STEPHEN CHU . com

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

THURSDAY, MARCH 27, 2008

params[:fu] #5) Update multiple models in `update` action atomically.

Updating multiple models is hard? It sounds complicated, but with Rails it actually isn't, if you know how to take advantage of it. Knowing what you know about Rails params, let's take a look at today's topic: the `update` action.

```
<% form_for @reader do |f| %>
  <%= f.text_field :name %>

  <% for @subscription in @subscriptions %>
    <p>
    <% fields_for @subscription do |ff| %>
      <%= ff.collection_select :magazine_id, Magazine.find(:all), :id, :name, {}, :index => @subscription.id %>
    <% end %>
    </p>
  <% end %>

  <%= f.submit 'Save' %>
<% end %>
```

generates:

```
<p>
  <select id="subscription_4_magazine_id" name="subscription[4][magazine_id]">
    <option value="101">PC Magazine</option>
    <option value="102">IT Pro</option>
    <option value="103" selected="selected">WIRED</option>
  </select>
</p>

<p>
  <select id="subscription_5_magazine_id" name="subscription[5][magazine_id]">
    <option value="101">PC Magazine</option>
    <option value="102">IT Pro</option>
    <option value="103" selected="selected">WIRED</option>
  </select>
</p>
```

... (and more)

```
Processing ReadersController#update (for 127.0.0.1 at 2008-01-14 21:12:56) [PUT]
Parameters: { "commit"             => "Update",
              "reader"             => { "name" => "stephen chu" },
              "subscriptions"      => { "4" => { "magazine_id" => "101" },
                                        "6" => { "magazine_id" => "102" },
                                        "7" => { "magazine_id" => "103" } },
              "authenticity_token" => "238ba79b8282882ba01d840352616c2cc79280f0",
              "action"             => "create",
              "controller"         => "readers" }
```

See the pattern? The POST-ed parameters are of the same structure as in one of our last example, hash of hashes. The sub-hashes are keying off of the subscription id, because this time around we are updating existing subscriptions. So last time we used `params[:subscriptions].values`, what would it look like this time? Let take a look.

```
def update
  @reader = Reader.find params[:id]
  @reader.attributes = params[:reader]
  @reader.subscriptions.update params[:subscription].keys, params[:subscription].values

  if @reader.save
    flash[:notice] = 'Reader was successfully updated.'
    redirect_to @reader
  else
    flash[:notice] = 'Failed.'
  end
end
```

Again, another ActiveRecord model method utilizes the array of hashes pattern! The `update` method source code on RDoc looks like it is just updating one at a time. But a peek at the source code says otherwise:

```
def update(id, attributes)
  if id.is_a?(Array)
    idx = -1
    id.collect { |id| idx += 1; update(id, attributes[idx]) }
  else
    object = find(id)
    object.update_attributes(attributes)
    object
  end
```

```
end
```

Again, the `update` method recognizes array! So, where to get the arrays that we will use in our controller action? They come from `.keys` and `.values` of course:

```
params[:subscriptions].keys    # => [ "4", "6", "7" ]
params[:subscriptions].values  # => [ { "magazine_id" => "101" }, { "magazine_id" => "102" }, ... ]
```

So in essence, our controller code is free from all those ugly params-munging activities. Remember, controller actions should not shuffle around their params, or otherwise it fails to abide the "Skinny Controller, Fat Model" principle, and they will stink.

Now, if you are thinking about by using `update`, we run the risk of not atomically saving all of our models should any of our models fail validation, you are correct. This is where `rescue_from` in controller saves the day. Just transact our `update` action using AR transaction, and re-render the edit page should it catches `ActiveRecord::RecordInvalid` error, you should be able to make your `update` action atomic. Given how lean our controller action looks like, having a transaction block that wraps around our code is not so much a nuisance anymore.

This also wraps up our `params[:fu]` series. Remember, how you assemble your views form elements have a lot to do with how thin and skinny your controllers look like. Thanks for reading!

Posted by Stephen Chu at 10:25 PM
Labels: ruby/rails

1 comments:

Daniel said...

I have one question about this. Do you also know a nice method to cleanup the controller code when the user has the possibility to add and/or remove entries and the amount of the entries is flexible.

Nice series thank you.

3/31/2008 10:51:00 PM

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)