The Architecture of PicCollage Server

http://godfat.org/slide/2013-01-08-PicCollage.pdf

Table of Contents

Table of Contents

Table of Contents

PicCollage,
Software
&
Services

Ruby Codes

- Gemfile
- Procfile
- Rakefile
- rainbows.rb
- concurrency.rb
- config.ru

Ruby Codes (Cont.)

- server.rb
- error.rb
- eagerload.rb
- newrelic.rb
- shell.rb
- waiter.rb

Ruby Codes (Cont.) (Cont.)

- image_util.rb
- run_later.rb
- /search
- /create_and_share

Miscellaneous Extras

- crazy inspect
- who cares backtrace for 404
- assets pipeline and
  database connection
- fiber stack overflow
- fiber database connection pool
- thread-safe kaminari patch

PicCollage,
Software
&
Services

PicCollage

- Make collages of your photos
- Import photos from places
- Touch gestures to rotate, resize, delete
- Double-tap to edit, clip, etc
- Clip photos by outlining the area
- Lots of backgrounds and stickers

PicCollage,
Software
&
Services

Software

- Ruby ~~1.8.7~~ -> 1.9.2 -> 1.9.3

Software

- Ruby ~~1.8.7~~ -> 1.9.2 -> 1.9.3
- Rails ~~3.1.0.rc4~~ -> ~~3.1.0.rc5~~ -> 3.1.0 -> 3.1.3 -> 3.2.1 -> 3.2.2 -> 3.2.3 -> 3.2.5 -> 3.2.6 -> 3.2.7 -> 3.2.8 -> 3.2.9 -> 3.2.10

Software

- Ruby ~~1.8.7~~ -> 1.9.2 -> 1.9.3
- Rails ~~3.1.0.rc4~~ -> ~~3.1.0.rc5~~ -> 3.1.0 -> 3.1.3 -> 3.2.1 -> 3.2.2 -> 3.2.3 -> 3.2.5 -> 3.2.6 -> 3.2.7 -> 3.2.8 -> 3.2.9 -> 3.2.10
- ~~Sinatra~~ -> Jellyfish

Software

- Ruby ~~1.8.7~~ -> 1.9.2 -> 1.9.3
- Rails ~~3.1.0.rc4~~ -> ~~3.1.0.rc5~~ -> 3.1.0 -> 3.1.3 -> 3.2.1 -> 3.2.2 -> 3.2.3 -> 3.2.5 -> 3.2.6 -> 3.2.7 -> 3.2.8 -> 3.2.9 -> 3.2.10
- ~~Sinatra~~ -> Jellyfish
- ~~Thin~~ -> Zbatery

Software

- Ruby ~~1.8.7~~ -> 1.9.2 -> 1.9.3
- Rails ~~3.1.0.rc4~~ -> ~~3.1.0.rc5~~ -> 3.1.0 -> 3.1.3 -> 3.2.1 -> 3.2.2 -> 3.2.3 -> 3.2.5 -> 3.2.6 -> 3.2.7 -> 3.2.8 -> 3.2.9 -> 3.2.10
- ~~Sinatra~~ -> Jellyfish
- ~~Thin~~ -> Zbatery
- ~~Resque~~ -> EM.next_tick or Thread.new -> Sidekiq (?)

PicCollage,
Software
&
Services

Services

- Github

Services

- Github
- Heroku ~~Bamboo~~ -> Cedar

Services

- Github
- Heroku ~~Bamboo~~ -> Cedar
- Heroku PostgreSQL 9.0 -> 9.2

Services

- Github
- Heroku ~~Bamboo~~ -> Cedar
- Heroku PostgreSQL 9.0 -> 9.2
- ~~MongoHQ~~ -> ~~MongoLab~~

Services

- Github
- Heroku ~~Bamboo~~ -> Cedar
- Heroku PostgreSQL 9.0 -> 9.2
- ~~MongoHQ~~ -> ~~MongoLab~~
- ~~RedisToGo~~ -> OpenRedis

Services

- Github
- Heroku ~~Bamboo~~ -> Cedar
- Heroku PostgreSQL 9.0 -> 9.2
- ~~MongoHQ~~ -> ~~MongoLab~~
- ~~RedisToGo~~ -> OpenRedis
- Memcache (?) -> Memcachier

Services

- Github
- Heroku ~~Bamboo~~ -> Cedar
- Heroku PostgreSQL 9.0 -> 9.2
- ~~MongoHQ~~ -> ~~MongoLab~~
- ~~RedisToGo~~ -> OpenRedis
- Memcache (?) -> Memcachier
- NewRelic

Services

- Github
- Heroku ~~Bamboo~~ -> Cedar
- Heroku PostgreSQL 9.0 -> 9.2
- ~~MongoHQ~~ -> ~~MongoLab~~
- ~~RedisToGo~~ -> OpenRedis
- Memcache (?) -> Memcachier
- NewRelic
- Exceptional

Services

- Github
- Heroku ~~Bamboo~~ -> Cedar
- Heroku PostgreSQL 9.0 -> 9.2
- ~~MongoHQ~~ -> ~~MongoLab~~
- ~~RedisToGo~~ -> OpenRedis
- Memcache (?) -> Memcachier
- NewRelic
- Exceptional
- SendGrid -> Mailgun

Ruby Codes

- Gemfile
- Procfile
- Rakefile
- rainbows.rb
- concurrency.rb
- config.ru

Gemfile

```
ruby '1.9.3'
source 'http://rubygems.org'

gem 'rails', '3.2.9'
gem 'jellyfish'

gem 'pg', :platform => :ruby
gem 'redis-objects'
gem 'dalli'
```

Gemfile (Cont.)

```
# service
gem 'newrelic_rpm', :require => false
gem 'exceptional'
gem 'apns', :github => 'jpoz/APNS'
    # ^^^^ needs feedback patch


# server
platform :ruby do
  gem 'zbatery', :require => false
  gem 'em-http-request'
  gem 'yajl-ruby'
end
```

Gemfile (Cont.)

```
# upload
gem 'paperclip' # needs various patches
gem 'aws-sdk'

# plist
gem 'nokogiri'

# pagination
gem 'kaminari' # needs thread-safe patch

# javascript
gem 'jquery-rails'
```

Gemfile (Cont.)

```
# make `rails server` work for zbatery
gem 'rack-handlers'

# respect rack logger
gem 'rack-rails-logger'

# redirect from www to root domain
gem 'rack-rewrite'

# cache for API
gem 'rack-cache'
```

Gemfile (Cont.)

```
# api client
gem 'rest-core'
gem 'rest-more'

# bluebase
gem 'bb_more'    , :path => 'bluebase'
gem 'bb_locales', :path => 'bluebase'
gem 'bb_auth'    , :path => 'bluebase'
gem 'bb_redis'   , :path => 'bluebase'
```

Gemfile (Cont.)

```
group :assets do
  gem 'sass-rails'
  gem 'coffee-rails'
  gem 'uglifier'
end

group :cedar do
  gem 'rib'
  gem 'bond'
  gem 'readline_buffer',
      :platform => :ruby
end
```

Gemfile (Cont.)

```
group :test do
  gem 'minitest'
  gem 'rr'
  gem 'webmock'
  gem 'kramdown'
end


group :test do
  gem 'capybara'
  gem 'rspec'
  gem 'rspec-rails'
end
```

Ruby Codes

- Gemfile
- Procfile
- Rakefile
- rainbows.rb
- concurrency.rb
- config.ru

Procfile

```
web: ruby -r bundler/setup -S zbatery
-c config/rainbows.rb
-p $PORT -E $RACK_ENV
```

Ruby Codes

- Gemfile
- Procfile
- Rakefile
- rainbows.rb
- concurrency.rb
- config.ru

Rakefile

```
task 'test:all' => ['test',
                    'test:bluebase']

task 'env:test' do
  Rails.env = 'test'
end
```

Rakefile (Cont.)

```
task 'test:bluebase' => ['env:test'] do
  # remove bundler shit
  gemfile = ENV.delete('BUNDLE_GEMFILE')
  sh './bin/bluebase-test'
  ENV['BUNDLE_GEMFILE'] = gemfile
end
```

Rakefile (Cont.)

```
task :test => ['test:api']
```

Rakefile (Cont.)

```
task 'test:api' => ['env:test',
                    'environment'] do
  require './config/environments/test'
  # force rails loaded, otherwise it's
  # causing too much loading troubles,
  # by rails' autoloading mechanism
  Dir['./test/api/*.rb'].each do |test|
    require test
  end
  # ...
end
```

Rakefile(Cont.)

```
# this is only available in 1.9.3+
if Test::Unit.const_defined?(:RunCount)
  Test::Unit::RunCount.run_once{
    status = Test::Unit::Runner.new.run
    exit(status) if status != 0
  }
end
```

Ruby Codes

- Gemfile
- Procfile
- Rakefile
- rainbows.rb
- concurrency.rb
- config.ru

rainbows.rb

```
worker_processes 2 # assume 2 CPU cores
preload_app        true

l = ::Logger.new($stderr)
l.level = case ENV['RACK_ENV']
          when 'production'
            ::Logger::WARN
          else
            ::Logger::DEBUG
          end
logger l
```

rainbows.rb (Cont.

```
Rainbows! do
  use :EventMachine, :em_client_class =>
    lambda{
      require 'concurrency'
      client = Concurrency.
        eventmachine_client
      l.info("Using #{client}")
      client
    }
  worker_connections 64
end
```

rainbows.rb (Cont.

```
before_fork do |_, _|
  # don't hold connections on master
  unless defined?(Zbatery)
    l.info("Discon' PSQL and Redis...")
    AR::Base.connection.disconnect!
    Redis.current.quit

  end
end
```

rainbows.rb (Cont.

```
after_fork  do |_, _|
  #       hold connections on workers
  unless defined?(Zbatery)
    l.info("Conn' PSQL and Redis...")
    Redis.current.connect
    AR::Base.connection.
      establish_connection
  end
end
```

rainbows.rb (Cont.

```
EM.error_handler do |e|
  puts "Err: #{e.inspect}" \
       " #{e.backtrace.inspect}"
  begin
    ::Exceptional::Catcher.handle(e)
    ::NewRelic::Agent.instance.
      error_collector.notice_error(e)
  rescue Exception => e
    puts "Err: Exceptional/NewRelic:" \
         " #{e.inspect}" \
         " #{e.backtrace.inspect}"
  end
end
```

rainbows.rb (Cont.

Rainbows! EventMachine Thread Client

https://gist.github.com/4451322
https://github.com/godfat/rainbows/pull/2

rainbows.rb (Cont.

```
class ::RainbowsEMThreadPoolClient <
        Rainbows::EventMachine::Client
  def app_call input
    @deferred = true
    EM.defer do
      # Call the application here!
    end
  end
end
```

rainbows.rb (Cont.

```
# Call the application here!
begin
  @env.merge!(RACK_DEFAULTS)
  status, headers, body = APP.call(@env)
  @deferred = nil # EM.next_tick?
  ev_write_response(status,
    headers, body, @hp.next?)
rescue Exception => e
  EM.instance_variable_get(
    :@error_handler).call(e)
  handle_error(e)
end
```

Ruby Codes

- Gemfile
- Procfile
- Rakefile
- rainbows.rb
- concurrency.rb
- config.ru

concurrency.rb

```
module Concurrency
  Model = Thread
  module_function
  def eventmachine_client
    if     Model == Thread
      RainbowsEMThreadPoolClient
    elsif Model == Fiber
      RainbowsEMFiberSpawnClient
    else
      Rainbows::EventMachine::Client
    end
  end
end
```

concurrency.rb(Con

```
module Concurrency
  def wrap
    if     Model == Thread
      EM.defer do
        # ...
      end
    elsif Model == Fiber
      Fiber.new{ yield }.resume
    else
      yield
    end
  end
```

concurrency.rb(Con

```
def wrap
  begin
    yield
  rescue Exception => e
    EM.instance_variable_get(
      :@error_handler).call(e)
  ensure
    AR::Base.clear_active_connections!
  end
```

Ruby Codes

- Gemfile
- Procfile
- Rakefile
- rainbows.rb
- concurrency.rb
- config.ru

config.ru

```
use Api::HostDispatcher, Api::Server

map '/api' do
  run Api::Server
end

map '/'     do
  run PicCollage::Application
end
```

Ruby Codes (Cont.)

- server.rb
- error.rb
- eagerload.rb
- newrelic.rb
- shell.rb
- waiter.rb

server.rb

```ruby
Api::Server = Rack::Builder.new do
  # Embrace Rack is the Jellyfish way
end
```

server.rb (Cont.)

```
use Rack::CommonLogger, debug_logger
use Rack::Deflater
use Rack::Static,
  :urls => ['/images'], :root =>
  "#{File.dirname(__FILE__)}/public"

use AR::ConnectionAdapters::
    ConnectionManagement
```

server.rb (Cont.)

```
use Api::MiddleError
use Api::MiddleDevice
use Api::MiddleAuth
use Api::MiddleCache
use Rack::Cache, {} # [snipped]
run Api::ServerCore.new
```

server.rb (Cont.)

```
class Api::ServerCore
  include Jellyfish
  handle_exceptions false
  def controller; Controller; end

  class Controller < Api::Controller
    include Api::ServerMethods
    include Jellyfish::NewRelic
  end
end
```

server.rb (Cont.)

```
class Api::ServerCore
  post '/collages/create_and_share' do
    render_collage create_collage{ |c|
      transfer "/collages/#{c.id}/share"
    }
  end
end
```

Ruby Codes (Cont.)

- server.rb
- error.rb
- eagerload.rb
- newrelic.rb
- shell.rb
- waiter.rb

error.rb

```ruby
class Api::MiddleError
  include Jellyfish
  handle_exceptions true
  def controller; Api::Controller; end

  handle Api::Error do |error|
    render(error_custom(error))
  end
end
```

error.rb (Cont.)

```
class Api::MiddleError
  handle Jellyfish::NotFound do
    render(error_custom(
      Api::NotFound.new(
        "Path not found.")))
  end

  handle Exception do |error|
    render(error_500(error))
  end
```

Ruby Codes (Cont.)

- server.rb
- error.rb
- eagerload.rb
- newrelic.rb
- shell.rb
- waiter.rb

eagerload.rb

```
module Eagerload
  module_function
  def eagerload
    if ENV['RAILS_GROUPS'] == 'assets'
      $stderr.puts "Don't eagerload " \
          "for assets precompilation."
      return
    end
    # ...
```

eagerload.rb (Cont

```
def eagerload
  # ...
  require 'rest-more'
RC.eagerload
  require 'aws-sdk'
RC.eagerload(AWS::Core)
RC.eagerload(AWS::S3)
```

eagerload.rb (Cont

```
def eagerload
  # ...
  %w[app/models/**/*
     app/controllers/**/*
     app/mailers/**/*].
     each{ |pattern|
       require_all(Dir[pattern], 2) }
```

eagerload.rb (Cont

```
def eagerload
  # ...
  require 'api/requirements'
  require_all(Dir['lib/**/*'], 1)
  $stderr.puts "Eagerly loaded."
  true
end
```

Ruby Codes (Cont.)

- server.rb
- error.rb
- eagerload.rb
- newrelic.rb
- shell.rb
- waiter.rb

newrelic.rb

```
ActiveSupport.on_load(:active_record) do
  require 'newrelic_rpm'

NewRelic::Agent.after_fork(
  :force_reconnect => true) if
  defined?(Unicorn)
# ...
```

newrelic.rb (Cont.

```
require 'shell'
Shell.module_eval do
  include NewRelic::Agent::
                MethodTracer
  add_method_tracer :system_multi
end
```

newrelic.rb (Cont.

```
Cocaine::CommandLine.module_eval do
  include NewRelic::Agent::
                MethodTracer
  add_method_tracer :run
end
```

newrelic.rb (Cont.

```
Paperclip::Attachment.module_eval do
  include NewRelic::Agent::
                MethodTracer
  add_method_tracer :url
end
```

newrelic.rb (Cont.

```
require 'redis'
Redis::Client.module_eval do
  include NewRelic::Agent::
                MethodTracer
  add_method_tracer :process
end
```

newrelic.rb (Cont.

```
require 'rest-core/engine/future/future'
RC::Future.module_eval do
  include NewRelic::Agent::MethodTracer
  alias_method :yield_original, :yield
  def yield
    method = env[RC::REQUEST_METHOD]
    path   = env[RC::REQUEST_PATH]
    # ...
```

newrelic.rb (Cont.

```
def yield
  # ...
  metrics = ["External/#{path}/" \
      "#{self.class.name}/#{method}",
                 "External/#{path}/all",
                "External/all",
                "External/allWeb"]
  self.class.
    trace_execution_scoped metrics do
      yield_original
    end
end
```

Ruby Codes (Cont.)

- server.rb
- error.rb
- eagerload.rb
- newrelic.rb
- shell.rb
- waiter.rb

shell.rb

```ruby
require 'waiter'
require 'concurrency'
module Shell
  module_function
  # this is for cocaine
  def call command, env={}
    Shell.system(*command)
  end
  # e.g. Shell.system('identify', 'img')
  def system *args
    system_multi(args).first
  end
```

shell.rb (Cont.)

```
# e.g. Shell.system_multi(%w[identify
#        image.png], %w[identify me.png])
def system_multi *cmds
  if EM.reactor_running?
    # ...
  else
    cmds.map{ |cmd|
      system_blocking(cmd) }
  end
end
```

shell.rb (Cont.)

```
def system_multi *cmds
  # ...
  w = Waiter.new
  r = {}
  cmds.each.with_index do |cmd, idx|
    system_async(*cmd) do |output|
      r[idx] = output
      w.resume(r.sort.map(&:last)) if
        r.size == cmds.size
    end
  end
  w.yield
```

shell.rb (Cont.)

```
def system_async *args
  tuple = system_spawn(args)
  Thread.new do
    begin
      result = system_wait(*tuple)
    rescue Exception => e
      EM.instance_variable_get(
        :@error_handler).call(e)
    ensure
    # ...
ennnd
```

shell.rb (Cont.)

```
def system_async *args
  # ...
    if Concurrency::Model == Fiber
      EM.next_tick{ yield(result) }
    else
      yield(result)
    end
```

shell.rb (Cont.)

```
def system_spawn args
  err = if %w[test production].
                    include?(Rails.env)
          IO::NULL
        else
          Rails.logger.debug(
            "Shell: #{args.inspect}")
          $stderr
        end
# ...
```

shell.rb (Cont.)

```
def system_spawn args
  # ...
  rd, wr = IO.pipe
  pid = Process.spawn(*args,
    :out => wr, :err => err)
  [pid, rd, wr]
end
```

shell.rb (Cont.)

```
def system_wait pid, rd, wr
  wr.close
  result = rd.read
  Process.waitpid(pid)
  result
end
```

Ruby Codes (Cont.)

- server.rb
- error.rb
- eagerload.rb
- newrelic.rb
- shell.rb
- waiter.rb

waiter.rb

```ruby
require 'thread'
require 'fiber'

class Waiter
  def self.fiber_around?
    RC::RootFiber != Fiber.current &&
      Thread.main == Thread.current
  end
end
```

waiter.rb (Cont.)

```
class Waiter
  def initialize
    if Waiter.fiber_around?
      @fiber = Fiber.current
    else
      @condv = ConditionVariable.new
      @mutex = Mutex.new
    end
  end
end
```

waiter.rb (Cont.)

```
class Waiter
  def resume value
    if @fiber
      @fiber.resume(value)
    else
      @value = value
      @condv.broadcast
    end
  end
end
```

waiter.rb (Cont.)

```
class Waiter
  def yield
    if @fiber
      Fiber.yield
    else
      @mutex.synchronize{
        @condv.wait(@mutex) }
      @value
    end
  end
end
```

Ruby Codes (Cont.) (Cont.)

- image_util.rb
- run_later.rb
- /search
- /create_and_share

collage_web.rb

```
class CollageWeb < Collage
  extend ImageUtil

  def self.create_from urls_or_url,
                       fallback_urls=[]
    create(:image =>
      merge_images(
        urls2images(urls_or_url,
                    fallback_urls)))
  end
end
```

image_util.rb

```
require 'tempfile'
require 'shell'

module ImageUtil
  module_function
  def urls2files urls_or_url,
                 fallback_urls=[]
    urls2images(urls_or_url,
                fallback_urls).map(&:file)
  end
end
```

image_util.rb (Cont

```ruby
module ImageUtil
  module_function
  def urls2images urls_or_url,
                  fallback_urls=[]
    urls = [urls_or_url].flatten
    w = Waiter.new
    results = {}
    # ...
    w.yield.sort.map(&:last).compact
  end
end
```

image_util.rb (Cont

```
def urls2images ...
urls.zip([fallback_urls].flatten).
  each.with_index do |(url, fb), idx|
    ImageFile.from_url(url, fb) do |img|
      results[idx] = img
      w.resume(results) if
        results.size == urls.size
    end
  end
w.yield.sort.map(&:last).compact
```

image_util.rb(Cont

```ruby
module ImageUtil
  module_function
  def merge_images images
    # ...
  end

  ImageStruct =
    Struct.new(:width, :height, :file)
  class ImageFile < ImageStruct
    include ImageUtil
  # ...
end
```

image_util.rb (Cont

```
module ImageUtil
  module_function
  def wget?
    @wget ||= !`which wget`.strip.empty?
  end
end
```

image_util.rb(Cont

```
module ImageUtil
  module_function
  def wget_command path, url
    if wget?
      ['wget', '-O', path,
       '--no-check-certificate', url]
    else
      ['curl', '-o', path,
       '--insecure'                , url]
    end
  end
end
```

image_util.rb(Cont

```
def identify_command path
  ['identify', '-format', '%wx%h',
   path]
end
```

```
def identify_result output
  if m = output.match(/(\d+)x(\d+)/)
    [m[1], m[2]].map(&:to_i)
  else
    nil
  end
end
```

Ruby Codes (Cont.) (Cont.)

- image_util.rb
- run_later.rb
- /search
- /create_and_share

run_later.rb

```ruby
require 'concurrency'
module RunLater
  module_function
  def with
    if EM.reactor_running?
      EM.next_tick{
        Concurrency.wrap{ yield } }
    else
      yield
    end
  end
end
```

Ruby Codes (Cont.) (Cont.)

- image_util.rb
- run_later.rb
- /search
- /create_and_share

/search

```
get '/search' do
  params['q'].blank? &&
    missing_arguments('q') # this raises

  images, total = if limit == 0
    [[], 0]
  else
    # ...
  end
  # ...
end
```

/search (Cont.)

```
case params['engine']
when 'pinterest'; sch_pinterest
when 'pinterest_api'; sch_pinterest_api
when 'bing'   ; sch_bing   offset, limit
when 'azure'  ; sch_azure  offset, limit
when 'google'; sch_google offset, limit
when 'flickr'; sch_flickr offset, limit
else          ; sch_google offset, limit
                # default google

end
```

/search (Cont.)

```
render('search' =>
  with_paging('total'   => total,
              'limit'   => limit,
              'offset'  => offset,
              'q'       => params[:q],
              'engine'  => engine,
              'data'    => images))
```

/search (Cont.)

```
def search_from_google offset, limit
  google = RC::GoogleImage.new(
    referer: request.url,
    userip: request.ip)
  search_image_and_flatten(
    google, offset, limit) do |q, n, i|
      google.search_image(
        q, rsz: n, start: i)
  end
end
```

Ruby Codes (Cont.) (Cont.)

- image_util.rb
- run_later.rb
- /search
- /create_and_share

/create_and_share

```
post '/collages' do
  render_collage create_collage
end

post '/collages/create_and_share' do
  render_collage create_collage{ |c|
    transfer "/collages/#{c.id}/share" }
end
```

/create_and_share(Co

```
def transfer path
  status, headers, body =
    jellyfish.call(
      env.merge('PATH_INFO' => path))
  self.status  status
  self.headers headers
  self.body    body
end
```

/create_and_share(Co

```
def create_collage &upload_callback
  para  = params
  image = (para['image']||{})[:tempfile]
  plist = case para['plist']
when Hash; para['plist'][:tempfile]
else     ; para['plist']
end
# ...
```

/create_and_share(Co

```
if image
  # ...
else
  missing_arguments('image')
end
```

/create_and_share(C

```
c_para = params.merge('plist' => plist)
c_para.delete('image')
if rc_tumblr &&
    for_auths_and_clients.empty?
  # no user for tumblr
else # check user for any other cases
  c_para['user'] =
    current_user_with_check
end
```

/create_and_share(Co

```
c = Collage.create_with_params(c_para)
# make paperclip happy
image.extend(OriginalFilename)
image.original_filename =
  params['image'][:filename]
c.save_image(image, &upload_callback)
```

/create_and_share(Co

```
def save_image img, &block
  save_dimension img
  process_thumbnails img, &block
  self
end
```

/create_and_share(Co

```
def process_thumbnails img, &blk
  if blk
    RunLater.with do
      process_thumbnails_inplace img
      if blk.arity == 1; blk.call(self)
      else             ; blk.call; end
    end
  else
    process_thumbnails_inplace img
  end
end
```

Miscellaneous Extras

- crazy inspect

Miscellaneous Extras

- crazy inspect
- who cares backtrace for 404

Miscellaneous Extras

- crazy inspect
- who cares backtrace for 404
- assets pipeline and database connection

Miscellaneous Extras

- crazy inspect
- who cares backtrace for 404
- assets pipeline and
  database connection
- fiber stack overflow

Miscellaneous Extras

- crazy inspect
- who cares backtrace for 404
- assets pipeline and
  database connection
- fiber stack overflow
- fiber database connection pool

Miscellaneous Extras

- crazy inspect
- who cares backtrace for 404
- assets pipeline and
  database connection
- fiber stack overflow
- fiber database connection pool
- thread-safe kaminari patch

Q?