**Full Stack Notes**



# Session and Life-Cycle Callbacks

HTTP, the protocol used for communication between web-servers and web-browsers is stateless. A web-server does not retain information about the user/browser between requests.

Rails includes a session mechanism to allow us to presist user-specific state over time.

We can also tap into the “life-cycle” of a Rails request to make data available.

## Table of Contents

# Persistence with Session

In the context of Rails this means that variables cannot retain their values between requests. All variables loose scope after a view is rendered.

Sessions were created to allow for data persistence across browser requests. In Rails, a session is a hash-like structure that allows a web-server to store and then restore objects across requests. Rails associates these persisted objects with specific users/browsers.

To allow us to persistent data cross browser request, Rails provides a special `session` hash.

RESOURCES

- Rails Guide on Session

# Session Example

Imagine a controller action called `remember_me`:

```
def remember_me
  if session[:count].nil?
    session[:count] = 1
  else
    session[:count] += 1
```

```
  end
  @count = session[:count]
end
```

Each time this action is executed the `@count` will increase by one. Session is tied to a specific browser by way of a browser cookie. This means that a separate count will be maintained for each user that executes this action.

## Controller Before Filters

Before filters are methods that are run before a controller action. Here's some example code.

```
class SomeController < ApplicationController
  # Run some_method before all actions:
  before_action :some_method

  # Run method_name before the index and show actions:
  before_action :method_name, only: [:index, :show]

  # Run another_method before all actions except those shown:
  before_action :another_method, except: [:edit, :update]
```

```
  # The rest of the controller actions have been omitted.
end
```

Note that `some_method`, `method_name` and `another_method` must be defined within this controller, or within the ApplicationController.rb file. All controllers inherit the methods added to the ApplicationController.

RESOURCES

- Before Filters - Rails Controller Guide

## Before Filters to Autoload From the Database

Use a `before_action` when your controllers have groups of actions that all require the same data.

For example, controllers generated using the `rails scaffold` command have member actions (`show`, `edit`, `update` & `destroy`) that all need to fetch an ActiveRecord object by id.

Imagine a `ProductsController` providing RESTful access to a `Product` model:

```
class ProductsController < ApplicationController
  # Run set_product (private method defined below) before all member actions:
  before_action :set_product, , only: [:show, :edit, :update, :destroy]


  # The show, edit, update, destroy actions, can now all use the @product fetched below.
```

```
  # For example, the show action (GET /products/1):
  def show
      # Empty method as @product is loaded by the before_action.
  end


  private
    # Used with before_action to share common setup between actions.
    def set_product
      @product = Product.find(params[:id]) # Find by the :id param of the GET route.
    end
end
```

## Before Filters for Simple Auth

Simple username/password protection can be added to a Rails project using HTTP digest authentication. In the application_controller.rb you need to add the following within the class definition:

```
private
USERS = { "admin" => "gorgonzola7!" }
def require_sudo
  authenticate_or_request_with_http_digest do |username|
    USERS[username]
```

```
  end
end
```

We can then use a before filter to protect all (or a subset of) any controller's actions from within that controller:

```
before_action :require_sudo, only: [:update, :create, :destroy]
```

RESOURCES

- HTTP Authentication Further Simplified

## Helper Methods to Preload "Global" Data

Sometimes you need the same collection of data on every page. For example, data to build the site's header menu.

You can make data available to all views by loading it in the `ApplicationController`, the parent class of all your controllers, and exposing it to all your views as a `helper_method`.

### Global `helper_method` Example: Menu Data

Imagine you have a `Menu` model with a class method `Menu.top_level_sections` that returns the data you need on every page for the header menu. You could create a private `menu_sections` method in your `ApplicationController` and expose it to your views as a `helper_method`.

```
class ApplicationController < ActionController::Base
  # Your ApplicationController should always set a forgery protection scheme.
  protect_from_forgery with: :exception

  # Expose 'global' data to all views as follows:
  private
  def menu_sections # Create a private method that loads the data you which to export.
    @menu_sections ||= Menu.top_level_sections # Memoizing as an instance var: https://bit.ly/memoize_ruby
  end

  # Make menu_sections method available in all views:
  helper_method :menu_sections
  # View usage with a partial: <%= render partial: 'menu', object: menu_sections %>
end
```