribute(attr_name)
```

Question Two

How does Ruby facilitate customisation of the `form_for` method in the `ActionView::Helpers::FormHelper` module?

Customising form_for

Customising form_for

Suppose we have a form...

Customising form_for

Suppose we have a form...

```
<% form_for(:contact, :url => contacts_path) do |f| %>
  <p><label>First Name <em class="required">(required)</em></label><br />
    <%= f.text_field :first_name %>
  </p>
  <p><label>Last Name <em class="required">(required)</em></label><br />
    <%= f.text_field :last_name %>
  </p>
  <p><label>Notes</label><br />
    <%= f.text_area :notes %>
  </p>
<% end %>
```

Customising form_for

Suppose we have a form...

```
<% form_for(:contact, :url => contacts_path) do |f| %>
  <p><label>First Name <em class="required">(required)</em></label><br />
    <%= f.text_field :first_name %>
  </p>
  <p><label>Last Name <em class="required">(required)</em></label><br />
    <%= f.text_field :last_name %>
  </p>
  <p><label>Notes</label><br />
    <%= f.text_area :notes %>
  </p>
<% end %>
```

We can improve on this...

tagged_form_for

tagged_form_for

```
<% tagged_form_for(:contact, :url => contacts_path) do |f| %>  
  <%= f.text_field :first_name, :required => true %>  
  <%= f.text_field :last_name, :required => true %>  
  <%= f.text_area :notes %>  
<% end %>
```

tagged_form_for

Look, now there are no HTML tags!

```
<% tagged_form_for(:contact, :url => contacts_path) do |f| %>
  <%= f.text_field :first_name, :required => true %>
  <%= f.text_field :last_name, :required => true %>
  <%= f.text_area :notes %>
<% end %>
```

tagged_form_for

Look, now there are no HTML tags!

```
<% tagged_form_for(:contact, :url => contacts_path) do |f| %>
  <%= f.text_field :first_name, :required => true %>
  <%= f.text_field :last_name, :required => true %>
  <%= f.text_area :notes %>
<% end %>
```

Needs some support in `application_helper.rb`

The Helper

The Helper

In module `ApplicationHelper`...

The Helper

In module `ApplicationHelper`...

```
def tagged_form_for(name, *args, &block)
  options = args.last.is_a?(Hash) ? args.pop : {}
  options = options.merge(:builder => TaggedBuilder)
  args = (args << options)
  form_for(name, *args, &block)
end
```

The Helper

In module `ApplicationHelper`...

```
def tagged_form_for(name, *args, &block)
  options = args.last.is_a?(Hash) ? args.pop : {}
  options = options.merge(:builder => TaggedBuilder)
  args = (args << options)
  form_for(name, *args, &block)
end
```

And the `TaggedBuilder` class...

```

class TaggedBuilder < ActionView::Helpers::FormBuilder

  def self.create_tagged_field(method_name)
    define_method(method_name) do |label, *args|
      @template.content_tag(:p, @template.content_tag(:label,
        label.to_s.humanize + required_tag(*args)) + "<br />" + super)
    end
  end

  field_helpers.each do |name|
    create_tagged_field(name)
  end

  def required_tag(*args)
    required?(*args) ? " " + @template.content_tag(:em, "(required)",
      :class => "required") : ""
  end

  def required?(*args)
    args.last != nil && args.last.is_a?(Hash)
    && args.last.has_key?(:required) && args.last.key?(:required)
  end
end

```

```
class TaggedBuilder < ActionView::Helpers::FormBuilder

  def self.create_tagged_field(method_name)
    define_method(method_name) do |label, *args|
      @template.content_tag(:p, @template.content_tag(:label,
        label.to_s.humanize + required_tag(*args)) + "<br />" + super)
    end
  end

  field_helpers.each do |name|
    create_tagged_field(name)
  end

  def required_tag(*args)
    required?(*args) ? " " + @template.content_tag(:em, "(required)",
      :class => "required") : ""
  end

  def required?(*args)
    args.last != nil && args.last.is_a?(Hash)
    && args.last.has_key?(:required) && args.last.key?(:required)
  end
end
```

define_method

define_method

- Dynamically defines a method

define_method

- Dynamically defines a method
- In this case the `field_helpers` are inherited from `ActionView::Helpers::FormBuilder`

define_method

- Dynamically defines a method
- In this case the `field_helpers` are inherited from `ActionView::Helpers::FormBuilder`
- `text_field` and `text_area` methods in `TaggedBuilder` are defined dynamically

define_method

- Dynamically defines a method
- In this case the `field_helpers` are inherited from `ActionView::Helpers::FormBuilder`
- `text_field` and `text_area` methods in `TaggedBuilder` are defined dynamically
- Another example of metaprogramming

Also worth examining methods in `FormHelper`
`action_view/helpers/form_helper.rb` such as:

Also worth examining methods in `FormHelper`
`action_view/helpers/form_helper.rb` such as:

```
form_for(object_name, *args, &proc)  
fields_for(object_name, *args, &proc)
```

Also worth examining methods in `FormHelper`
`action_view/helpers/form_helper.rb` such as:

```
form_for(object_name, *args, &proc)  
fields_for(object_name, *args, &proc)
```

as well as `InstanceTag` and `FormBuilder` classes...

Also worth examining methods in `FormHelper`
`action_view/helpers/form_helper.rb` such as:

```
form_for(object_name, *args, &proc)  
fields_for(object_name, *args, &proc)
```

as well as `InstanceTag` and `FormBuilder` classes...

and `form_tag` method in `FormTagHelper`

Question Three

What are some other examples of
Ruby **metaprogramming** within Rails?

Answer

Answer

As Tim Lucas advised me at Railscamp,

Answer

As Tim Lucas advised me at Railscamp,
"Look in the Rails source code."

What to look for

What to look for

- `module_eval`, `class_eval`, `instance_eval`

What to look for

- `module_eval`, `class_eval`, `instance_eval`
- Modifying the singleton class with `class <<`

What to look for

- `module_eval`, `class_eval`, `instance_eval`
- Modifying the singleton class with `class <<`
- Callable objects (`procs`, `lambdas`, `blocks`)

What to look for

- `module_eval`, `class_eval`, `instance_eval`
- Modifying the singleton class with `class <<`
- Callable objects (`procs`, `lambdas`, `blocks`)
- `self.included`, `self.inherited`

What to look for

- `module_eval`, `class_eval`, `instance_eval`
- Modifying the singleton class with `class <<`
- Callable objects (`procs`, `lambdas`, `blocks`)
- `self.included`, `self.inherited`
- `extend`

What to look for

- `module_eval`, `class_eval`, `instance_eval`
- Modifying the singleton class with `class <<`
- Callable objects (`procs`, `lambdas`, `blocks`)
- `self.included`, `self.inherited`
- `extend`
- extending Ruby's core functionality
(hint: see `ActiveSupport`)

Further Exploration

Further Exploration

- Max Muermann's "Metaprogramming Techniques" presentation

Further Exploration

- Max Muermann's "Metaprogramming Techniques" presentation
- "Ruby for Rails" by David A. Black, Chapter 13 - Ruby dynamics

Further Exploration

- Max Muermann's "Metaprogramming Techniques" presentation
- "Ruby for Rails" by David A. Black, Chapter 13 - Ruby dynamics
- "The Ruby Way" by Hal Fulton, Chapter 11 - OOP and Dynamic Features

Further Exploration

- Max Muermann's "Metaprogramming Techniques" presentation
- "Ruby for Rails" by David A. Black, Chapter 13 - Ruby dynamics
- "The Ruby Way" by Hal Fulton, Chapter 11 - OOP and Dynamic Features
- Why's "Poignant Guide to Ruby"

Further Exploration

- Max Muermann's "Metaprogramming Techniques" presentation
- "Ruby for Rails" by David A. Black, Chapter 13 - Ruby dynamics
- "The Ruby Way" by Hal Fulton, Chapter 11 - OOP and Dynamic Features
- Why's "Poignant Guide to Ruby"
- The Rails source code!

In Summary

In Summary

- Why is Ruby such a good language for Rails?

In Summary

- Why is Ruby such a good language for Rails?
- I think I have a better answer now

In Summary

- Why is Ruby such a good language for Rails?
- I think I have a better answer now
- There is something special about Ruby

In Summary

- Why is Ruby such a good language for Rails?
- I think I have a better answer now
- There is something special about Ruby
- It's the power of metaprogramming

Questions and Comments?

Thank you for listening!



Keith Pitty

keith@keithpitty.org

<http://keithpitty.org>