shortcut

Addison-Wesley Professional Ruby Series

Rails Refactoring to Resources

Using CRUD and REST in Your Rails Application

Trotter Cashion

Addison-Wesley
Pearson Education

www.awprofessional.com/ruby

Visit us on the Web: www.awprofessional.com

Pearson Education, Inc.
Rights and Contracts Department
75 Arlington St., Suite 300
Boston, MA 02116
Fax: (617) 848-7047

ISBN-13: 978-0-321-50174-5
ISBN-10: 0-321-50174-8

First release, April 2007

```
class CommentsController < ApplicationController
  before_filter :get_task

  def add_comment
    @comment = @task.comments.new
  end

  def create_comment
    @comment = @task.comments.build(params[:comment].merge(:user_id => current_user.id))
    if @comment.save
      flash[:notice] = 'Comment was successfully created.'
      redirect_to :controller => 'tasks', :action => 'show', :id => @comment.task.id
    else
      render :action => 'add_comment'
    end
  end

  def destroy_comment
    @comment = @task.comments.find(params[:comment_id]).destroy
    redirect_to :controller => 'tasks', :action => 'show', :id => @task.id
  end

  protected
    def get_task
      @task = Task.find(params[:id])
    end
end
```

Now all our tests for the comments controller should pass. We can safely remove the actions from `TasksController`, along with their associated tests. Doing so also requires that we switch the links on `Tasks#show` to now point to the comments controller. We should also run our integration tests (preferably written with selenium or watir) to ensure that we have not missed any links. When all our tests pass, we take the next step of this refactoring, which is to rename the methods on `CommentsController` to fit the CRUD standard.

```
class CommentsController < ApplicationController
  before_filter :get_task

  def new
    @comment = @task.comments.new
  end

  def create
    @comment = @task.comments.build(params[:comment].merge(:user_id => current_user.id))
    if @comment.save
      flash[:notice] = 'Comment was successfully created.'
      redirect_to :controller => 'tasks', :action => 'show', :id => @comment.task.id
    else
      render :action => 'add_comment'
    end
  end

  def destroy
    @comment = @task.comments.find(params[:comment_id]).destroy
    redirect_to :controller => 'tasks', :action => 'show', :id => @task.id
  end
```

```
  protected
    def get_task
      @task = Task.find(params[:id])
    end
end
```

In addition to changing the controller, you also must search your application for all links to the old action names and change them to the new names. I find that using grep with the regular expression "`action.*your_action_name`" usually finds all occurrences. Now it is time for the final step, which is to change `params[:id]` to `params[:task_id]` and `params[:comment_id]` to `params[:id]`. Normally, I perform this step at the same time as the previous one because it involves finding all the links to the actions. However, I separate it here to show you the smallest possible steps you can take. Backing out a change and redoing it in smaller steps can be a lifesaver when your refactoring appears to introduce new bugs.

```
class TasksController < ApplicationController
  before_filter :get_project

  def index
    list
    render :action => 'list'
  end

  def list
    @task_pages, @tasks = paginate :tasks, :conditions =>
    ['&&project_id = ? '&&, @project.id], :per_page => 10
  end
```

```
  def show
    @task = @project.tasks.find(params[:id])
    @comments = @task.comments
  end

  def new
    @task = @project.tasks.new
  end

  def create
    @task = @project.tasks.build(params[:task])
    if @task.save
      flash[:notice] = 'Task was successfully created.'
      redirect_to :action => 'list'
    else
      render :action => 'new'
    end
  end

  protected
    def get_project
      @project = Project.find(params[:project_id])
    end
end
class CommentsController < ApplicationController
  before_filter :get_task
```

```
  def new
    @comment = @task.comments.new
  end

  def create
    @comment = @task.comments.build(params[:comment].merge(:user_id => current_user.id))
    if @comment.save
      flash[:notice] = 'Comment was successfully created.'
      redirect_to :controller => 'tasks', :action => 'show', :task_id => @comment.task.id
    else
      render :action => 'add_comment'
    end
  end

  def destroy
    @comment = @task.comments.find(params[:id]).destroy
    redirect_to :controller => 'tasks', :action => 'show', :task_id => @task.id
  end

  protected
    def get_task
      @task = Task.find(params[:task_id])
    end
end
```

I included the `TasksController` again so you can see that it is much cleaner without the `Comment` actions on it. It more accurately represents what it does, and the new `CommentsController` is a place to put code dealing with `Comment`. With fewer actions per controller, it is easier to discern the purpose of each controller and it will be easier for you to decipher your code in the future.

3.2.4 More Information

Sometimes it might seem impossible to perform a CRUD refactoring because the extra methods on a controller are responsible for maintaining a many-to-many relationship. In these cases, it is necessary to first refactor the many-to-many relationship into `has_many :through`. This concept is explained in more detail in DHH's RailsConf 2006 Keynote [Hansson].

3.3 Resource

When you want to use ActiveResource to interact with a controller, a resource refactoring is necessary.

3.3.1 Motive

A resource refactoring is also necessary if you want to support ActiveResource and the Rails way of doing REST. I am not completely sold on the Rails concept of REST (each controller strives to have seven actions and XML is returned using `to_xml`), however, it might be required by your company that you have a REST API. Whether you want REST or are required to use REST, resource refactoring is a necessary step.