Open Source Frameworks for Rapid Application Development

Marek Krętowski
Krzysztof Bandurski, Tomasz Łukaszuk, Tomasz Rybak

Software Departament
Faculty of Computer Science
Bialystok University of Technology

m.kretowski@pb.edu.pl
k.bandurski@pb.edu.pl, t.lukaszuk@pb.edu.pl, t.rybak@pb.edu.pl

Lecture topic

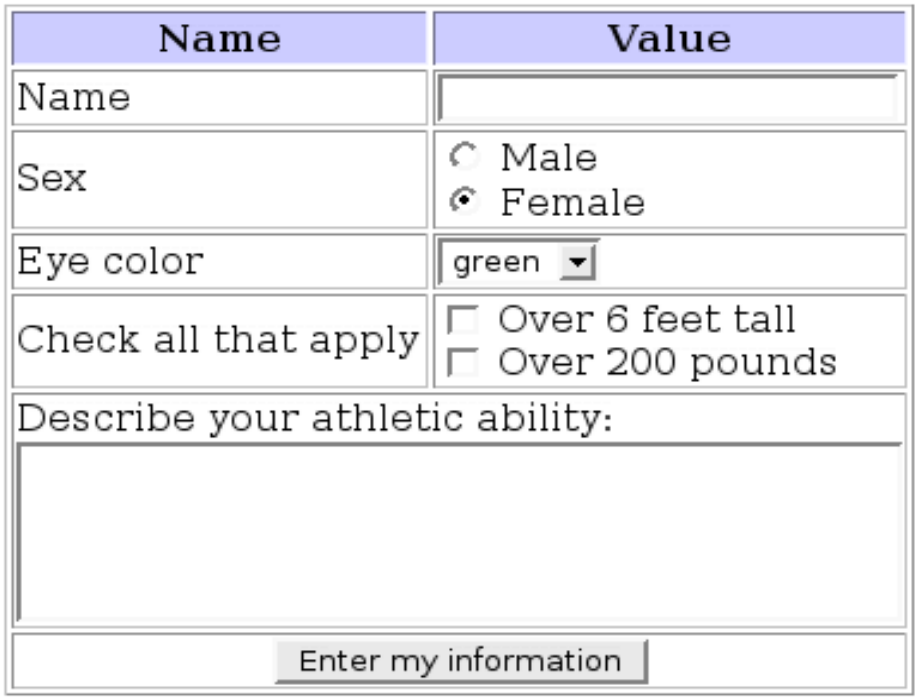
Forms in Ruby on Rails

Forms in Ruby on Rails: Table of content

Introduction

Introducion

- forms in web applications are an essential interface for user input
- form markup can quickly become tedious to write and maintain
- Rails provides view helpers for generating form markup
- developers are required to know all the differences between similar helper methods before putting them to use




```
<form name="input" action="file_name"
      method="get">
 Username:
 <input type="text" name="user" />
 <input type="submit" value="Submit" />
</form>
```

Basic forms

form_tag

```
1  <% form_tag do %>
2    Form contents
3  <% end %>
```

form_tag without arguments creates a form element that has the current page as its action and "post" as its method

```
<form action="/home/index" method="post">
  <div style="margin:0;padding:0">
    <input name="authenticity_token" type="hidden"
        value="f755bb0ed134b76c432144748a6d4b7a7ddf2b71" />
  </div>
  Form contents
</form>
```

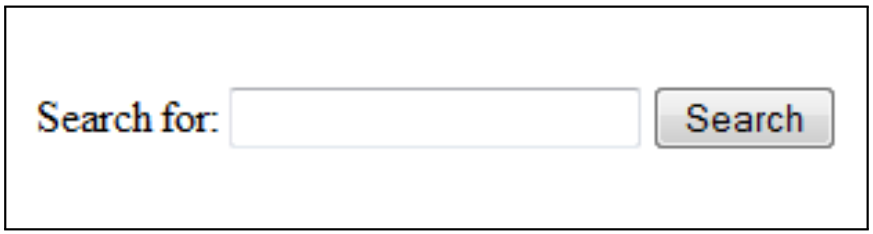
a div element with a hidden input inside - a security feature of Rails called cross-site request forgery protection, form helpers generate it for every form whose action is not "get"

Search form

```
1  <% form_tag(search_path, :method => "get") do %>
2    <%= label_tag(:q, "Search for:") %>
3    <%= text_field_tag(:q) %>
4    <%= submit_tag("Search") %>
5  <% end %>
```

form_tag, label_tag, text_field_tag and submit_tag

```
<form action="/search" method="get">
  <label for="q">Search for:</label>
  <input id="q" name="q" type="text" />
  <input name="commit" type="submit" value="Search" />
</form>
```



Helpers for generating form elements

- basic helpers: ending in _tag such as `text_field_tag`, `check_box_tag`, generate a single `<input>` element
- the first parameter to helper is always the name of the input
- in the controller, this name will be the key in the `params` hash

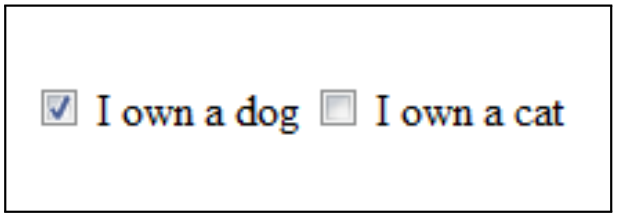
```
1  <%= text_field_tag(:query) %> #in view
2  params[:query]                #in coltroller
```

- when naming inputs be aware that Rails uses certain conventions that control whether values are at the top level of the params hash, inside an array or a nested hash and so on

Checkboxes

```
1  <%= check_box_tag(:pet_dog) %>
2    <%= label_tag(:pet_dog, "I own a dog") %>
3  <%= check_box_tag(:pet_cat) %>
4    <%= label_tag(:pet_cat, "I own a cat") %>
```

```
<input id="pet_dog" name="pet_dog" type="checkbox" value="1" />
  <label for="pet_dog">I own a dog</label>
<input id="pet_cat" name="pet_cat" type="checkbox" value="1" />
  <label for="pet_cat">I own a cat</label>
```

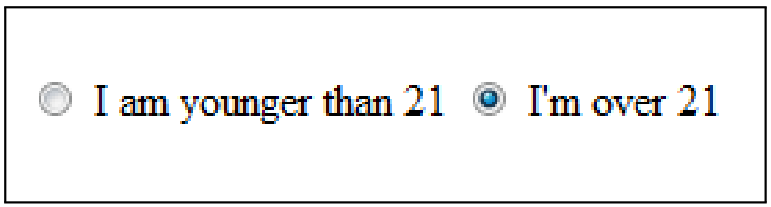


the second parameter to check_box_tag is the value of the input
check the value of `params[:pet_dog]` and `params[:pet_cat]` to see which pets the user owns

Radio buttons

```
1  <%= radio_button_tag(:age, "child") %>
2    <%= label_tag(:age_child, "I am younger than 21") %>
3  <%= radio_button_tag(:age, "adult") %>
4    <%= label_tag(:age_adult, "I'm over 21") %>
```

```
<input id="age_child" name="age" type="radio" value="child" />
  <label for="age_child">I am younger than 21</label>
<input id="age_adult" name="age" type="radio" value="adult" />
  <label for="age_adult">I'm over 21</label>
```



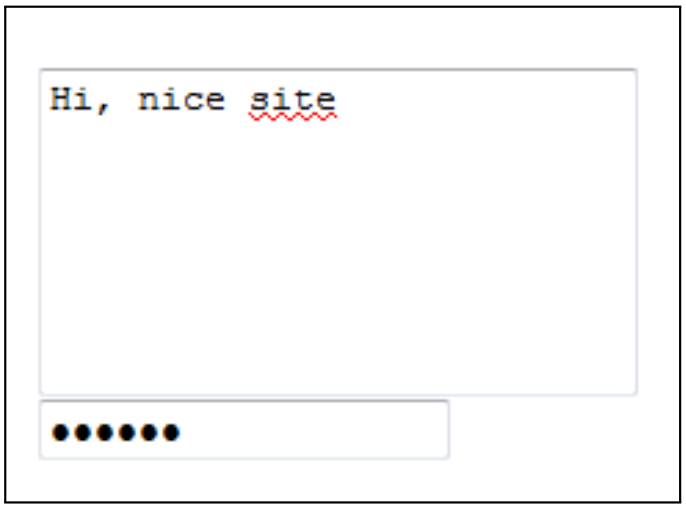
the second parameter to radio_button_tag is the value of the input

check the value of `params[:age]` to see the user answer

Other helpers

```
1  <%= text_area_tag(:message, "Hi, nice site", :size => "24x6") %><br/>
2  <%= password_field_tag(:password) %>
3  <%= hidden_field_tag(:parent_id, "5") %>
```

```
<textarea id="message" name="message" cols="24" rows="6">Hi, nice site</textarea><
<input id="password" name="password" type="password" />
<input id="parent_id" name="parent_id" type="hidden" value="5" />
```



Forms for model objects

Model object helpers

- Model object helpers - helpers lack the `_tag` suffix, for example `text_field`, `text_area`

```
<%= text_field(:person, :name) %>
# => <input id="person_name" name="person[name]" type="text" value="Henry"/>
```

- the first argument is the name of an instance variable
- the second argument is the name of a method (usually an attribute) to call on that object
- Rails will set the value of the input control to the return value of that method for the object and set an appropriate input name
- the value entered by the user will be stored in `params[:person][:name]`
- `params[:person]` hash is suitable for passing to `Person.new` or, if `@person` is an instance of `Person`, `@person.update_attributes`

Binding a form to an object

form_for helper

```
1   #articles_controller.rb
2   def new
3     @article = Article.new
4   end
5
6   #articles/new.html.erb
7   <% form_for :article, @article, :url => { :action => "create" },
8                :html => {:class => "nifty_form"} do |f| %>
9     <%= f.text_field :title %>
10    <%= f.text_area :body, :size => "60x12" %>
11    <%= submit_tag "Create" %>
12  <% end %>
```

```
<form action="/articles/create" method="post" class="nifty_form">
  <input id="article_title" name="article[title]" size="30" type="text" />
  <textarea id="article_body" name="article[body]" cols="60" rows="12"></textarea>
  <input name="commit" type="submit" value="Create" />
</form>
```

f variable is a FormBuilder object

fields_for helper

- fields_for helper - binding without actually creating <form> tags
- useful for editing additional model objects with the same form

```
1  <% form_for @person do |person_form| %>
2    <%= person_form.text_field :name %>
3    <% fields_for @person.contact_detail do |contact_details_form| %>
4      <%= contact_details_form.text_field :phone_number %>
5    <% end %>
6  <% end %>
```

```
<form action="/people/1" class="edit_person" id="edit_person_1" method="post">
  <input id="person_name" name="person[name]" size="30" type="text" />
  <input id="contact_detail_phone_number"
         name="contact_detail[phone_number]" size="30" type="text" />
</form>
```

Forms with PUT or DELETE methods

- most browsers don't support methods other than "GET" and "POST" when it comes to submitting forms
- Rails works around this issue by emulating other methods over POST with a hidden input named _method

```
1   form_tag(search_path, :method => "put")
```

```
<form action="/search" method="post">
  <div style="margin:0;padding:0">
    <input name="_method" type="hidden" value="put" />
    <input name="authenticity_token" type="hidden"
           value="f755bb0ed134b76c432144748a6d4b7a7ddf2b71" />
  </div>
  ...
```

Select boxes

select_tag and options

```
<select name="city_id" id="city_id">
  <option value="1">Lisbon</option>
  <option value="2">Madrid</option>
  ...
  <option value="12">Berlin</option>
</select>
```

- select_tag

```
1  <%= select_tag(:city_id, '<option value="1">Lisbon</option>...') %>
```

- options_for_select

```
1  <%= options_for_select([['Lisbon', 1], ['Madrid', 2], ...],2) %>
2  # => <option value="1">Lisbon</option>
3  # => <option value="2" selected="selected">Madrid</option>
4  # => ...
```

- combination select_tag and options_for_select

```
1  <%= select_tag(:city_id, options_for_select(...)) %>
```

Select boxes and models

- select helper

```
1  # controller:
2  @person = Person.new(:city_id => 2)
3
4  # view:
5  <%= select(:person, :city_id, [['Lisabon', 1], ['Madrid', 2], ...]) %>
6  # select on a form builder
7  <%= f.select(:city_id, ...) %>
```

Option tags from a collection of objects

- a nested array by iterating over them

```
1   <% cities_array = City.find(:all).map { |city| [city.name, city.id] } %>
2   <%= options_for_select(cities_array) %>
```

- options_from_collection_for_select helper

```
1   <%= options_from_collection_for_select(City.all, :id, :name) %>
```

- collection_select helper

```
1   <%= collection_select(:person, :city_id, City.all, :id, :name) %>
```

Date and time form helpers

Date and time form helpers

- dates and times are not representable by a single input element
- barebones helpers: select_date, select_time and select_datetime

```
1   <%= select_date Date::today, :prefix => :start_date %>
```

```
<select id="start_date_year" name="start_date[year]"> ... </select>
<select id="start_date_month" name="start_date[month]"> ... </select>
<select id="start_date_day" name="start_date[day]"> ... </select>
```

- model object helpers: date_select, time_select and datetime_select

```
1   <%= date_select :person, :b_date %>
```

```
<select id="person_b_date_1i" name="person[b_date(1i)]"> ... </select>
<select id="person_b_date_2i" name="person[b_date(2i)]"> ... </select>
<select id="person_b_date_3i" name="person[b_date(3i)]"> ... </select>
```

Uploading files

Uploading files

- the form's encoding MUST be set to multipart/form-data

```
1  <% form_tag({:action => :upload}, :multipart => true) do %>
2    <%= file_field_tag 'picture' %>
3  <% end %>
4
5  <% form_for @person, :html => {:multipart => true} do |f| %>
6    <%= f.file_field :picture %>
7  <% end %>
```

- The object in the `params` hash is an instance of a subclass of `IO`: `StringIO` or `File`

```
1  def upload
2    uploaded_io = params[:person][:picture]
3    File.open(Rails.root.join('public', 'uploads',
4              uploaded_io.original_filename), 'w') do |file|
5      file.write(uploaded_io.read)
6    end
7  end
```

Displaying the validation errorss

Validation helpers

- validation helpers can be used inside class (model) definitions
- validation helpers -> validation rules
- the object's `errors` collection
- helper attributes, `:on` (:save (the default), :create or :update) and `:message` options

```
1  class Person < ActiveRecord::Base
2    validates_acceptance_of :terms_of_service
3    validates_confirmation_of :email
4    validates_presence_of :email_confirmation
5    validates_format_of :description, :with => /^[a-zA-Z]+$/,
6        :message => "Only letters allowed"
7    validates_length_of :name, :minimum => 2
8    validates_uniqueness_of :email
9  end
```

you can write your own validation methods

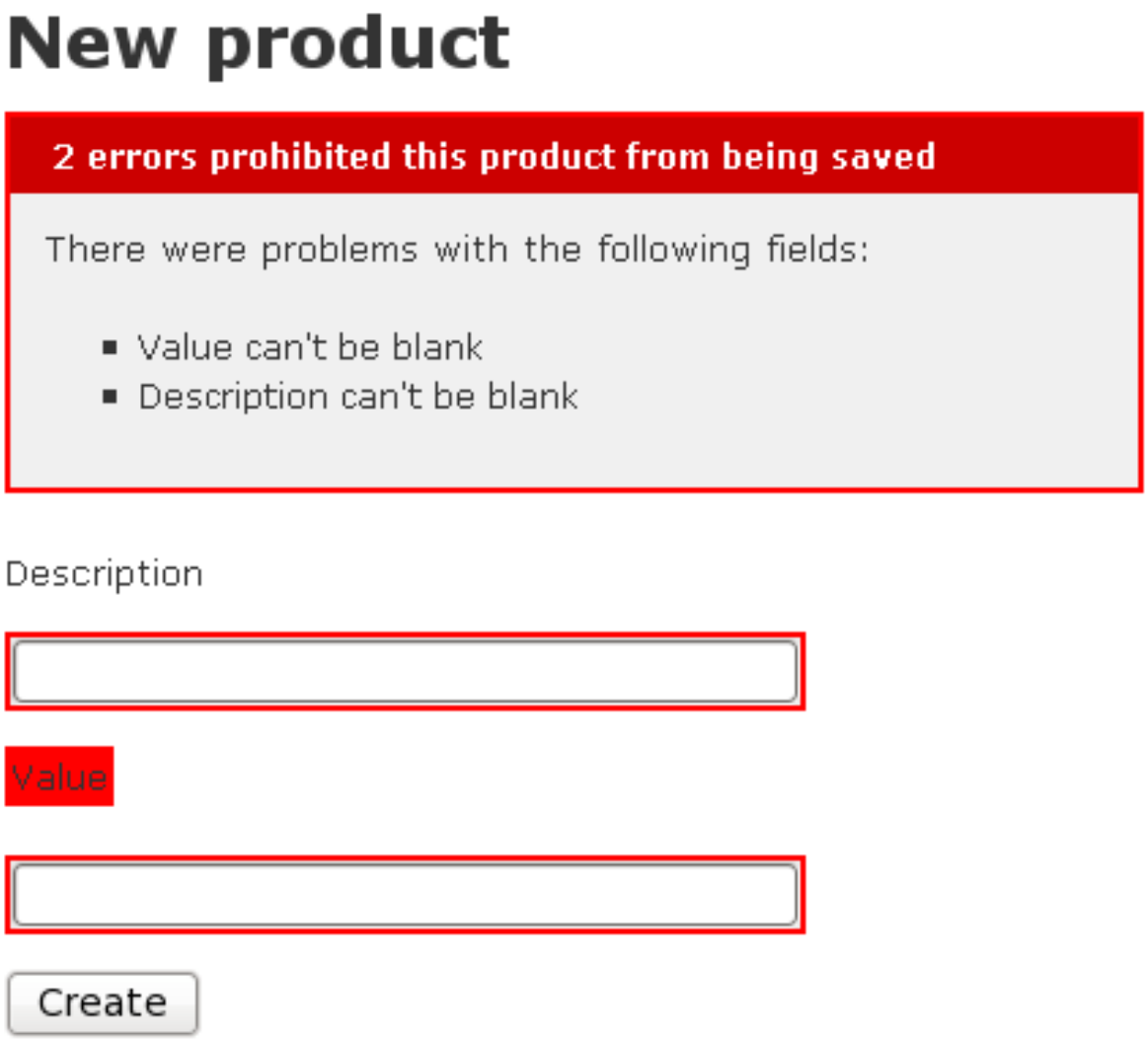
Using the errors collection in view templates I

`error_messages` method on the form builder

```
1  class Product < ActiveRecord::Base
2    validates_presence_of :description, :value
3    validates_numericality_of :value, :allow_nil => true
4  end
```

```
1  <% form_for(@product) do |f| %>
2    <%= f.error_messages %>
3    <p>
4      <%= f.label :description %><br />
5      <%= f.text_field :description %>
6    </p>
7    <p>
8      <%= f.label :value %><br />
9      <%= f.text_field :value %>
10   </p>
11   <p>
12     <%= f.submit "Create" %>
13   </p>
14 <% end %>
```

Using the errors collection in view templates II



References

- Ruby on Rails guides - http://guides.rubyonrails.org/
- Przewodniki po Ruby on Rails - http://apohllo.pl/guides/index.html