

controllers

data layer between the view and model

screenshots from:

http://guides.rubyonrails.org/action_controller_overview.html

naming convention

something plural with "Controller" on the end

`RoomsController`

creating a controller

subclass ApplicationController

each public method inside the class is an "action"

action

the destination of a route

params

hash available in every action
holds query string parameters and post data

arrays in params

```
?ids[ ]=1&ids[ ]=2&ids[ ]=3
```

will become

```
params[:ids]
```

hash in params

To send a hash you include the key name inside the brackets:



```
<form accept-charset="UTF-8" action="/clients" method="post">
  <input type="text" name="client[name]" value="Acme" />
  <input type="text" name="client[phone]" value="12345" />
  <input type="text" name="client[address][postcode]"
value="12345" />
  <input type="text" name="client[address][city]" value="Carrot
City" />
</form>
```

respond_to

requests for non-html data formats



```
class UsersController < ApplicationController
  def index
    @users = User.all
    respond_to do |format|
      format.html # index.html.erb
      format.xml { render xml: @users }
      format.json { render json: @users }
    end
  end
end
```


filter

non-action (private) methods in a controller that you configure to run before, after or around an action

before_action



```
class ApplicationController < ActionController::Base
  before_action :require_login

  private

  def require_login
    unless logged_in?
      flash[:error] = "You must be logged in to access this
section"
      redirect_to new_login_url # halts request cycle
    end
  end
end
```

after_action

has access to the response data

around_action



```
class ChangesController < ApplicationController
  around_action :wrap_in_transaction, only: :show

  private

  def wrap_in_transaction
    ActiveRecord::Base.transaction do
      begin
        yield
      ensure
        raise ActiveRecord::Rollback
      end
    end
  end
end
```

view conventions

TODO

Routing

maps urls to controller actions
(in config/routes.rb)

screenshots from:

<http://guides.rubyonrails.org/routing.html>

Routing Parameters

also available in params inside an action



```
get '/clients/:status' => 'clients#index', foo: 'bar'
```

Path Generation



```
get '/patients/:id', to: 'patients#show', as: 'patient'
```

and your application contains this code in the controller:



```
@patient = Patient.find(17)
```

and this in the corresponding view:



```
<%= link_to 'Patient Record', patient_path(@patient) %>
```


resources



resources :photos, :books, :videos

restrictions



```
resources :photos, only: [:index, :show]
```

name-spacing



```
namespace :admin do
  resources :articles, :comments
end
```

This will create a number of routes for each of the articles and comments controller. For `Admin::ArticlesController`, Rails will create:

HTTP Verb	Path	Controller#Action	Named Helper
GET	/admin/articles	admin/articles#index	admin_articles_path
GET	/admin/articles/new	admin/articles#new	new_admin_article_path
POST	/admin/articles	admin/articles#create	admin_articles_path

concerns

share routing definitions



```
concern :commentable do  
  resources :comments  
end
```



```
resources :messages, concerns: :commentable
```



```
resources :messages do  
  resources :comments  
end
```

urls from objects



```
<%= link_to 'Ad details', magazine_ad_path(@magazine, @ad) %>
```



```
<%= link_to 'Edit Ad', [:edit, @magazine, @ad] %>
```

bound parameters



```
get ':controller(/:action(/:id))'
```

dynamic segments



```
get ':controller/:action/:id/:user_id'
```



```
get ':controller(/:action(/:id))', controller: /admin\[^\]/+/'
```

static segments




```
get ':controller/:action/:id/with_user/:user_id'
```


query string

 `get ':controller/:action/:id'`

An incoming path of `/photos/show/1?user_id=2`

defaults



```
get 'photos/:id', to: 'photos#show', defaults: { format: 'jpg' }
```

naming routes



```
get ':username', to: 'users#show', as: :user
```

redirect

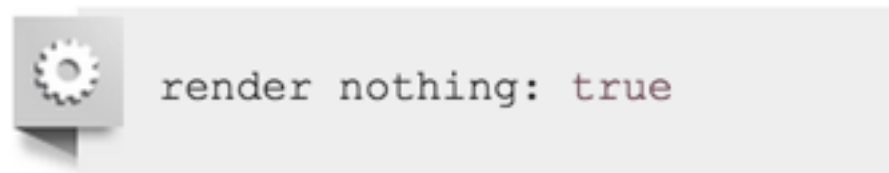


```
get '/stories', to: redirect('/articles')
```

rendering


sends a response to the request
called from actions

nothing



not actually nothing, because a response is sent (headers but no data)

switching views



```
def update
  @book = Book.find(params[:id])
  if @book.update(book_params)
    redirect_to(@book)
  else
    render "edit"
  end
end
```

switching controllers



render from string



render plain: "OK"



render html: "Not Found".html_safe

render from an object



```
render json: @product
```



```
render xml: @product
```

:layout



```
render layout: "special_layout"
```



```
render layout: false
```

layouts

wrapper views. use yield to render the inner view



```
<html>
  <head>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

content_for

yielding multiple sections



```
<html>
  <head>
    <%= yield :head %>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```



```
<% content_for :head do %>
  <title>A simple page</title>
<% end %>

<p>Hello, Rails!</p>
```



```
<html>
  <head>
    <title>A simple page</title>
  </head>
  <body>
    <p>Hello, Rails!</p>
  </body>
</html>
```

partials



```
<%= render "shared/menu" %>
```

That code will pull in the partial from `app/views/shared/_menu.html.erb`.

partial layouts



```
<%= render partial: "link_area", layout: "graybar" %>
```

passing data to partials



```
<h1>New zone</h1>  
<%= render partial: "form", locals: {zone: @zone} %>
```


:object

Every partial also has a local variable with the same name as the partial (minus the underscore). You can pass an object in to this local variable via the `:object` option:



```
<%= render partial: "customer", object: @new_customer %>
```

collection partial



```
<h1>Products</h1>  
<%= render partial: "product", collection: @products %>
```