Ruby on Rails: Introduction and Project 3

CS 142 Section – October 11, 2010

Overview

- Rails Basics
- Controllers and Views
- View Helpers
- Layouts
- Partials

Creating a Rails project

>> rails <dirname>

where <dirname> is your desired project folder

Note: if you are using Rails 3.0, you'll need to use

>> rails new <dirname>

Creates many directories -- for Proj. 3, we're only really concerned w/:

app/

|-- app/models/

|-- app/controllers/

|-- app/views/

|-- app/views/layouts

public/

|-- public/images/

|-- public/stylesheets/

Model-View-Controller

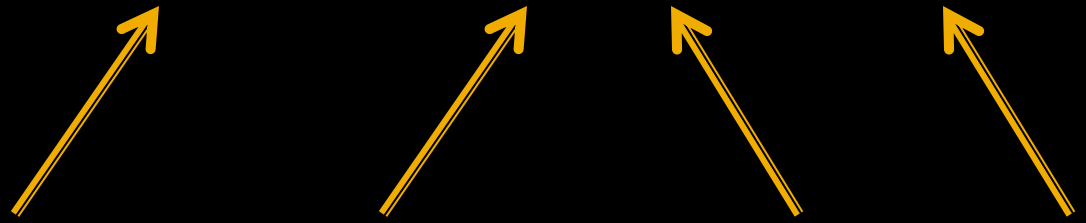
Models are Ruby classes that manage data
(used very sparingly in project 3)

Views are what the user sees: they contain your HTML, CSS, JavaScript.

Controllers generally do "browser stuff"
- parsing your URLs into actions and parameters
- assembling data to be displayed in a view

The Basics

http://localhost:3000/one/two?query=hello



app hostname controller action params

Rails convention:

look up the controller called OneController
call the method named "two" in OneController,
passing in a params hash { :query => "hello" }
find the view corresponding to "two" and display it

The Basics

http://localhost:3000/one/two

1) look up the controller called OneController
   - this will be app/controllers/one_controller.rb

2) call the method named "two" in OneController

3) find the view corresponding to the "two" method of OneController and display it
   - this will be app/views/one/two.html.erb

Controllers

To create a controller, go to the root directory of your Rails project and type:

>> rails script/generate controller <name>
where <name> is the desired controller name

Note: if you are using Rails 3.0, you'll need to use
>> rails generate controller <name>

If we use <name> = one, this creates a controller named OneController, with path app/controllers/one_controller.rb

It also creates an empty folder called app/views/one

Actions

```
class OneController < ApplicationController
    def two
        @string = "two two two"
    end

    def another_action
        # do something...
    end
end
```

Here, calling “two” sets the instance variable @string

Views

Views (also known as templates) in Rails are HTML documents that can be made dynamic through the use of embedded Ruby

```
<ul>
    <% for num in (1..100) %>
        <% if num % 2 == 0 %>
        <li><%= num %></li>
        <% end %>
    <% end %>
</ul>
```

- 2
- 4
- 6
- 8
- 10
- 12
- 14
- 16
- 18
- 20
- 22

They are located in app/views, and always have the extension .html.erb (you may see .rhtml in books or online – that was the pre-Rails 2.0 standard)

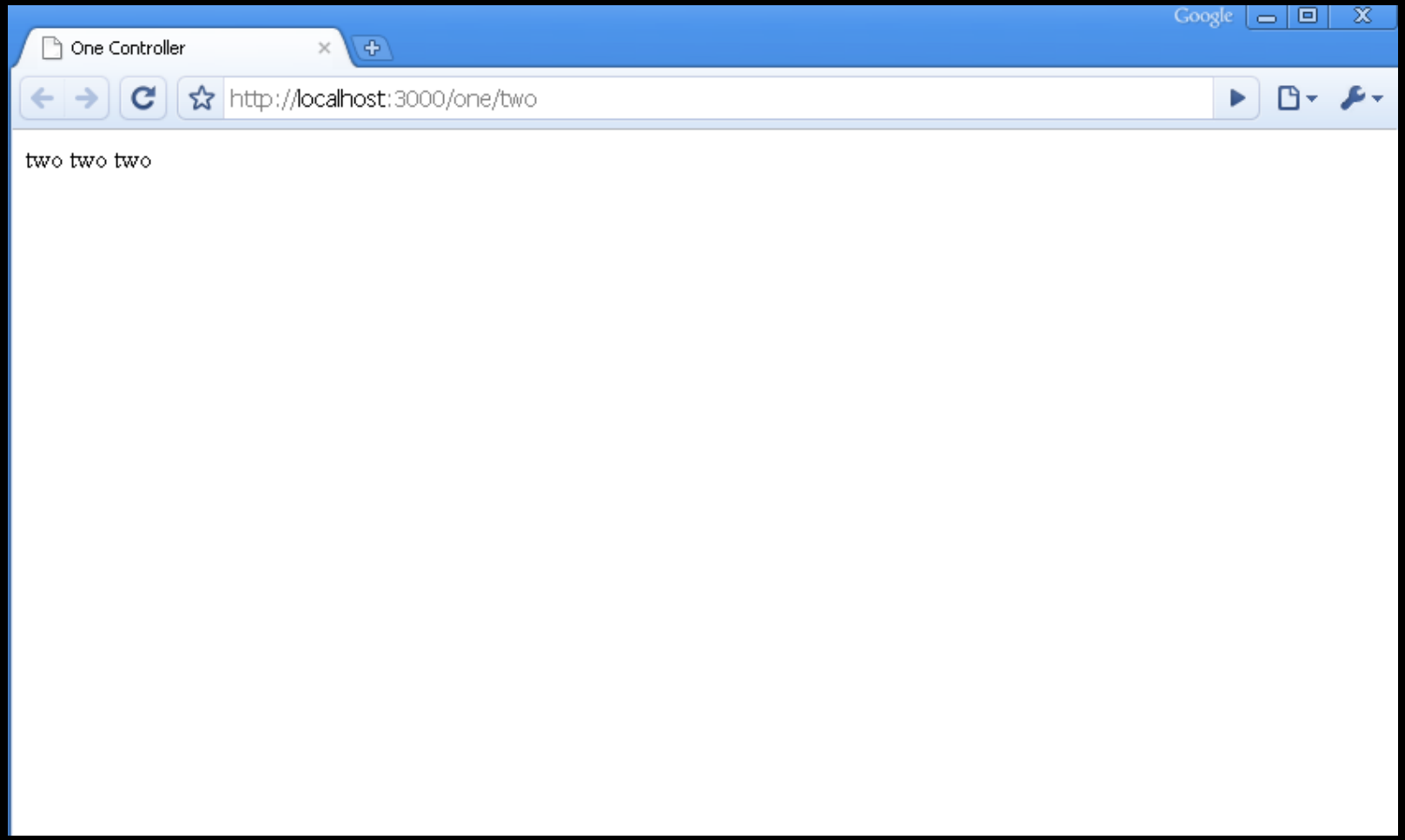
From Actions to Views

The default behavior of the "two" action of OneController is to render whatever is in the file app/views/one/two.html.erb

We can reference OneController's instance variables (e.g. @string) because they are automatically passed into this view

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>One Controller</title>
  </head>
  <body>
    <%= @string %>
  </body>
</html>
```

From Actions to Views



View Helpers

```
<%= link_to "ABC", "http://www.abc.com" %>
```

Generates ABC

```
<%= link_to "ABC", :action => "my_action" %>
```

Creates a link with text ABC that references the my_action action in the current controller.

```
<%= link_to "ABC", :controller => "Bcd",
                   :action => "my_action" %>
```

As above, but routes to the action in Bcd Controller.

View Helpers

<%= stylesheet_link_tag "my_stylesheet" %>

Creates a <link> tag with a reference to the stylesheet public/stylesheets/my_stylesheet.css.

More on helpers in the Rails book, 23.2 and 23.3

Layouts

Layouts are essentially views that wrap other views

Layouts allow you to extract common code between multiple views into a single template; this decreases code repetition and maintenance

Layouts generally reduce boilerplate in your views (e.g. we should use a layout instead of putting the doctype or stylesheet info in every one of our views)

Layouts are located in app/views/layouts

Sections 7.2 and 22.9 in the Rails book

Layout Example

Adapt our previous two.html.erb view to use a layout
(Take all the previous boilerplate and extract it into a re-usable form)

app/views/layouts/application.html.erb
(this is the **global layout** used by all views, unless overridden – see a few slides later)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>One Controller</title>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

two.html.erb will be inserted here when http://HOST/one/two is visited

app/views/two.html.erb

```
<%= @string %>
```

Layouts: Conventions

In app/views/layouts/

application.html.erb will be used for all views (if it is defined)

abc.html.erb will be used for views related to AbcController

abc/xyz.html.erb will be used for the view corresponding to action xyz in AbcController

Layouts: Overriding conventions

You can override these layout conventions in your controllers:

```
class OneController < ApplicationController
  layout "one_layout"

  def two
    @string = "two two two"
    render :layout => "two_layout"
  end

  def another_method
  end
end
```

use one_layout.html.erb for all views corresponding to actions in OneController (instead of one.html.erb or the global application.html.erb)

use two_layout.html.erb for the "two" view

End result: two_layout.html.erb for the "two" view, one_layout.html.erb for everything else in one controller

Partials

Partials (short for partial templates) provide another way to extract components from a page without code repetition

Think of partials like subroutines – they simplify views via decomposition
If you're writing a Facebook-like news feed, you might want every news item to be a partial.

Partials are like any other view, except that their filenames always begin with an underscore (e.g. _three.html.erb)

Partials are invoked from within another view using render (:partial =>)

```
... view code ...
    <%= render (:partial => "three") %>
... view code ...
```

inserts _three.html.erb into the page

Partials: Setting local variables

You can pass a hash of local variables to a partial by passing a :locals parameter to the render method

```
<%= render (:partial => "three",
            :locals => {:foo => "bar"}) %>
```

Partials can then use these locals:

```
... partial code ...
<div>The foo local is <%= foo %></div>
... partial code ...
```

Partials: Setting layouts

A partial can use a layout file, just like any other view can.

```
<%= render (:partial => "three",
            :layout => "some_layout") %>
```

Note: layouts for partials are expected to be in the same folder as the partial (not in the app/views/layout folder!), and also must follow the underscore naming convention.

So, this code will render a partial named _three.html.erb with the layout _some_layout.html.erb.