

Key Ruby Idioms for Rails

Andy Vanasse
PCSS, Inc

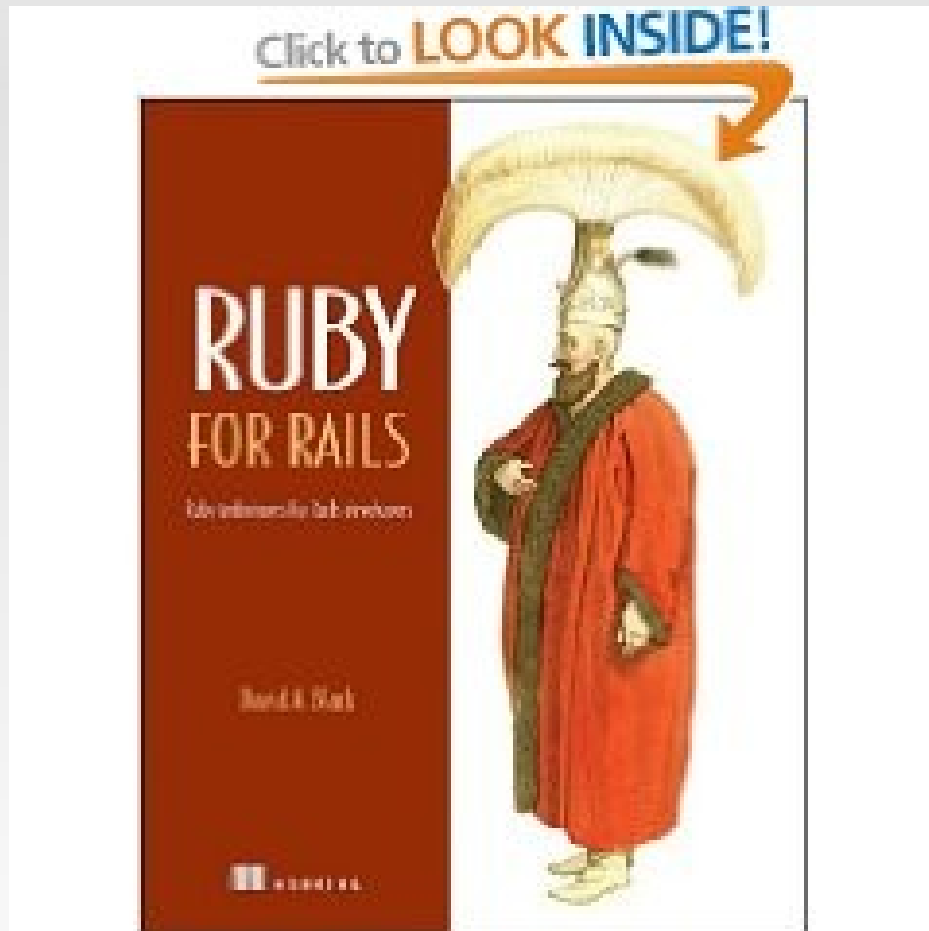
Key Ruby Idioms for Rails

Andy Vanasse
Koinonia Software
homeschoolsafe.com -- churchsafe.com

What?

- Ruby Object and Enumerable methods
- Some of "the way to do Ruby"
- Some code extracted from real projects
(with names changed to protect the innocent)

Why?



“An alarm went off in my head, therefore, when I saw how many budding Rails developers were asking themselves whether it was necessary to learn Ruby in order to use Rails. The fact that this question was the subject of disagreement and debate surprised me.”

- David Black

A Rose by Any Other Name...

- Generating your welcome message

```
msg = "Happy %s Birthday!" % age.to_s
```

```
msg = "Happy " + age.to_s + " Birthday!"
```

String interpolation: #{ }

- `msg = "Happy #{age} Birthday!"`

Why?

- Ruby is a *language* – it should be readable
- Most type-casting is done for you via inspect

Aside: Design a readable API

- Name methods for how they will appear *when used*

```
class Attendee
  def self.named(name)
    find_all_by_name_and_paid(name, true)
  end
end

# @badge_holder = Attendee.named "Andy Vanasse"
```

Every rose has a thorn...

- Object#respond_to?
 - Find out if an object supports a particular method
 - The foundation for “Duck typing” ~ “interfaces”
 - Process a group of objects from different classes united by functionality
 - Process objects that mix in functionality based on state

Example: Processing params

```
unless params[:case_cause].nil?  
  if params[:case_cause].respond_to?(:each)  
    params[:case_cause].each do |seq, attr|  
      kase.case_causes << CaseCause.new(attr)  
    end  
  else  
    kase.case_causes << CaseCause.new(params[:case_cause])  
  end  
end  
end
```

Sending out an S. O. S.

- Object#send
 - Ruby is an *object oriented* language
 - The essence of 'invoking' methods is sending messages
- Why?
 - Calculate method names
 - Iterate over method names

Ex: Iterating over method names

- I use this often with rspec (BDD)

```
[ :first_name, :last_name, :born_on ].each do |attr|  
  it "should require a #{attr}" do  
    @test_obj.send "#{attr}=", nil  
    @test_obj.should_not be_valid  
    @test_obj.should have(1).error_on(attr)  
  end  
end
```

Ex: User-driven processing

- User selects optional processing from checkboxes

```
def import
  messages = []
  params.keys.each do |processing_option|
    messages << self.send(processing_option)
  end
end

private
def process_1
  ...
end
```

alias_method_chain

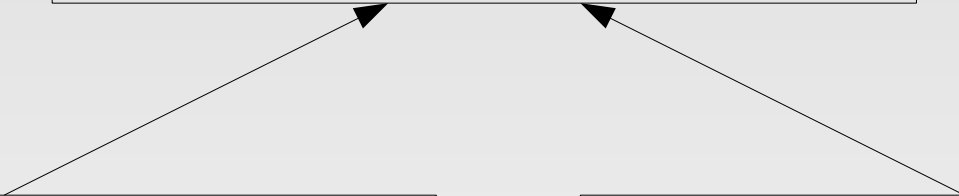
- Wrap existing methods with new functionality
- Previously:
alias_method :foo_without_bar, :foo
alias_method :foo, :foo_with_bar
- Now:
alias_method_chain :foo, :bar
- Help avoid infinite loops and errors introduced when methods are mixed-in in different orders
- Note: You must define :foo_with_bar first!

Addresses Using STI

Class: Address
inheritance_column = 'address_type'
address_type: nil

Class: HomeAddress
inheritance_column = 'address_type'
address_type: 'HomeAddress'

Class: WorkAddress
inheritance_column = 'address_type'
address_type: 'WorkAddress'



Combined Example: STI Factory

```
module StiClassMethods
  def subclass_names
    subclasses.map(&:name).push(self.name)
  end

  def new_with_factory(*args)
    options = args.extract_options!

    class_name =
      options.delete(self.inheritance_column.to_sym) ||
      self.name
    klass = self.subclass_names.include?(class_name) ?
      class_name.constantize : self

    klass.new_without_factory(*args.push(options))
  end
end
```

Combined Example (cont)

```
module ClassMethods
  def has_sti_factory
    extend Koinonia::StiFactory::StiClassMethods
    class << self
      alias_method_chain :new, :factory
      unless method_defined?(:new_without_factory)
      end
    end
  end
end
```

High Crimes and Misdemeanors

```
<% for attendee in @attendees do %>
  <tr>
    <td><%=h attendee.first_name %></td>
    <td><%=h attendee.last_name %></td>
    <td><%=h attendee.paid_on %></td>
    <td><%= link_to 'Show', attendee_path(attendee) %></td>
    <td><%= link_to 'Edit',
                  edit_attendee_path(attendee) %></td>
    <td><%= link_to 'Destroy', attendee_path(attendee),
                  :confirm => 'Are you sure?',
                  :method => :delete %></td>
  </tr>
<% end %>
```

Put the *Objects* Back in OOP!

```
<% @attendees.each do |attendee| %>
  <tr>
    <td><%=h attendee.first_name %></td>
    <td><%=h attendee.last_name %></td>
    <td><%=h attendee.paid_on %></td>
    <td><%= link_to 'Show', attendee_path(attendee) %></td>
    <td><%= link_to 'Edit',
                  edit_attendee_path(attendee) %></td>
    <td><%= link_to 'Destroy', attendee_path(attendee),
                  :confirm => 'Are you sure?',
                  :method => :delete %></td>
  </tr>
<% end %>
```

All Together Now

- Bad:

```
def attendee_names
  names = []
  Attendee.find(:all).each do |attendee|
    names << attendee.name
  end
  return names
end
```

#collect the answers instead!

- Much Better

```
def attendee_names
  Attendee.find(:all).collect do |attendee|
    attendee.name
  end
end
```

Or #map the instance methods!

- Best!

```
def attendee_names
  Attendees.all.map(&:name)
end
```

Enumerable#inject

- `@collection.inject(accumulator){...block...}`
 - What it does:
Iterate over the collection, operate on each item, and 'accumulate' the results into an accumulator
 - Similar to but more basic than map/collect
 - Allows a greater range of transformations
 - **WARNING:** inject is *functional*

```
sum = [1, 2, 3, 4, 5].inject(0) do |acc, val|  
  val+sum  
end
```

Example: Filling out a calendar

```
start_date = Date.today.beginning_of_month
end_date = start_date.end_of_month
appointments = Appointment.find(:all,
  :conditions=>{:scheduled_at=>[start_date..end_date]})
@appointments = appointments.inject({}) do |by_date, appt|
  scheduled_on = appt.scheduled_at.to_date
  appts = by_date.delete(scheduled_on) || []
  appts.push(appt)
  by_date.merge scheduled_on => appts
end
```

Double, secret probation

- `lambda{|var1, var2| ...block...}`
 - A verb waiting for a noun to happen to...
 - Ruby's built in command pattern
 - Wrap a method in an object so you can call it later
 - Allows you to abstract code that "differs only in white space"

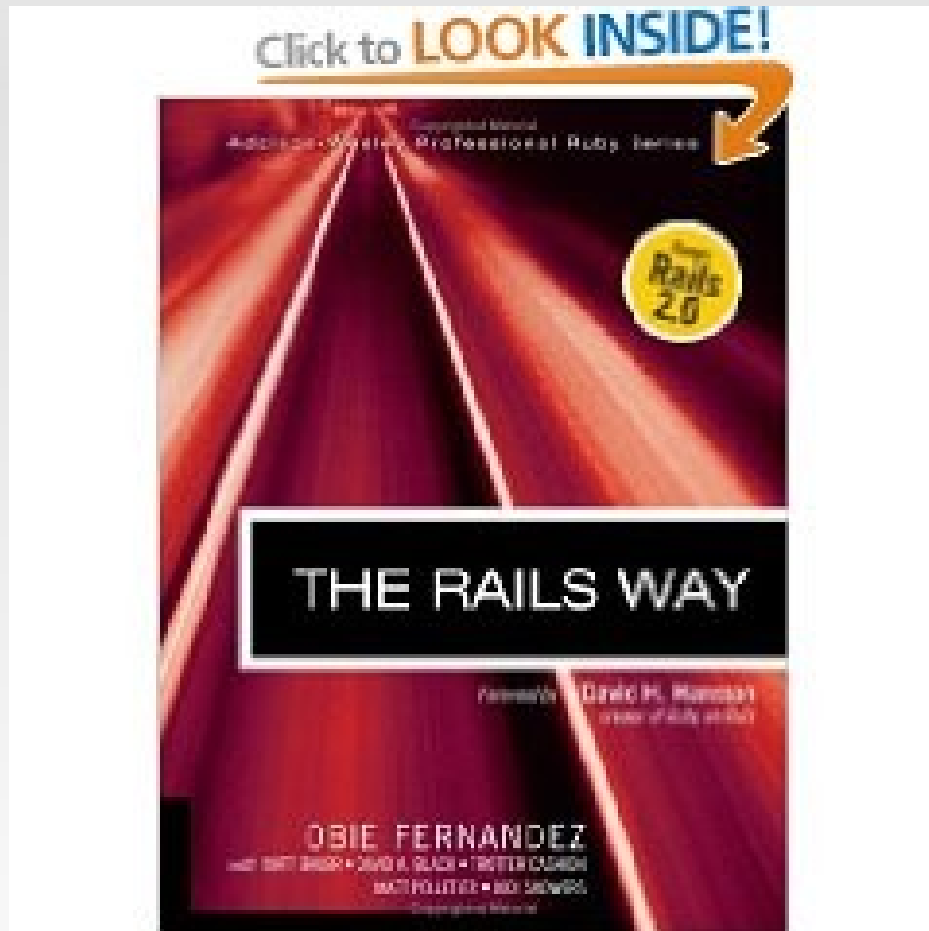
Simple example: add2

```
add2 = lambda|x| x+2}
```

```
add2.call(15)
```

```
#=> 17
```

A tiles Helper



- A thumbnail photo
- A short description
- Usually presented in a grid

The Original Partial

```
1 <table class="cities tiles">
2 <% cities.in_groups_of(columns) do |row| %>
3   <tr>
4     <% row.each do |city| %>
5       <td id=<% dom_id(city) %>">
6         <div class="left">
7           <%=image_tag city.photo.public_filename(:thumb)%>
8         </div>
9         <div class="right">
10          <div class="title"><%=h city.name %></div>
11          <div class="desc"><%= city.description %></div>
12        </div>
13      </td>
14    <% end # row.each -%>
15  </tr>
16 <% end # in_groups_of -%>
17 </table>
```

The Restated Partial (items)

```
1 <table class="tiles">
2 <% collection.in_groups_of(columns) do |row| %>
3   <tr>
4     <% row.each do |item| %>
5       <td id=<% dom_id(item) %>">
6         <div class="left">
7           <%=image_tag item.photo.public_filename(:thumb)%>
8         </div>
9         <div class="right">
10          <div class="title"><%=h item.name %></div>
11          <div class="desc"><%= item.description %></div>
12        </div>
13      </td>
14    <% end # row.each -%>
15  </tr>
16 <% end # in_groups_of -%>
17 </table>
```

The Partial Using Lambdas

```
<div class="left">
  <%= image_tag thumbnail.call(item) %>
</div>
<div class="right">
  <div class="title">
    <%= link_to title.call(item), link.call(item) %>
  </div>
  <div class="desc"><%=h description.call(item) %></div>
</div>
```

The tiled Helper

```
1 def tiled(collection, options={})
2   raise 'link option is required' unless options[:link]
3   options[:columns] ||= 3
4   options[:thumbnail] ||= lambda do |item|
5     image_tag(item.photo.public_filename(:thumb))
6   end
7   options[:title] = lambda{|item| item.to_s}
8   options[:desc] = lambda{|item| item.description}
9   render :partial=> "shared/tiled_table",
10         :locals =>{:collection => collection,
11                   :columns     => options[:columns],
12                   :thumbnail   => options[:thumbnail],
13                   :title       => options[:title],
14                   :description=> options[:desc]}
15 end
```

The tiled Helper in Action

```
<%= tiled(@cities, :link=> lambda{|city| city_path(city)}  
%>
```

:method_missing?

- Questions?
- What didn't I explain clearly?
- What other *Ruby* methods and constructs can you not live without?

Contact me

- Andy Vanasse
andy@pcssinc.com
andyvanasse@gmail.com
upstate-ruby google group
cell: don't call me, I'll call you...
- Slides available at:
<http://upstaterb.org>