Debugging Rails; Forms and submission; redirection and the flash

COSC 480
26 February 2018

Joel Sommers
jsommers@colgate.edu
Colgate University

Warmup

If you set an instance variable in a controller method, its value will be retained for how long?

A. This request and all subsequent requests
B. This request and the next request
C. Only this request — once the view is rendered, the variable is reset to nil
D. It gets stored as an HTTP cookie, so it will persist as long as the cookies survive

Warmup

If you set an instance variable in a controller method, its value will be retained for how long?

A. This request and all subsequent requests
B. This request and the next request
C. Only this request — once the view is rendered, the variable is reset to nil
D. It gets stored as an HTTP cookie, so it will persist as long as the cookies survive

C

Debugging: Reading Ruby error messages

- The backtrace shows you the call stack (where you came from) at the stop point
- A very common message:

```
undefined method 'foo' for nil:NilClass
```

- Often, it means an assignment silently failed and you didn't check

```
@p = Product.find(id) # could be nil
```

Debugging: Instrumentation and interactive debugging

- In views

```
= debug(@product)
= @product.inspect
```

- In the log, usually from the controller method

```
logger.debug(@product.inspect)
```

- Don't use `puts` or `printf` — the output has no where to go when in production!
- Interactive debugging (like gdb, pdb, or similar)
  - Add a call to `byebug` *anywhere* in your code
    - Including in any tests!

Debugging: Use rails console

- Like `irb`, but loads Rails and your app's code
- Great for playing with models and trying things out
  - But context isn't right for peeking into controllers and views

```
$ rails console
Loading development environment (Rails 5.1.3)
irb(main):001:0> Product.find((1..2).to_a)
  Product Load (0.2ms)  SELECT "products".* FROM "products" WHE
=> [#<Product id: 1, name: "Slinky", description: nil, price: #
irb(main):002:0>
```

Checkpoint

If you put puts or printf to print out debugging messages in a production app...

A. Your app will raise an exception and die
B. Your app will continue, but the message will be lost forever
C. Your app will continue and the messages will go to the log file

Checkpoint

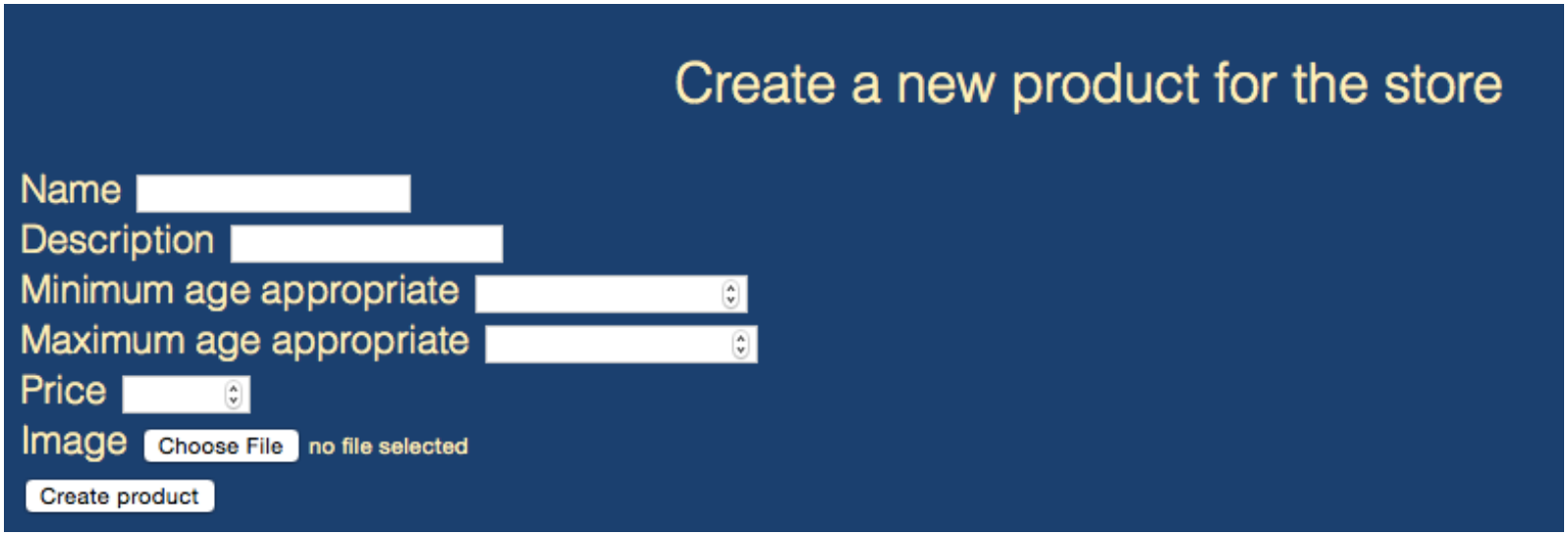
If you put puts or printf to print out debugging messages in a production app...

A. Your app will raise an exception and die
B. Your app will continue, but the message will be lost forever
C. Your app will continue and the messages will go to the log file

B.

Dealing with forms

- Creating a resource usually takes two interactions
  - `new`: retrieve/render a blank form
  - `create`: submit a filled-in form
- How to generate/display?
- How to get values filled in by user?
- What to "return" (render) after creation?
- Do a `rails routes` to see new/create



To create a new submittable form

1. Identify the action that gets the blank form (e.g., `new`)
2. Identify the action that receives the submitted form (e.g., `create`)
3. Create routes, actions, views for each
   - In form view, form element `name` attributes control how values appear in `params[]`
   - Labels for form elements help test scripts "find" the appropriate input fields
4. Generating the form
   - The action and method attributes define the route
     - Often can use URI helper for actions, since it's just the URI part of a route (still need to specify method, e.g., POST)
     - Only named form attributes will be submitted
     - Naming form fields as `product[name]`, `product[price]` means that you end up with `params[:product]` as a hash ready to pass to `create` or `update` (but this can be dangerous, too)
     - *form field helpers* generate conveniently-named form inputs

Adding 'create a new product' to toy application

- On the index page, make link to `new` route for rendering a blank form for creating a new product

```
-# in app/views/products/index.html.haml
-# probably at the bottom:
-# verify uri helper in rails routes

=link_to "Create a new product", new_product_path
```

- Next, stub out the controller method `new`
- (Can just be an empty method for now)

Making the new template: form field helpers

- http://guides.rubyonrails.org/form_helpers.html
- http://api.rubyonrails.org/classes/ActionView/Helpers/FormHelper.html
- http://api.rubyonrails.org/classes/ActionView/Helpers/FormTagHelper.html

```
-# in app/views/products/new.html.haml

=form_for Product.new do |f|
  =f.label :name
  =f.text_field :name

  =f.label :description
  =f.text_field :description

  =f.label :price
  =f.number_field :price

  =f.submit
```

Checkpoint

```
=form_for Product.new do |f|
  =f.label :name
  =f.text_field :name

  =f.label :description
  =f.text_field :description

  =f.label :price
  =f.number_field :price

  =f.submit
```

What is wrong with this view template?

A. There are no default values specified for each form field
B. The layout is going to be terrible; no CSS id's or classes have been specified
C. There's model-related code in a view
D. There's not going to be a button to reset the field values

Checkpoint

```
=form_for Product.new do |f|
  =f.label :name
  =f.text_field :name

  =f.label :description
  =f.text_field :description

  =f.label :price
  =f.number_field :price

  =f.submit
```

What is wrong with this view template?

A. There are no default values specified for each form field
B. The layout is going to be terrible; no CSS id's or classes have been specified
C. There's model-related code in a view
D. There's not going to be a button to reset the field values

In the controller method new

- The view definitely shouldn't have any model-related code in it!
- Create a new Product object (initially empty), pass it into the view as `@product`
- Even better: factor out to a helper method
  - Put in `app/helpers/products_helper.rb`
  - More on helpers in a couple weeks or so

```
# inside ProductsHelper module
  def empty_product
    Product.new
  end
```

What view should be rendered for create action?

- Idea: redirect user to a more useful page
  - e.g., product index if create was successful
  - e.g., new product form, if unsuccessful
- Redirect triggers a whole new HTTP request
  - How to inform the user *why* they were redirected?
- Solution: `flash[]` — quacks like a hash that persists until the end of the next request
  - `flash[:notice]` conventionally used for information
  - `flash[:warning]` conventionally used for errors
- Since the redirect causes a new HTTP request, we need the flash to convey information to the user

Flash and session

- `session[]`: like a hash that persists forever
  - `reset_session` nukes the whole thing
  - `session.delete(:some_key)`, to remove some key, like a hash
- By default, cookies store *entire contents* of session and flash
  - Alternative 1: store sessions in standard DB table
  - Alternative 2: store sessions in a "NoSQL" storage system, like `memcached`

Doing the create method: first try

- Controller method for create

```
def create
  p = Product.new(params[:product]) # "mass assignment" of attributes!
  if p.save
    flash[:notice] = "Product #{p.name} successfully created"
    redirect_to products_path
  else
    flash[:warning] = "Product couldn't be created"
    redirect_to new_product_path
  end
end
```

What happened?

- Mass assignment protection! (aka ForbiddenAttributesError)
- What if we were updating some user attributes and the user model had a password attribute?
  - Might be possible for someone to put a new password in the params hash (i.e., through the POST or GET data) and cause the password to be mass-assigned to
- Rails 4 introduced new mechanisms for identifying which model attributes are "whitelisted"
  - All done in the controller through `require` and `permit` methods on params

```
# as private method in ProductsController:
private
  def create_params
    # we require a product to be in params
    # allow name, description, and price to be mass-assigned
    params.require(:product).permit(:name, :description, :price)
  end
```

Change mass assignment line in create method to:

```
p = Product.new(create_params)
```

Getting the flash messages to show up

- Edit `app/views/layouts/application.html.haml`
  - If you have an ERb application layout template, use `html2haml` to convert it to haml

HTTP redirection

- When you do `redirect_to` in a controller method, the controller method continues to execute!
  - Calling `redirect_to` does not cause you to *return* from the method
- If you redirect more than once, you'll get a multiple-render error (can't render more than one template in response to one request!)
- If you want to ensure that you bail out after redirect, can do something like

```
redirect_to products_path and return  # redirect and force return
```

Checkpoint

True or false: from a controller method, you can render a view then redirect

A. True

B. False

Checkpoint

True or false: from a controller method, you can render a view then redirect

A. True

B. False

False. You can only do one or the other, not both.