STEPHEN CHU . com

My thoughts on software development, open source, agile methodology, technology, and stuff that matters.

THURSDAY, NOVEMBER 15, 2007

Seeing Rails Resources Clearly

REST-fulness is probably one of the hot topics you will face when you walk into the Rails world today. One of the terms you will notice when you dig deeper into Rails REST-ful routes is "resources". In Rails, after running the scaffold resource generator, you will get a database table migration, a domain model class, a controller, and 4 CRUD views. This is the skeleton that Rails provides for you to build upon.

However, a lot of Rails developers don't know how to bend this structure when it comes to non-CRUD operations (eg. search for something, reset password, login, various ajax actions etc.)

I have a solution for you: *Routes Driven Development* - Let your named routes guide you. Here's how.

As an example, suppose you have a Rails app that an admin can create/remove users that access your application. You have a database table named Users, which has a password column.

In your routes.rb, with a single line of:

map.resources :users

You automatically get these academically REST-ful urls:

```
GET    /users          # => users_url
POST   /users
GET    /users/new      # => new_user_url
GET    /users/:id/edit # => edit_user_url
GET    /users/:id      # => user_url
PUT    /users/:id
DELETE /users/:id
```

Then, there comes a requirement that says "From the login page, a user who forgot his password can go to a page to see directions on how to retrieve his forgotten password." We are all familiar with this concept, right?

The next thing you know, the developer jumps into the code:

```
class UsersController < ActionController::Base
```

```
  def index...
  def show...
  def new...
  def edit...
  def create...
  def update...
  def destroy...

  def forgot_password
    # new code here
  end
```

And in your routes.rb:

```
map.resouces :users
map.forgot_password_url    '/users/forgot_password', :controller => 'users', :action => 'forgot_password', :method => :get
```

Oops. And you just made a wrong move in building upon the Rails skeleton. You put the action in the wrong controller.

Most developers make the assumption that a Rails "resource" is synonymous to a database table. This observation is made because this is how most people would start their Rails application, as in our case our Users concept. With this in mind, since password is a column rather than a table in the database, it should not have its own controller, and thus no model, nor view, etc. Still, we got to jam the code somewhere, and therefore the Users controller, the table on which it belongs. But this observation of one-table, one-controller is not always true.

Instead, I would keep my users controller free from password related operations and do this:

```
class PasswordsController < ActionController::Base
  def forgot
    # new code here
  end

map.resource :password, :member => { :forgot => :get }
```

This gives me an url of:

```
GET: /passwords/forgot    # => Which gives you forgot_password_url
```

My named route of forgot_password_url is telling me that I am doing the right thing, because it is extremely readable. Had I not been creating a new passwords controller and put the action in my

users controller, using the same technique I used to define my named route, I would end up with a not only incorrect but also nasty forgot_password_user_url(:id) by the naming convention (<action>_<controller>_url). Definitely not what I want to pass to any link_to() or button_to().

If you realize, you just created a controller (and its view) with no database table backing it what-so-ever. It doesn't even have a model class. But then again, how much more complicated can it be when the customer is asking for a page that shows directions?

By putting actions in the wrong controller, your controller actions will lack cohesiveness (i.e. actions don't operate on the right thing) which inhibits sharing, you will also run into problems in sharing views (pages, partials, rjs, helpers) like "why am I rendering a partial that is in an obscure folder?", etc. In short, a lot of bad code.

Using this routes driven technique to drive your actions and controllers, can you identify which action/controller combination is correct for a 'Login' button?

1) create_session_url # => *:controller => 'sessions', :action => 'create', :method => :post*
2) login_user_url # => *:controller => 'users', :action => 'login', :method => :post*
3) login_url # => *map.login '/login', :controller => 'whatever', :action => 'anything', :method => :post*

The true answer is, it doesn't really matter, as long as your controller is responsible for the correct "resource", and the named route is screamingly readable to anyone using it.. No matter which controller you put your actions in, you can always fall back to creating a named route for it using `map.connect` like option (3) above. The bottom line is, Rails "resources" might not be what you and others think it is.

As homework, go read more about Rails resources. Not only can you specify :member, but learn how to use other options like :new, :collection, and nested routes. Also, there is also a singular "resource" method and a plural "resources" method. Get familiar with them. This rdoc page is IMHO pretty poorly documented, but hopefully this article gives you a reason to explore them.

Posted by Stephen Chu at 8:02 PM
Labels: pattern, ruby/rails

3 comments:

Chris said...

I'm just starting to understand REST-fulness and this really helped. Thanks :)

11/16/2007 02:53:00 AM

agum said...

Thanks for the useful articles, man.
Btw, this is Bigi.

I guess I am still considered a newbie at Rails, so I find your stuff quite helpful in my learning.