Kim Dalsgaard

Co-owner of, and Software Designer at Trifork Athene
Co-founder of Aarhus Ruby Brigade

REST in Ruby

How Ruby can support a RESTful architecture

What is REST?

REST is first described in Roy Fielding’s PhD dissertation

Architectural Styles and the Design of Network-based Software Architectures

REST is <u>one</u> of these architectural styles

What is an architectural style?

An architectural style is about constraints

What is the constraints for REST?

Key constraints

- Identifiable resources
- Uniform interface
- Stateless communication
- Resource representations
- Hypermedia

Identifiable resources

- A resource represents a real or virtual entity
- Identified by URIs
- Each URI adds value to the Net as a whole

Uniform interface

- GET
- POST
- PUT
- DELETE
- and some more

Stateless communication

- A server does not need to maintain state for each client
- Massive advantages in terms of scalability
- Enforces loose coupling (no shared session knowledge)

Resource representations

- Resources are always accessed through a representation
- Resources should be represented using well-known (standardized) content types
- HTTP provides content types and content negotiation

Hypermedia

- Possible state transitions are made explicit through links
- Links are always provided by the server, not created by the client (low coupling again)

REST Servers

How can the Ruby web frameworks help us with the REST constraints?

Identifiable resources - mapping of URLs to controllers and parameters

Uniform interface - mapping of HTTP verbs to actions

Stateless communication - switching off the session store

Resource representations - executing code according to content type

Hypermedia - creating URLs from objects

REST and Rails

The resource and resources methods for routing

```ruby
ActionController::Routing::Routes.draw do |map|

  map.resources :groups

end
```

```ruby
class GroupsController < ApplicationController

  def index # GET /groups
  end

  def show # GET /groups/{id}
  end

  def update # PUT /groups/{id}
  end

  def create # POST /groups
  end

  def destroy # DELETE /groups/{id}
  end

end
```

```
ActionController::Routing::Routes.draw do |map|

  map.resources :groups do |groups|
    groups.resources :members
    groups.resource :admin
  end

end
```

```
ActionController::Routing::Routes.draw do |map|

  map.resources :groups,
      :has_many => :members,
      :has_one => :admin

end
```

The `respond_to` method for executing code according to content type

```ruby
def index
  @groups = Group.find :all
  respond_to do |format|
    format.html
    format.xml { render :xml => @groups }
    format.json { render :json => @groups }
  end
end
```

The magic _url methods for creating URL's from objects

```
puts groups_url
# => http://<host>/groups

puts group_url @group
# => http://<host>/groups/1

puts group_members_url @group
# => http://<host>/groups/1/members

puts group_member_url @group, @member
# => http://<host>/groups/1/members/2
```

The `session` method for turning off sessions

```ruby
class GroupsController < ApplicationController
  session :off

  def index # GET /groups
    @groups = Group.find :all
    render
  end

end
```

REST and Merb

The resource and resources methods for routing

```ruby
Merb::Router.prepare do |r|
  r.resources :groups
end
```

```ruby
class Groups < Application

  def index # GET /groups
  end

  def show(id) # GET /groups/{id}
  end

  def update(id) # PUT /groups/{id}
  end

  def create # POST /groups
  end

  def destroy(id) # DELETE /groups/{id}
  end

end
```

```ruby
Merb::Router.prepare do |r|

  r.resources :groups do |groups|
    groups.resources :members
    groups.resource :admin
  end

end
```

The `provides` and `display` methods for rendering objects

The `provides` method for registering mime-types to render

The `display` method for rendering objects

```ruby
class Groups < Application
  provides :yaml, :json

  def show(id)
    @group = Group[id]
    display @group
  end

end
```

The url methods for creating URLs from objects

```
puts url(:groups)
# => http://<host>/groups

puts url(:group, @group)
# => http://<host>/groups/1

puts url(:members, @member)
# => http://<host>/groups/1/members

puts url(:member, @member)
# => http://<host>/groups/1/members/2
```

The `:session_store` configuration key for turning off sessions

```ruby
Merb::Config.use do |c|
  c[:session_store] = 'none'
end
```

REST Clients

How can the Ruby HTTP client libraries help us with the REST constraints?

Identifiable resources - holding resource identity

Uniform interface - mapping of HTTP verbs to method calls

Stateless communication -
a server responsibility

Resource representations - setting the ‘Accept’ header

Hypermedia - fetching and following URLs

REST and Net::HTTP

```ruby
require 'net/http'
include Net

url = URI.parse('http://host/index.html')
req = HTTP::Get.new(url.path)
res = HTTP.start(url.host, url.port) {|http|
  http.request(req)
}
puts res.body
```

Too low level!

REST and rest-open-uri

```ruby
require 'rest-open-uri'

uri = URI.parse "http://host/groups"
uri.open :method => :post, :body => pl do |r|
  puts r.status
end

uri = URI.parse "http://host/groups/12"
uri.open "Accept" => "text/xml" do |r|
  puts r.read
end

uri.open :method => :put, :body => pl do |r|
  puts r.status
end
```

- URI objects holding resource identity
- HTTP verbs mapped to key/value pair in options Hash
- Low level access the ‘Accept’ header
- No fetching and following URLs

REST and RestClient

```ruby
require 'rest_client'

include RestClient

groups = Resource.new 'http://host/groups'
groups.post "<group>...</group>"

group = Resource.new 'http://host/groups/12'
put group.get :accept => "application/json"
group.put "{ name: 'The Ruby Group', ...}",
    :content_type => "application/json"
group.delete
```

- Resource objects holding resource identity
- HTTP verbs mapped to methods
- Easy to set the ‘Accept’ header
- No fetching and following URLs
- Buggy post method!

REST and ActiveResource

```ruby
require 'activeresource'

class Group < ActiveResource::Base
  self.site = "http://host"
end

group = Group.create :name => "Ruby"

group.find 1
admin = group.admin

group.name = "The Ruby Group"
group.save
group.destroy
```

- Base objects holding resource identity
- HTTP verbs mapped to alternative methods
- Fetching and following URLs
- Possible to change the serialization format

- Links are not provided by the server, but created by the client!
- Too much additional protocol

Questions?