

RESTful Everything

Toward a Complete Resource-oriented Workflow

Ingo Weiss, Metaversum

| | METAVERSUM®

twinity^{beta}
powered by real life

1. What is resource-orientation?
2. How does present Rails development benefit from resource-orientation?
3. Could it benefit more? How?

I. What is resource-orientation?

This section merely summarizes parts of the 'Resource-oriented Architecture' (ROA) as outlined in the excellent book 'RESTful Web Services' by Leonard Richardson and Sam Ruby

1. Statelessness
2. Addressability
3. Uniform Interface
4. Connectivity

Uniform Interface

GET POST PUT DELETE

Connectivity

- Resources should be connected by links and forms linking to possible next application states
- ‘Hypermedia as the engine of application state’
- Links and forms are acting as ‘Levers of state’

**II. How does current
RAILS development
benefit from resource-
orientation?**

I. Rails Routes

```
map.resources :tasks
```

GET	/tasks	index
POST	/tasks	create
GET	/tasks/new	new
GET	/tasks/1/edit	edit
GET	/tasks/1	show
PUT	/tasks/1	update
DELETE	/tasks/1	destroy

	/tasks	/tasks/1	/tasks/new	/tasks/edit
GET	index	show	new	edit
POST	create			
PUT		update		
DELETE		destroy		

tasks_path	/tasks
task_path(task)	/tasks/1
new_task_path	/tasks/new
edit_task_path(task)	/tasks/1/edit

2. Rails Controllers

```
class TasksController < ApplicationController  
  def index  
  
  def create  
  
  def new  
  
  def edit  
  
  def show  
  
  def update  
  
  def destroy  
  
end
```

3. Rails Views

```
<%- content_tag_for :li, task do -%>

  <%= link_to(task.name, task_path(task)) %>

  <%= link_to('Edit', edit_task_path(task)) %>

  <%- form_for(task, :html => {:method => :delete}) do |form| -%>
    <%= submit_button 'Delete' %>
  <%- end -%>

  <%- form_for :completion, :url => task_completion_path(task) do |form| -%>
    <%= submit_button 'Done' %>
  <%- end -%>

  <%- form_for task do |form| -%>
    <%= form.select :assignee, ['Joe', 'Janet', 'Jane'] %>
    <%= submit_button 'Assign' %>
  <%- end -%>

  <%- form_for task do |form| -%>
    <%= hidden_field_tag 'task[assignee]', 'Me' %>
    <%= submit_button 'Assign to me' %>
  <%- end -%>

<%- end -%>
```

```
<%- content_tag_for :li, task do -%>

  <%= link_to(task.name, task_path(task)) %>

  <%= link_to('Edit', edit_task_path(task)) %>

  <%- form_for(task, :html => {:method => :delete}) do |form| -%>
    <%= submit_button 'Delete' %>
  <%- end -%>

  <%- form_for :completion, :url => task_completion_path(task) do |form| -%>
    <%= submit_button 'Done' %>
  <%- end -%>

  <%- form_for task do |form| -%>
    <%= form.select :assignee, ['Joe', 'Janet', 'Jane'] %>
    <%= submit_button 'Assign' %>
  <%- end -%>

  <%- form_for task do |form| -%>
    <%= hidden_field_tag 'task[assignee]', 'Me' %>
    <%= submit_button 'Assign to me' %>
  <%- end -%>

<%- end -%>
```

- URL and HTTP methods are sometimes implicit, sometimes explicit
- CRUD actions are (mostly) absent
- Forms and Links are treated very differently
- Knowledge of both routing rules and helper syntax required
- Syntax is elegant in some places at the expense of consistency and expressiveness

- [Water plants Edit](#)

Delete

Done

Joe ▾ Assign

Assign to me

```
<li class="task" id="task_1">
```

```
<a href="/tasks/1">Water plants</a>
```

```
<a href="/tasks/1/edit">Edit</a>
```

```
<form action="/tasks/1" class="edit_task" id="edit_task_1" method="post">
```

```
<input name="_method" type="hidden" value="delete" />
```

```
<button>Delete</button>
```

```
</form>
```

```
<form action="/tasks/1/completion" method="post">
```

```
<button type="submit">Done</button>
```

```
</form>
```

```
<form action="/tasks/1" class="edit_task" id="edit_task_1" method="post">
```

```
<input name="_method" type="hidden" value="put" />
```

```
<select id="task_assignee" name="task[assignee]">
```

```
<option value="Joe">Joe</option>
```

```
<option value="Janet">Janet</option>
```

```
<option value="Jane">Jane</option>
```

```
</select>
```

```
<button type="submit">Assign</button>
```

```
</form>
```

```
<form action="/tasks/1" class="edit_task" id="edit_task_1" method="post">
```

```
<input name="_method" type="hidden" value="put" />
```

```
<input id="task[assignee]" name="task[assignee]" type="hidden" value="Ingo" />
```

```
<button type="submit">Assign to me</button>
```

```
</form>
```

```
</li>
```

- No consistent pattern for classes and ids across links and form
- Increased likelihood of inconsistent, representational class/id attributes
- Many micro-dependencies that make the code brittle and hard to change

What if view helpers would...?

- use the same syntax for both links and forms
- focus on the things that matter: Resource, action, parameters
- consistently set true resource-oriented class attributes

ResourcefulViews

- Plugin
- Extends the `map.resources` method to define a set of seven view helpers for each resource, one for each CRUD action
- Best demonstrated by example...

```
<%- content_tag_for :li, task do -%>

  <%= link_to(task.name, task_path(task)) %>

  <%= link_to('Edit', edit_task_path(task)) %>

  <%- form_for(task, :html => {:method => :delete}) do |form| -%>
    <%= submit_button 'Delete' %>
  <%- end -%>

  <%- form_for :completion, :url => task_completion_path(task) do |form| -%>
    <%= submit_button 'Done' %>
  <%- end -%>

  <%- form_for task do |form| -%>
    <%= form.select :assignee, ['Joe', 'Janet', 'Jane'] %>
    <%= submit_button 'Assign' %>
  <%- end -%>

  <%- form_for task do |form| -%>
    <%= hidden_field_tag 'task[assignee]', 'Ingo' %>
    <%= submit_button 'Assign to me' %>
  <%- end -%>

<%- end -%>
```

```
<%- task_item task do -%>

  <%= show_task(task, :label => task.name) %>

  <%= edit_task(task) %>

  <%= destroy_task(task) -%>

  <%= create_task_completion(task, :label => 'Done') -%>

  <%- update_task(task) do |form| -%>
    <%= form.select :assignee, ['Joe', 'Janet', 'Jane'] %>
    <%= submit_button 'Assign' %>
  <%- end -%>

  <%= update_task(task, :attributes => {:assignee => 'Me'}, :label => 'Assign to me') -%>

<%- end -%>
```

The Helpers

- Are named following the pattern:
[CRUD action]_[resource name]
[CRUD action]_[name prefix]_[resource name]
- Render either a link or a form generating a request that will be routed to that CRUD action
- Take the same parameters as the named route used to generate the request's target URL, plus some options
- Set resource-oriented class attributes

```
<li class="task" id="task_1">
```

```
<a href="/tasks/1" class="task show show_task">Water plants</a>
```

```
<a href="/tasks/1/edit" class="task edit edit_task">Edit</a>
```

```
<form action="/tasks/1" class="task destroy destroy_task" method="post">
```

```
<input name="_method" type="hidden" value="delete" />
```

```
<button type="submit">Delete</button>
```

```
</form>
```

```
<form action="/tasks/1/completion" class="completion create create_completion" method="post">
```

```
<button type="submit">Done</button>
```

```
</form>
```

```
<form action="/tasks/1" class="task update update_task" method="post">
```

```
<input name="_method" type="hidden" value="put" />
```

```
<select id="task_assignee" name="task[assignee]">
```

```
<option value="Janet">Janet</option>
```

```
<option value="Jane">Jane</option>
```

```
<option value="Joe">Joe</option>
```

```
</select>
```

```
<button type="submit">Assign</button>
```

```
</form>
```

```
<form action="/tasks/1" class="task update update_task" method="post">
```

```
<input name="task[assignee]" type="hidden" value="Me" />
```

```
<input name="_method" type="hidden" value="put" />
```

```
<button type="submit">Assign to me</button>
```

```
</form>
```

```
</li>
```

Another view example

```
<%= new_task %>
```



```
<a href="/tasks/new" class="task new new_task">New</a>
```

```
<%= new_task :attributes => {:assignee => 'Fred'},  
      :label => 'Ask Fred' %>
```



```
<a href="/tasks/new?task[assignee]=Fred"  
      class="task new new_task">Ask Fred</a>
```

```
<%- new_task :attributes => {:assignee => 'Fred'} do |form| %>
  <%= form.text_field :name %>
  <%= submit_button 'Ask Fred' %>
<%- end %>
```



```
<form action="/tasks/new" class="task new new_task" method="get">
  <input type="hidden" name="task[assignee]" value="Fred" />
  <input id="task_name" name="task[name]" size="30" type="text" />
  <button type="submit">Ask Fred</button>
</form>
```

```
<%- create_task :attributes => {:assignee => 'Fred'} do |form| %>
  <%= form.text_field :name %>
  <%= submit_button 'Ask Fred' %>
<%- end %>
```



```
<form action="/tasks" class="task create create_task" method="post">
  <input type="hidden" name="task[assignee]" value="Fred" />
  <input id="task_name" name="task[name]" size="30" type="text" />
  <button type="submit">Ask Fred</button>
</form>
```

```
<%= create_task :attributes => {:assignee => 'Fred',  
  :name => 'Feed cat'}, :label => 'Ask Fred to feed the cat' %>
```



```
<form action="/tasks" class="task create create_task" method="post">  
  <input id="task_assignee" name="task[assignee]" type="hidden" value="Fred" />  
  <input id="task_name" name="task[name]" type="hidden" value="Feed cat" />  
  <button type="submit">Ask Fred to feed the cat</button>  
</form>
```

**CSS,
Resource-oriented**

```
form.create button { background-image: url(/images/save.gif) }
form.create_invitation button { background-image: url(/images/invite.gif) }
body.de form.create_invitation button { background-image: url(/images/einladen.gif) }
a.new { background-image: url(/new.gif) }
a.new_download { background-image: url(/images/download.gif) }
body#new_download form.create_download button url(/images/download_big.gif) }
a.edit { background-image: url(/pencil.gif) }
form.destroy button { background-image: url(/cross.gif) }
```



**JavaScript,
resource-oriented**

Example I: Deletion confirmation

form.**destroy**: ConfirmDestroyResource

```
var ConfirmDestroyResource = Behavior.create({
  onsubmit: function(submit_event){

    // extract resource name from class attribute
    var resource_name = ...;

    // ask for confirmation
    if (confirm('Are you sure you want to delete this ' + resource_name + '?')) {
      // send delete request
      new Ajax.Request(this.element.readAttribute('action'), {method: 'delete'});
    }
    submit_event.stop();
  }
});
```



The page at <http://localhost:3000> says:

Are you sure you want to delete this task?

Cancel

OK



The page at <http://localhost:3000> says:
Are you sure you want to delete this task?

Cancel

OK



The page at <http://localhost:3000> says:
Are you sure you want to delete this cms snippet?

Cancel

OK



The page at <http://localhost:3000> says:
Are you sure you want to delete this view translation?

Cancel

OK



The page at <http://localhost:3000> says:
Are you sure you want to delete this occupation?

Cancel

OK

Example II:

Mapping

(Progressive-enhancement-style)

```
<%- restaurant_item restaurant do -%>
```

```
  <%= show_restaurant(restaurant, :label => geo(restaurant)) %>
```

```
<%- end -%>
```



```
<ul class="restaurant_list">
  <li class="restaurant" id="restaurant_1">
    <a href="/restaurants/1" class="show restaurant show_restaurant">
      <abbr class="geo" title="52.43;12.53">Joe's Shanghai</abbr>
    </a>
  </li>
</ul>
```

```
ul.restaurant_list: Map
```

```
var Map = Behavior.create({
  initialize: function(){

    // insert map div:
    map_div = new Element('div', {'id': 'map'});
    this.element.insert({before: map_div});

    // initialize map:
    this._map = new GMap2(map_div);

    // loop through restaurants:
    this.select('li.restaurant').each(function(restaurant){

      // extract coordinates:
      var coordinates = restaurant.down('abbr.geo').getAttribute('title').split(';');

      // link marker to restaurant page and place it on map:
      var point = new GLatLng(parseFloat(coordinates[0]), parseFloat(coordinates[1]));
      var url = restaurant.down('a.show_restaurant').getAttribute('href');
      var marker = new GMarker(point, {icon: icon});
      this._map.addOverlay(marker);
      GEvent.addListener(marker, 'click', function(){window.location = url}.bind(this));

      // hide element:
      this.element.hide();

    })
  }
});
```

- Users with a modern, JavaScript-enabled browser will see Restaurant plotted on google map
- Other users/clients will see perfectly meaningful list of restaurants

**Tests,
resource-oriented**

```
response.should have_tag 'li.task' do
  with_tag 'a.show_task'
  with_tag 'a.edit_task'
  with_tag 'form.destroy_task'
  with_tag 'form.create_completion'
  with_tag 'form.update_task'
end
```

**Conclusion:
A complete
resource-oriented
workflow**

- Consistent resource-oriented ‘story’ running through all parts of the Rails application
- Supports clean separation of semantics, presentation and behavior
- Easy to create default styles and behaviors (great for rapid development)
- Compact and expressive views
- Common class attribute vocabulary shared by designers and developers
- Many micro-dependencies > one giant dependency that is well understood

For more information about the ResourcefulViews plugin please google for 'ResourcefulViews'

For questions and feedback please contact me at ingo@ingoweiss.com

Thank you!

Special thanks goes to:

- Sven Fuchs, Andreas Wolf, Ivo Stock, Niklas Hofer, Michael Nelson, Helmut Ebritsch, for their feedback, ideas, criticism, or just for being their inspiring selves
- The Rails community