Linguagem de Programação para Web

Ruby On Rails – parte 4 – Views

Prof. Tales Bitelo Viegas

Views

- Geram a visualização da resposta gerada por um Controller

Respostas

- Do ponto de vista de um Controller, há 3 maneiras de criar uma resposta HTTP:
  - Chamar *render* para criar uma resposta completa para o browser
  - Chamar *redirect_to* para redirecionar o usuário para uma outra URL
  - Chamar *head* para criar uma resposta completa

Resposta padrão

- Por padrão, o Rails renderiza um arquivo ERB (HTML com Ruby Embutido) localizado em /app/views/<controller>/<acao>.html.erb

Usando Render

- Renderizando nada (resposta vazia):
  - render :nothing => true
- Renderizando uma view diferente da padrão (sem executar o código do método)
  - render “edit”
  - render :edit
  - render :action => “edit”

Usando Render

- Renderizando um template de outro controller
  - render ‘products/show’
  - render :template => ‘products/show’
- Renderizando um arquivo qualquer
  - render ‘/usr/apps/teste/current/app/views/c/show’

Usando Render

- Render com :inline
  - render :inline => “<% product.each do |p| %> <%= p.name %> <%= end %>”
- Render plain-text
  - render :text => “OK”
- Render json
  - render :json => @product

Usando Render

- Render XML
  - render :xml => @product
- Render JavaScript
  - render :js => “alert(‘Hello Rails’);”

Usando Render

- Opções
  - render :file => filename, :content_type => “application/rss”
  - render :layout => ‘special_layout’
  - render :layout => false
  - render :status => 500
  - render :status => :forbidden

Evitando erros de renderização

```
def show
  @book = Book.find(params[:id])
  if @book.special?
    render :action => "special_show"
  end
  render :action => "regular_show"
end
```

```
def show
  @book = Book.find(params[:id])
  if @book.special?
    render :action => "special_show" and return
  end
  render :action => "regular_show"
end
```

Usando redirect_to

- Padrão
  - redirect_to photos_url
  - redirect_to :back

- Diferente código de status
  - redirect_to photos_path :status => 301

Layouts

- Quando Rails renderiza uma view, ele combina a view com o layout corrente, usando a seguinte regra:
  - Procura em app/views/layouts por um arquivo ERB com o mesmo nome do Controller
  - Usa o layout de app/views/layouts/application.html.erb
- Com a utilização de layouts, é possível utilizar yield e content_for e Partials

Yield

- Em um contexto de layout, yield identifica uma seção onde o conteúdo deve ser inserido.

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

- Para renderizar um conteúdo dentro de uma yield com nomes, você deve usar um content_for

Content_for

```
<% 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>
```

Layouts Alinhados

- /app/views/layouts/application.html.erb

```
<html>
<head>
  <title><%= @page_title or 'Page Title' %></title>
  <%= stylesheet_link_tag 'layout' %>
  <style type="text/css"><%= yield :stylesheets %></style>
</head>
<body>
  <div id="top_menu">Top menu items here</div>
  <div id="menu">Menu items here</div>
  <div id="content"><%= content_for?(:content) ? yield(:content) : yield %></div>
</body>
</html>
```

Layouts Alinhados

- /app/views/layouts/news.html.erb

```
<% content_for :stylesheets do %>
  #top_menu {display: none}
  #right_menu {float: right; background-color: yellow; color: black}
<% end %>
<% content_for :content do %>
  <div id="right_menu">Right menu items here</div>
  <%= content_for?(:news_content) ? yield(:news_content) : yield %>
<% end %>
<%= render :template => 'layouts/application' %>
```

Form Helpers

- Tags para facilitar o uso de formulários

- Disponível em:
http://guides.rubyonrails.org/form_helpers.html
http://api.rubyonrails.org/