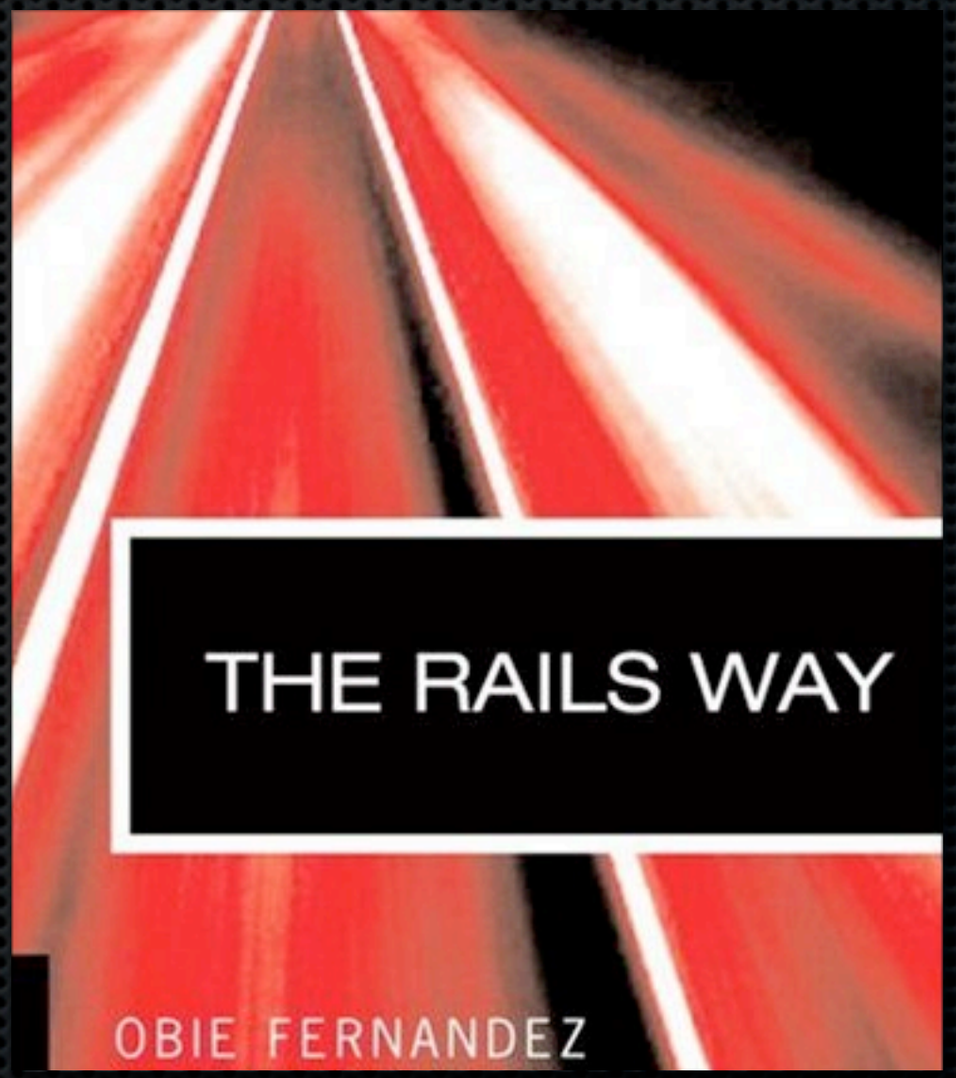
Designing RESTful Rails Applications

Obie Fernandez

Prepared exclusively for QCon 2007



Topics

- What is REST?
- REST in Rails
- Routing and CRUD
- Resources and Representations

Presentation Goals

- It's a big topic, so focusing on the most essential stuff
- Convince you that REST is worth investigating further
- Keep emphasis on practical advice and gotchas
- Don't put you, the audience to sleep

What is REST?

- Rails support (via CR
- but what happens wh

Resource identifiers

- Route configs are goo

templates would

manipulat

xcellent su

REST is an “architectural style” manifested in the web

The REST **constraints** include

- **Use of a client-server architecture**
- **Stateless communication**
- **Explicit signaling of response cacheability**

REST is designed to help you provide services using the native idioms and constructs of HTTP

Just use what HTTP already gives you.

One of the payoffs of REST is that it scales relatively well for big systems, like the **web**.

REST in Rails

view helper methods and enhancements to the routing system

Benefits of RESTful Rails

- Convenience and automatic best practices for you
- A REST interface to your application's services, for everyone else

Much Rails practice is noncompliant with the precepts of REST from the beginning.

Routing and CRUD

REST in Rails involves standardization of action names.

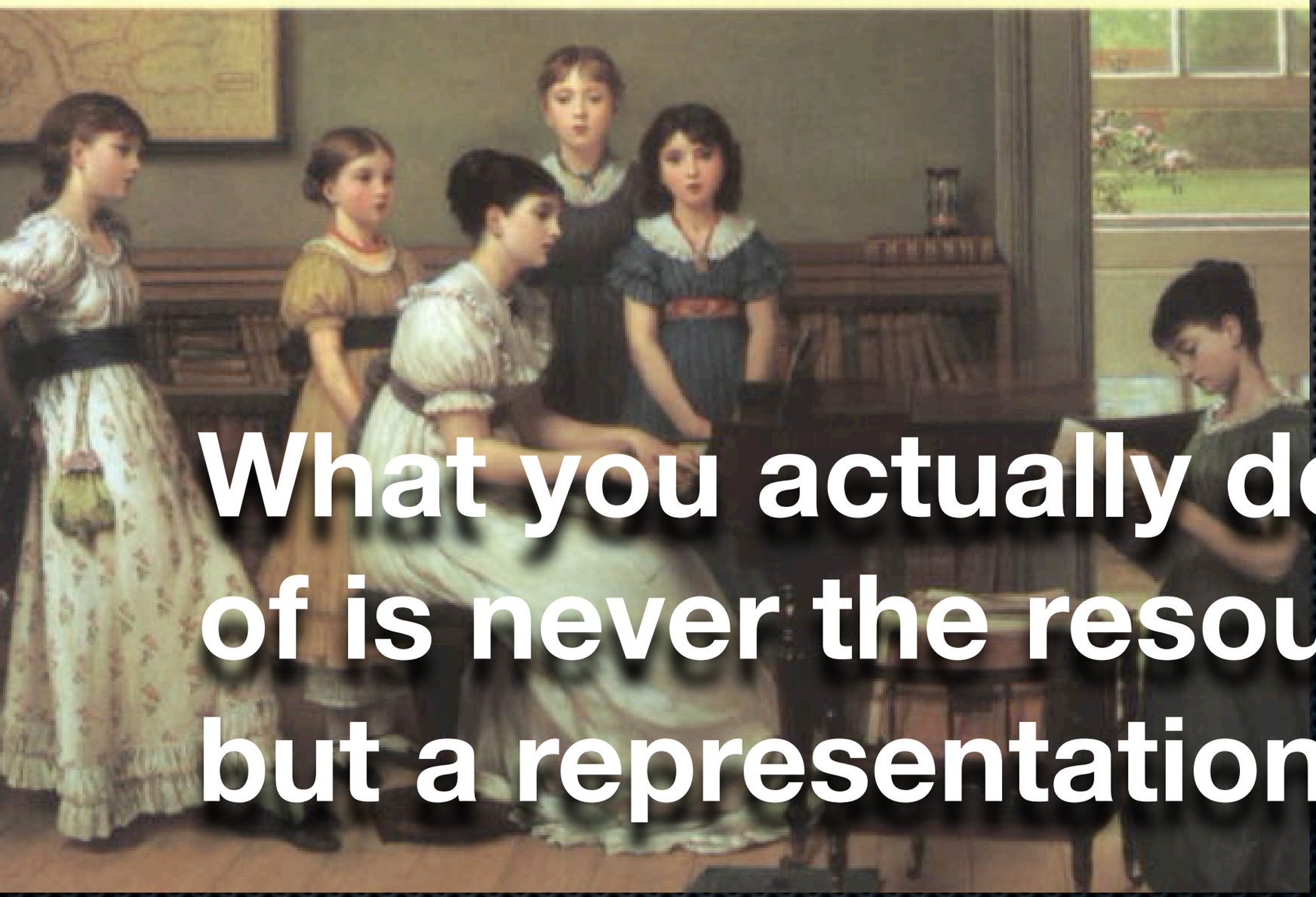
```
map.resources :auctions
```

Resources and Representations

REST characterizes communication between system components as a series of requests to which the responses are **representations** of **resources**.

LITTLE WOMEN

LOUISA MAY ALCOTT



What you actually do get hold of is never the resource itself, but a representation of it.

Opinionated REST Support

```
map.auction ‘auctions/:id’,
  :controller => “auction”,
    :action       => “show”
```

```
<%= link_to item.description,
    auction_path(item.auction) %>
```

auction_delete_path

auction_delete_path

auction_create_path

auction_create_path

auction_path

auction_path

auctions_path

```
<% form_tag auctions_path do |f| %>
```

```
<%= link_to “Click here to view
all auctions”, auctions_path %>
```

http://localhost:3000/auctions

The HTTP Verb

/auctions submitted in a **GET** request!

/auctions submitted in a **POST** request!

map.resources :auctions

The Standard RESTful Controller Actions

index
new
create
show
edit
update
destroy

A lot of work is done for you and the action names are nicely CRUD-like.

Table 4.1 RESTful Routes Table Showing Helpers, Paths, and the Resulting Controller Action

Helper Method	GET	POST	PUT	DELETE
client_url(@client)	/clients/1 *show*		/clients/1 *update*	/clients/1 *destroy*
clients_*url*	/clients *index*	/clients *create*		
edit_client_url(@client)	/clients/1/edit *edit*			
new_client_url	/clients/new *new*			

Rules for Request Methods

- The default is GET
- In a `form_tag` or `form_for` call, the POST method will be used automatically
- If necessary, you can explicitly specify a request method along with the URL generated by the named route (PUT and DELETE operations)

```
<%= link_to “Delete this auction”,
    :url => auction_path(auction),
    :method => :delete %>
```

```
<% form_for "auction",
    :url => auction_path(@auction),
  :html => { :method => :put } do |f| %>
```

The PUT and DELETE hack

The Special Pairs

create and update operations involve two actions

- The action that results in the display of the form
- The action that processes the form input when the form is created

```
<%= link_to "Create a new item",
new_item_path %>
```

```
<%= link_to “Edit”,
edit_item_path(item) %>
```

Singular Named Routes

map.resource :address_book

Nested Resources

/auctions/3/bids/5

/bids/5

/auctions/3/bids/5

params[:auction_id]

```
map.resources :auctions do |auction|
  auction.resources :bids
end
```

<%= link_to “See all bids”,
auction_bids_path(auction) %>

```
<%= link_to “See all bids”,
auction_bids_path(auction) %>
```

/auctions/3/bids

```
<%= link_to “See all bids”,
auction_bids_path(auction) %>
```

/auctions/3/bids

```
params[:auction_id] # in your controller
```

You can nest to any depth.

Each level of nesting adds one to the number of arguments you have to supply to the nested routes.

```
map.resources :my_auctions, :controller => :auctions do |auction|
  auction.resources :my_bids, :controller => :bids
end
```

Restful Route Customizations

```
map.resources :auctions do |auction|
  auction.resources :bids
end
```

```
map.resources :auctions do |auction|
  auction.resources :bids
end
```

/auctions/3/bids/5/retract

retract_bid_url(auction, bid)

```
map.resources :auctions do |auction|
  auction.resources :bids,
    :member => { :retract => :get }
end
```

```
map.resources :auctions do |auction|
  auction.resources :bids,
    :member => { :retract => :get }
  end
```

```
<%= link_to “Retract”, retract_bid_path(auction, bid) %>
```

Different Representations of Resources

The **responds_to** method

http://example.com/auctions.xml

```
def index
  @auctions = Auction.find(:all)
    respond_to do |format|
      format.html
      format.xml {
        render :xml => @auctions.to_xml
      }
  end
end
```

http://example.com/auctions.xml

```
<%= link_to "XML version of this auction",
formatted_auction_path(auction, "xml") %>
```

```
<%= link_to "XML version of this auction",
formatted_auction_path(auction, "xml") %>
```

```
<a href="/auctions/1.xml">XML version of this auction</a>
```

Discussion and Q/A

My Book is Shipping!

CHAPTER 4

REST, Resources, and Rails

> Before REST came I (and pretty much everyone else) never really knew where to put stuff.
>
> —Jonas Nicklas on the Ruby on Rails mailing list

With version 1.2, Rails introduced support for designing APIs consistent with the REST style. Representational State Transfer (REST) is a complex topic in information theory, and a full exploration of it is well beyond the scope of this chapter.[1] We'll touch on some of the keystone concepts, however. And in any case, the REST facilities in Rails can prove useful to you even if you're not a REST expert or devotee.

The main reason is that one of the inherent problems that all web developers face is deciding how to name and organize the resources and actions of their application. The most common actions of all database-backed applications happen to fit well into the REST paradigm—we'll see what that means in a moment.

My Blog

http://jroller.com/obie

If you are a Rails developer and you liked this talk, please recommend me on **workingwithrails.com**