What's new in Rails 4

Maricris Nonato / @maricris_sn

20.02.2015

4.0

Upgrades

- Ruby 1.8.x is history; Ruby 2.0 is preferred, but Ruby 1.9.3+ is required

- vendor/plugins completely removed in favor of managing your gems via Gemfile and Bundler

- some modules moved to gems: ActiveRecord observers, ActiveRecord session store (no longer recommended), ActiveResource, etc

ActionPack

- Strong parameters: allow only whitelisted parameters for update operations
- vendor/plugins completely removed in favor of managing your gems via Gemfile and Bundler
- some modules moved to gems: ActiveRecord observers, ActiveRecord session store (no longer recommended), ActiveResource, etc
- Routing concerns: allows you to refactor your routes via *concerns*
- Russian doll caching
- Turbolinks

Strong Parameters

```ruby
class EventsController < ApplicationController

  def new
    @event = Event.new
  end

  def create
    @event = Event.new(event_params)

    if @event.save
      #..do something..
    end
  end

  private

    def event_params
      params.require(:event).permit(:title, :description, :start_date, :end_date)
    end

end
```

Routing concerns

from this:

```ruby
YourApp::Application.routes.draw do
  resources :calendar_events do
    get :past, on: :collection
    resources :comments
  end

  resources :messages  { resources :comments }
  resources :forwards  { resources :comments }
  resources :uploads   { resources :comments }
  resources :documents { resources :comments }
  resources :todos     { resources :comments }
end
```

to this:

```ruby
YourApp::Application.routes.draw do
  concern :commentable do
    resources :comments
  end

  resources :messages, :forwards, :uploads, :documents, :todos, concerns: :commentable
end

```

https://github.com/rails/routing_concerns

Russian doll caching

- Cache nested fragments of views. Each fragment expires based on a set of dependencies (a cache key).
- The cache key is usually a template version number and a model object.
- No need to worry about observers or when to expire

Russian doll caching

cache like this in your view:

```
1   - cache @user do
2   (user view data)
3
4     - cache [ 'details', @user ] do
5     (user details view data)
6
7     - cache [ 'characters', @user ] do
8       - @user.characters.each do |character|
9
10        - cache character do
11          (character view data)
```

touch parent record
is a requirement:

```
1   class Character < ActiveRecord::Base
2     belongs_to :user, touch: true
3   end
```

and create caches like these:

```
views/users/3-20130530135425/7a1bb8bb15b02ee7aa69cec1d5f6f630
views/details/users/3-20130530135425/6f28ec6d31e7e3b73a575777d59e63ca
```

Turbolinks

- Serve only one initial HTML page. When the user navigates to another page, use pushState to update the URL and use AJAX to update the title and body.
- May not work properly for those pages which does post pageload queries to update some DOMs

Turbolinks

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page title</title>
    <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track' => true %>
    <%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
  </head>

  <body>
    <table data-source="<%= contacts_url(section: section, format: "json") %>" id="contacts">
      <thead>
        <tr>
          <th width="20%">Full Name</th>
          <th width="20%">Title</th>
          <th width="20%">Organization</th>
          <th width="20%">Email</th>
          <th width="10%">Phone</th>
          <th width="10%">Mobile</th>
        </tr>
      </thead>
    </table>

    <script>
      $(document).ready(function(){
        $('.table-primary').dataTable({
          "aaSorting": [[ 0, "asc" ]],
          "aoColumnDefs": [
            { 'bSortable': false, 'aTargets': [5] }
          ],
          pagingType: 'full_numbers',
          processing: true,
          serverSide: true,
          sAjaxSource: $('#contacts').data('source'),
          fnRowCallback: function(nRow, aData, iDisplayIndex) {
            nRow.setAttribute('id', 'contact-' + aData[7]);
          }
        });
      });
    </script>
  </body>
</html>
```

Others

- PATCH verb as part of HTTP actions to specify partial update of resource
- match in routes requires the verbs to be specified
- strings rendered in erb are now escaped unless specified by raw or html_safe is called

4.1

Spring Application PreLoader

```
$ bin/spring status
Spring is running:

 1182 spring server | my_app | started 29 mins ago
 3656 spring app    | my_app | started 23 secs ago | test mode
 3746 spring app    | my_app | started 10 secs ago | development
mode
```

config/secrets.yml

- was initially used to store secret key base of your application
- can be used to keep environment variables
- other options are to use dot_env (gem), figaro (gem), etc.

Action Pack Variants

```
request.variant = :tablet if request.user_agent =~ /iPad/
```

Respond to variants in the action just like you respond to formats:

```
respond_to do |format|
  format.html do |html|
    html.tablet # renders app/views/projects/show.html+tablet.erb
    html.phone { extra_setup; render ... }
  end
end
```

Provide separate templates for each format and variant:

```
app/views/projects/show.html.erb
app/views/projects/show.html+tablet.erb
app/views/projects/show.html+phone.erb
```

You can also simplify the variants definition using the inline syntax:

```
respond_to do |format|
  format.js          { render "trash" }
  format.html.phone { redirect_to progress_path }
  format.html.none   { render "trash" }
end
```

ActionMailer Previews

- Useful for verifying mail layouts
- Still lovin' letter_opener (gem) though!

```
class NotifierPreview < ActionMailer::Preview
  def welcome
    Notifier.welcome(User.first)
  end
end
```

The preview is available in http://localhost:3000/rails/mailers/notifier/welcome, and a list of them in http://localhost:3000/rails/mailers.

By default, these preview classes live in `test/mailers/previews`. This can be configured using the `preview_path` option.

ActiveRecord enums

```ruby
class Conversation < ActiveRecord::Base
  enum status: [ :active, :archived ]
end

# conversation.update! status: 0
conversation.active!
conversation.active? # => true
conversation.status  # => "active"

# conversation.update! status: 1
conversation.archived!
conversation.archived? # => true
conversation.status    # => "archived"

# conversation.update! status: 1
conversation.status = "archived"

# conversation.update! status: nil
conversation.status = nil
conversation.status.nil? # => true
conversation.status      # => nil
```

- Useful for putting “names” as identifiers to fields stored as integers
- This makes pretty convenience methods like “?” to the name of your enum field

Module#concerning

```ruby
class Event < ActiveRecord::Base
  concerning :RsvpEvent do
    included do
      has_many :rsvps
    end

    def last_rsvp
      ...
    end

    private
      def some_internal_method
        ...
      end
  end

  concerning :DinnerEvent do
    def menu_choices
      ...
    end
  end
end
```

- Elegant way of isolating responsibilities within a class
- Clear way of chopping your class into pieces that can still be reused or extended as a *concern* for another class

4.2

Active Job

- provides you the ease of a fixed convention regardless of what queuing framework you are using (Resque, Delayed Job, Sidekiq, etc)
- organize jobs within your app folder
- create jobs and enqueue them with ease
- ability to save on effort to pack/unpack your job parameters

Active Job

```ruby
# app/jobs, lib/jobs
module Jobs
  class ShopGeocoder
    @queue = :geocoding

    def self.perform(shop_id)
      if !shop_id.nil?
        shop = Shop.find shop_id
        shop.geocode
      end
    end
  end
end
```

```ruby
class ShopGeocoder < ActiveJob::Base
  def perform(shop)
    shop.geocode
  end
end
```

https://github.com/rails/globalid

Active Job

creating jobs

```
$ bin/rails generate job shop_geocoder --queue urgent
invoke  test_unit
create  test/jobs/guests_cleanup_job_test.rb
create  app/jobs/guests_cleanup_job.rb
```

enqueuing jobs

```
# Enqueue a job to be performed as soon the queueing system is
# free.
MyJob.perform_later record
```

```
# Enqueue a job to be performed tomorrow at noon.
MyJob.set(wait_until: Date.tomorrow.noon).perform_later(record)
```

```
# Enqueue a job to be performed 1 week from now.
MyJob.set(wait: 1.week).perform_later(record)
```

Active Job

setting your desired queuing framework

```ruby
# config/application.rb
module YourApp
  class Application < Rails::Application
    # Be sure to have the adapter's gem in your Gemfile
    # and follow the adapter's specific installation
    # and deployment instructions.
    config.active_job.queue_adapter = :sidekiq
  end
end
```

Active Job

cleaner way of handling exceptions

```ruby
class ShopGeocodeJob < ActiveJob::Base
  queue_as :default

  rescue_from(ActiveRecord::RecordNotFound) do |exception|
   # do something with the exception
  end

  def perform
    # Do something later
  end
end
```

http://edgeguides.rubyonrails.org/active_job_basics.html

ActionMailer's deliver_later

- handle mails asynchronously via its own queue
- you can choose to send immediately (deliver_now), or
- later via deliver_later

ActionMailer's deliver_later

```ruby
# If you want to send the email now use #deliver_now
UserMailer.welcome(@user).deliver_now

# If you want to send the email through Active Job use #deliver_later
UserMailer.welcome(@user).deliver_later
```

Adequate Record

- From @tenderlove: ***“TL;DR: AdequateRecord is a set of patches that adds cache stuff to make ActiveRecord 2x faster”***

- this add-on caches *simple* queries; but doesn’t work on complicated ones

Adequate Record

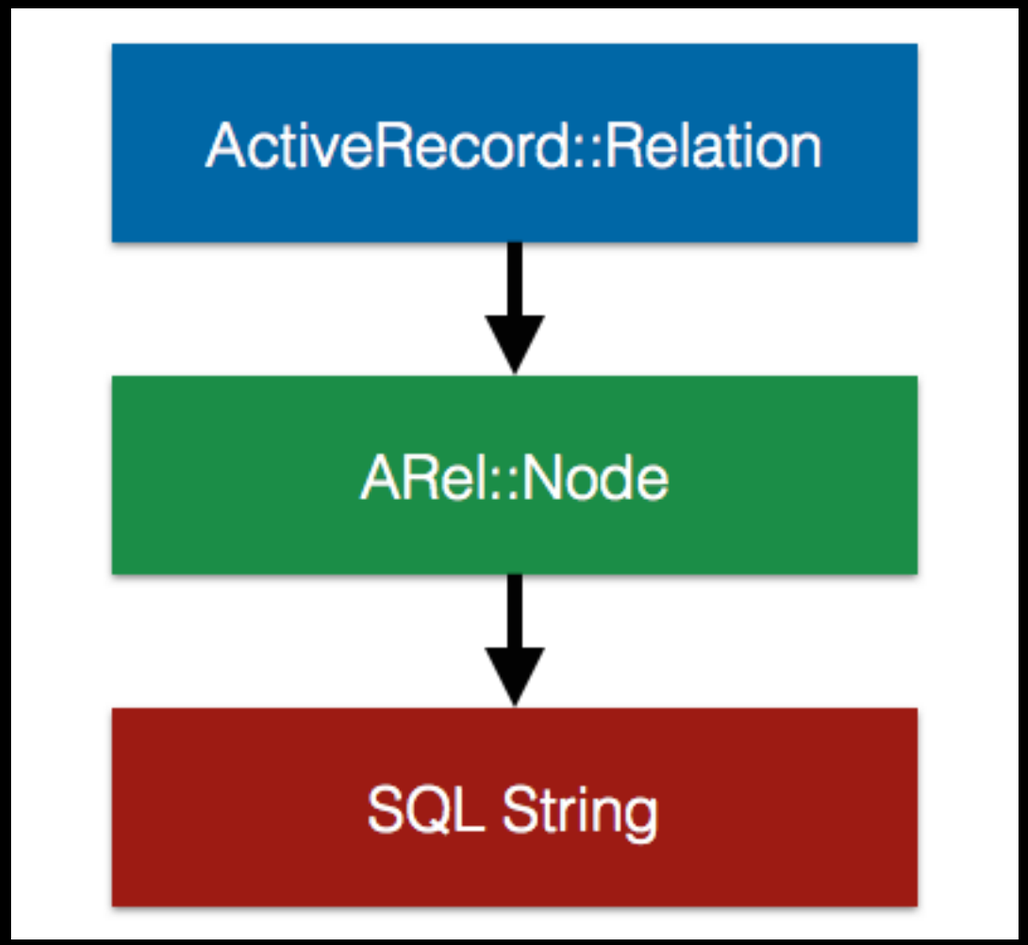
works with:

```
1  Post.find(id)
2  Post.find_by_name(name)
3  Post.find_by(name: name)
```

not yet compatible with:

```
1  Post.find(1, 2, 3)
2  Post.find([1,2])
3  Post.find_by('published_at < ?', 2.weeks.ago)
4  Post.where(...).where(...).order(..)
5
```

Adequate Record

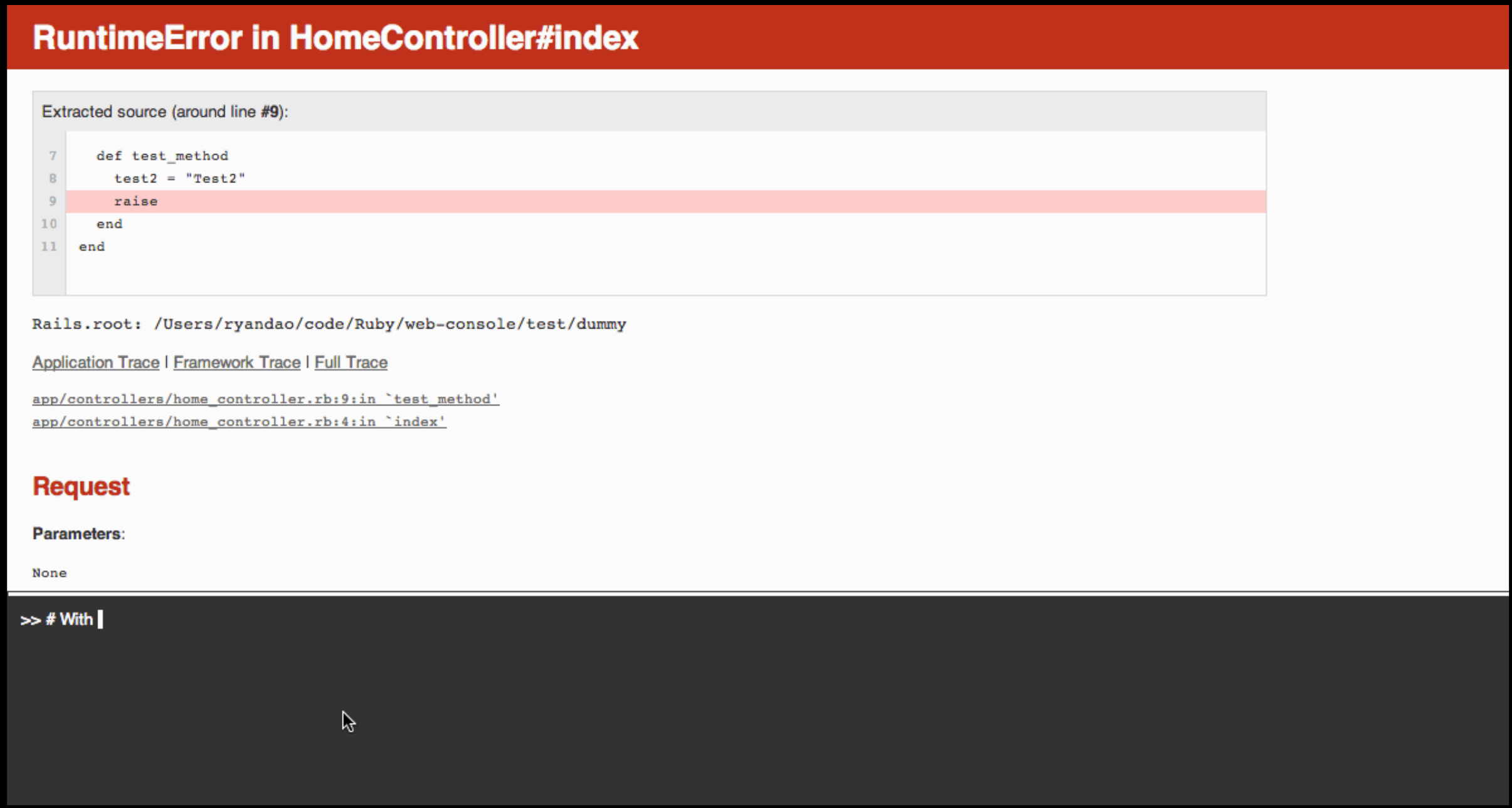


Web Console

- a gem that was built into the Rails framework
- a project from Google Summer of Code Rails campaign
- puts a web-console in your error page

http://weblog.rubyonrails.org/2014/5/23/meet-our-google-summer-of-code-students-and-mentors/

Web Console



https://github.com/rails/web-console

Foreign Key Support

- add / remove foreign keys to your migration
- almost similar to “references" in your migration, but with more flexibility
- works with MySQL and PostgreSQL adapters only

Foreign Key Support

how references work:

```
rails g migration AddUserToUpload user:references



# Which creates migration file as:
class AddUserToUpload < ActiveRecord::Migration
  def change
    add_reference :uploads, :user, index: true
  end
end
```

how foreign keys work:

```
# add a foreign key to `articles.author_id` referencing `authors.id`
add_foreign_key :articles, :authors

# add a foreign key to `articles.author_id` referencing `users.lng_id`
add_foreign_key :articles, :users, column: :author_id, primary_key: "lng_id"

# remove the foreign key on `accounts.branch_id`
remove_foreign_key :accounts, :branches

# remove the foreign key on `accounts.owner_id`
remove_foreign_key :accounts, column: :owner_id
```

Foreign Key Support

closer look:

Creating a simple foreign key

```
add_foreign_key :articles, :authors
```

generates:

```
ALTER TABLE "articles" ADD CONSTRAINT articles_author_id_fk FOREIGN KEY ("author_id") REFERENCES "authors" ("id")
```

Creating a foreign key on a specific column

```
add_foreign_key :articles, :users, column: :author_id, primary_key: "lng_id"
```

generates:

```
ALTER TABLE "articles" ADD CONSTRAINT fk_rails_58ca3d3a82 FOREIGN KEY ("author_id") REFERENCES "users" ("lng_id")
```

Creating a cascading foreign key

```
add_foreign_key :articles, :authors, on_delete: :cascade
```

generates:

```
ALTER TABLE "articles" ADD CONSTRAINT articles_author_id_fk FOREIGN KEY ("author_id") REFERENCES "authors" ("id") ON DELETE CASCADE
```