Rails Debugging / Error Handling / Gemfile Topics

Devpoint Labs - Jake Sorce / Dave Jungst

Gems To Use

- pry => used as a breakpoint in your project
- pry-byebug => step, next, up, down, continue with pry
- better_errors => replaces standard rails error page
- bullet => notifies of n+1 queries and makes suggestions
- rack-mini-profiler => shows page request times
- awesome-print => better display of objects in console & pry

pry

pry is a tool for pausing your project and allowing you to evaluate variables at a given state.

To add a breakpoint just add the line binding.pry

Typing exit will run your code until it hits the next binding.pry

pry

```
def index
  @posts = Post.all
  binding.pry
end
```

```
> @posts
  => #<ActiveRecord::Relation [ #<Post id: 1 …>, #<Post id:2 .. >]>
```

pry (cont)

```ruby
def y
  y = 0
  binding.pry
  @number = medthod_1(y)
end

def method_1(x)
  binding.pry
  @number += x
  binding.pry
end
```

```
> y
  => 0
> @number
  => nil
> exit
> x
  => 0
> @number
  => nil
> exit
> @number
  => 1
```

pry-byebug

pry-byebug is a pry plugin that gives you traditional debugger methods in pry such as

- step => move to the next line
- next => skip the next line
- up => move up the stack
- down => move down the stack
- continue => go until next breakpoint

pry (cont)

```
def y
  y = 0
  binding.pry
  @number = medthod_1(y)
end

def method_1(x)
  binding.pry
  @number += x
  binding.pry
  puts “hello world”
end
```

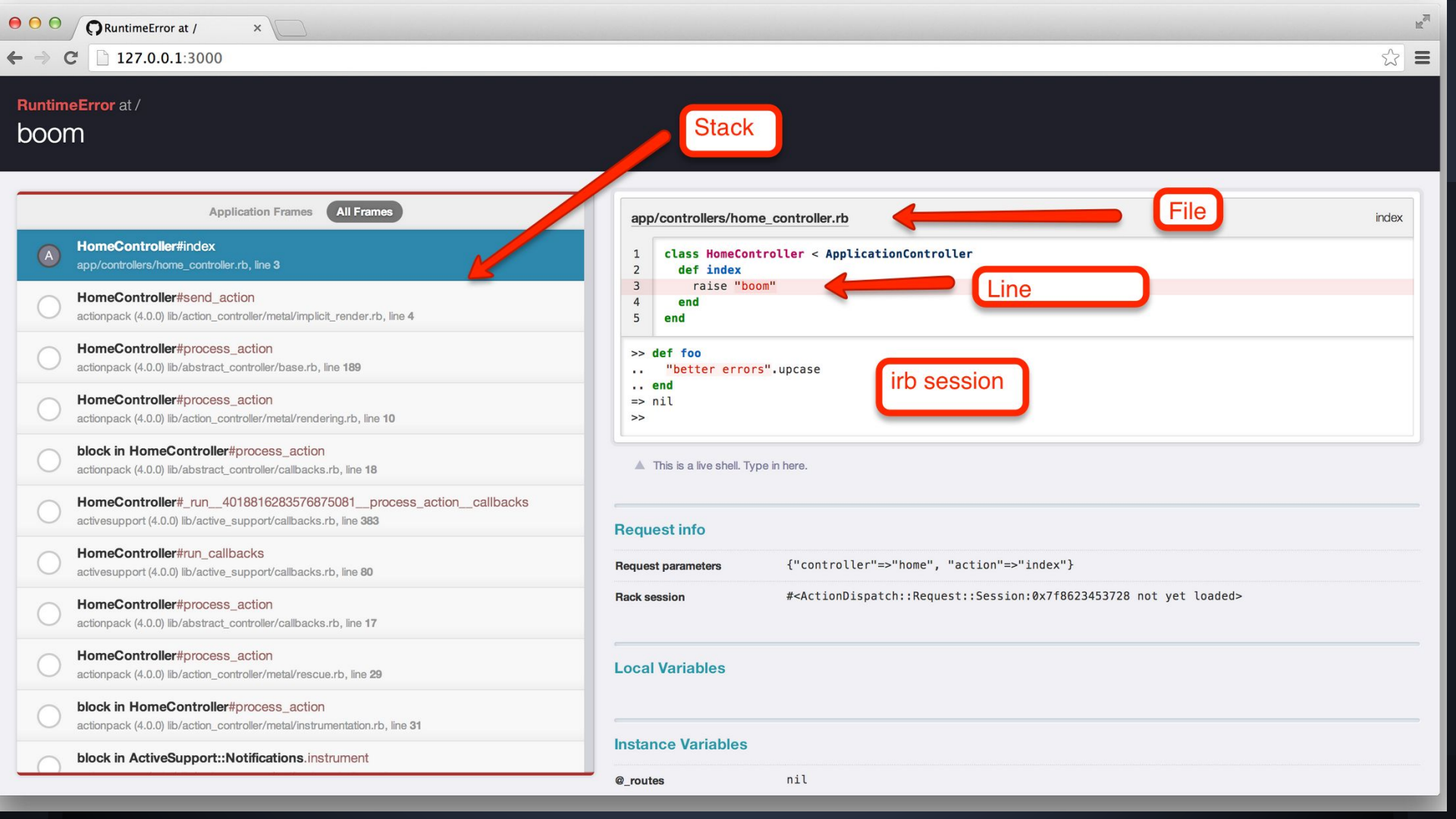
```
> step
> x
  => 0
> next
> @number
  => 1
> continue
  => “hello world”
```

better-errors

Better errors replaces the standard rails error page with one that is far easier to read and contains a live irb session paused at the point of the exception.

Better errors provides:

- The call stack
- The file where the exception occurred
- The line number (Including the code)
- A live shell
- Scoped variables and their values


RuntimeError at /
127.0.0.1:3000
RuntimeError at /
boom
Stack
Application Frames
All Frames
HomeController#index
app/controllers/home_controller.rb, line 3
HomeController#send_action
actionpack (4.0.0) lib/action_controller/metal/implicit_render.rb, line 4
HomeController#process_action
actionpack (4.0.0) lib/abstract_controller/base.rb, line 189
HomeController#process_action
actionpack (4.0.0) lib/action_controller/metal/rendering.rb, line 10
block in HomeController#process_action
actionpack (4.0.0) lib/abstract_controller/callbacks.rb, line 18
HomeController#_run__4018816283576875081__process_action__callbacks
activesupport (4.0.0) lib/active_support/callbacks.rb, line 383
HomeController#run_callbacks
activesupport (4.0.0) lib/active_support/callbacks.rb, line 80
HomeController#process_action
actionpack (4.0.0) lib/abstract_controller/callbacks.rb, line 17
HomeController#process_action
actionpack (4.0.0) lib/action_controller/metal/rescue.rb, line 29
block in HomeController#process_action
actionpack (4.0.0) lib/action_controller/metal/instrumentation.rb, line 31
block in ActiveSupport::Notifications.instrument
app/controllers/home_controller.rb
File
index
1 class HomeController < ApplicationController
2 def index
3 raise "boom"
4 end
5 end
Line
>> def foo
.. "better errors".upcase
.. end
=> nil
>>
irb session
This is a live shell. Type in here.
Request info
Request parameters
{"controller"=>"home", "action"=>"index"}
Rack session
#<ActionDispatch::Request::Session:0x7f8623453728 not yet loaded>
Local Variables
Instance Variables
@_routes
nil

bullet

Bullet notifies you of n + 1 queries and makes suggestions to fix them. This is a great gem to find out if you have slow queries in your code.

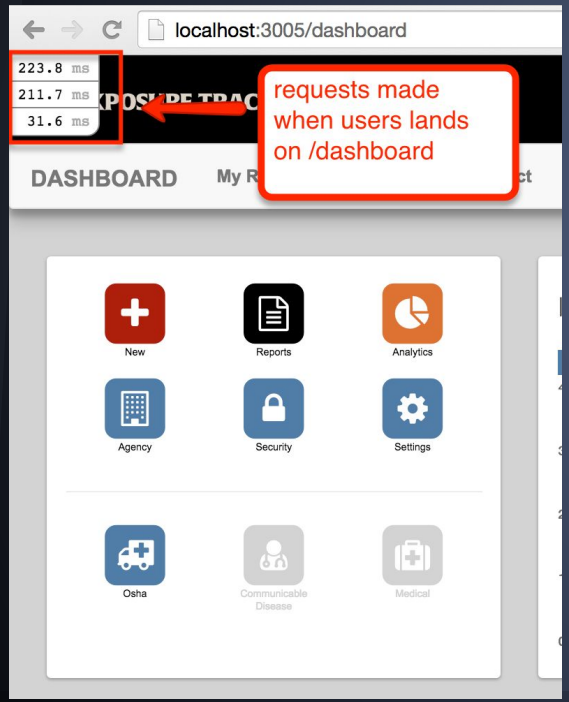
This will become far more useful as your queries become more complex.

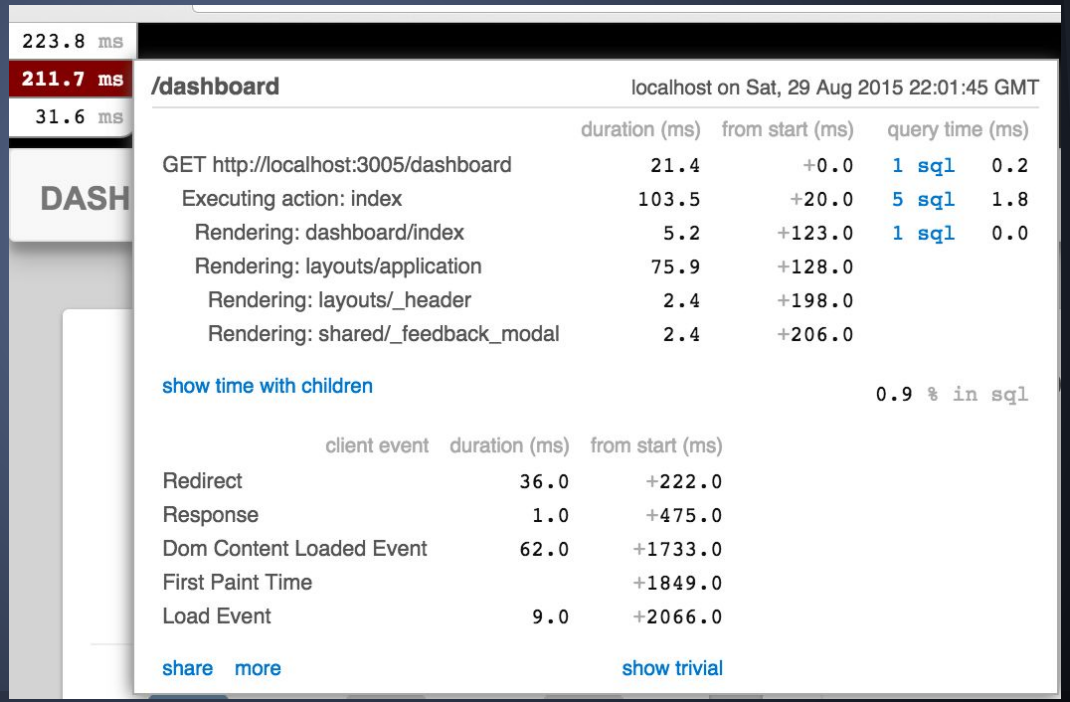
rack-mini-profiler

Rack mini profiler will put request time tabs in the upper left corner of your application. Clicking on one will give you more information about requests.

This is a great way to identify slow requests in your code.

rack-mini-profiler





awesome_print

Awesome print formats objects in console and pry to be more readable

BEFORE:

```
> @people
  => #<ActiveRecord::Relation [#<Person id: 1, …>, #<Person id: 2, …>]>
```

awesome_print

After:

```
> ap @people
[
  [0] #<Person:0x007f8af6239ff0> {
          :id => 1,
        :name => "Jake"
    },
   [1] #<Person:0x007f8afaa4ee30> {
           :id => 2,
         :name => "Dave",
    }
 ]
```

Gemfile Groups

Gemfile groups are good for organizing your gems for different environments:

```
group :production do
  gem 'passenger'
  gem 'rails_12factor'
  gem 'paper_trail'
end

group :test
  gem 'vcr'
  gem 'factory_girl_rails'
end

group :development, :test do
  # Call 'byebug' anywhere in the code to stop execution and get a debugger console
  gem 'byebug'

  # Access an IRB console on exception pages or by using <%= console %> in views
  gem 'web-console', '~> 2.0'

  # Spring speeds up development by keeping your application running in the background.
  gem 'spring'
end
```

- bundle install --without test
  - installs all gems in the Gemfile except for the ones in the test group
- bundle install --without development
  - installs all gems in the Gemfile except for the ones in the development group
- bundle install --without test development
  - installs all gems in the Gemfile except for the ones in the test and development groups

Pinning Gems

```
source 'https://rubygems.org'
ruby '2.2.0'

# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.0.13'

# Use SCSS for stylesheets
gem 'sass-rails', '~> 4.0.2'

# Use Less
gem 'less-rails', '~> 2.6.0'

# Bootstrap 3
gem 'bootstrap-sass', '~> 3.2.0'

# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'

# Use CoffeeScript for .js.coffee assets and views
gem 'coffee-rails', '~> 4.0.0'

# Use Postgres
gem 'pg', '~> 0.18.1'

# Use Haml
gem 'haml-rails', '~> 0.7.0'

# Use jquery as the JavaScript library
gem 'jquery-rails', '~> 3.1.2'

# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 1.2'

# Ruby Racer V8 Runtime
gem "therubyracer", '~> 0.12.1'

# Use Font Awesome
gem "font-awesome-rails", '~> 4.3.0.0'

# Use Devise
gem 'devise', '~> 3.4.1'
```

Pinning gems is a good habit to get into. When you pin a gem it tells your Gemfile to only install the version specified.

The reason why we pin gems is: maintained gems are always being developed and new versions are being pushed. New versions could cause breaking changes.

There are a few ways to pin gems in your Gemfile:

- gem ‘pg’, ‘0.18.1’
  - pins the pg gem to the exact version
- gem ‘pg’, ‘~> 0.18.1’
  - pins the pg gem to any newer version that isn’t 1
- gem ‘pg’, ‘<= 0.18.1’
  - pins the pg gem to any newer version that is below 0.18.1

Mini Project Time

Dave and I have created a really really broken rails project. We want you to use your new knowledge of debugging and tools to fix this project.

Basic Goals:

1. Get with a partner
2. Pick one person on the team to go to put this in their browser:

   https://github.com/jakesorce/code-smells-basic-example
3. Click the fork button on the top right
4. Click the Github (Cat) Icon on the top left to go to the dashboard
5. Click the forked repository
6. Add your partner/s as collaborators
7. Have one person fix a problem and push
8. The next person should fix a problem and push, ect...
9. Fix the entire project so it works

Mini Project Time - Advanced Goals

Advanced Goals:

- DRY the project up after it is fixed
- Add missing functionality
- Make a pull request to our github repo so we can code review your fixes