Exploring the Request/Response Life-Cycle through the Mongrel and Rails Source Code.

Who am I?

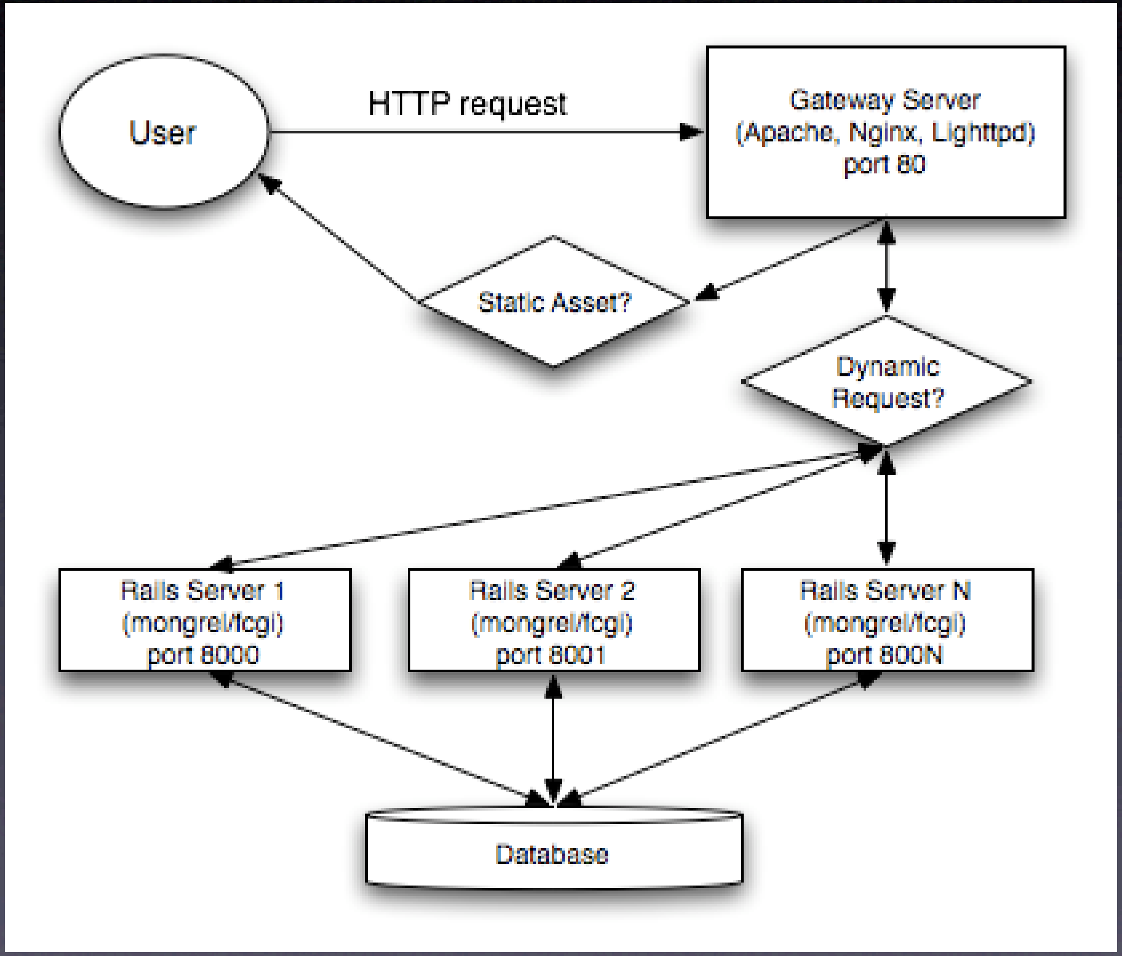
- Ezra Zygmuntowicz
- Rubyist for 4 years
- Engine Yard Founder and Architect
- Blog: http://brainspl.at

Keep your apps on track

- Understanding the high level request/response life-cycle can help you better understand your own applications
- Delving into the Rails source code to follow a request/response from start to finish will help you visualize things from your application's point of view

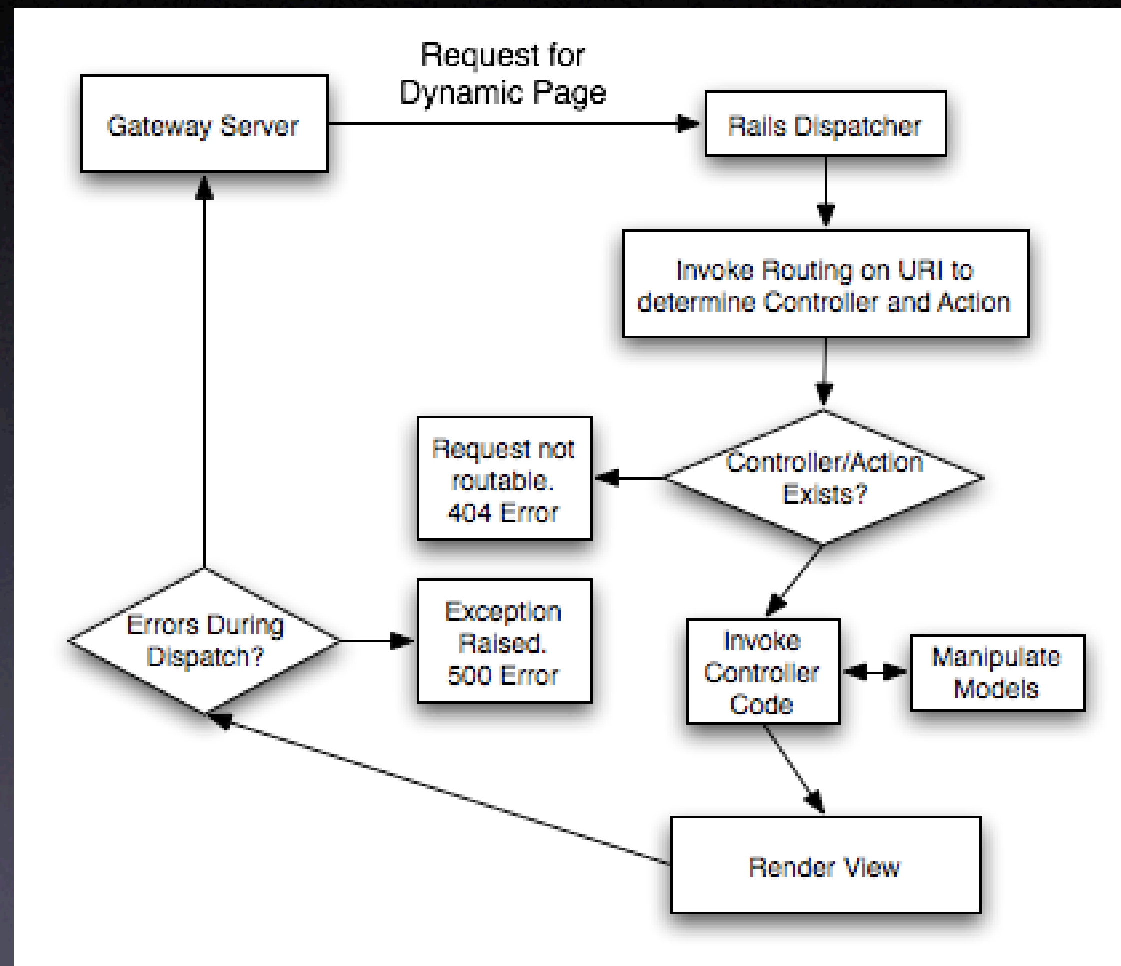
Full Stack Request/Response Life-Cycle

- Request comes into gateway server
- Rewrite rules are evaluated and request gets served directly if it's a static asset
- Dynamic requests are proxied to one Mongrel in the Mongrel Cluster
- Mongrel dispatches request through Rails and returns response to client



Rails Internal Request/Response Life-Cycle

- Rails Dispatcher is invoked with request/response objects
- Routing is invoked and returns the proper Controller object or 404 if no route found
- Filter chain is invoked
- Controller’s Action is called, manipulates Models
- View is rendered and any after filters are called
- Final response or error page returned to client



Mongrel Processes Request

- mongrel/lib/mongrel/rails.rb

```ruby
begin
  cgi = Mongrel::CGIWrapper.new(request, response)
  cgi.handler = self
  # We don't want the output to be really final until we're out of the lock
  cgi.default_really_final = false

  log_threads_waiting_for(@active_request_path || request.params["PATH_INFO"]) if $mongrel_debug_client

  @guard.synchronize {
    @active_request_path = request.params["PATH_INFO"]
    Dispatcher.dispatch(cgi, ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS, response.body)
    @active_request_path = nil
  }

  # This finalizes the output using the proper HttpResponse way
  cgi.out("text/html",true) {""}
rescue Errno::EPIPE
  response.socket.close
rescue Object => rails_error
  STDERR.puts "#{Time.now}: Error calling Dispatcher.dispatch #{rails_error.inspect}"
  STDERR.puts rails_error.backtrace.join("\n")
end
```

Rails Dispatcher.dispatch

- `railties/lib/dispatcher.rb`

```ruby
def dispatch(cgi = nil, session_options = ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS, output = $stdout)
  controller = nil
  if cgi ||= new_cgi(output)
    request, response = ActionController::CgiRequest.new(cgi, session_options), ActionController::CgiResponse.new(cgi)
    prepare_application
    controller = ActionController::Routing::Routes.recognize(request)
    controller.process(request, response).out(output)
  end
rescue Exception => exception  # errors from CGI dispatch
  failsafe_response(output, '500 Internal Server Error', exception) do
    controller ||= ApplicationController rescue LoadError nil
    controller ||= ActionController::Base
    controller.process_with_exception(request, response, exception).out(output)
  end
ensure
  # Do not give a failsafe response here.
  reset_after_dispatch
end
```

CgiRequest & CgiResponse

- actionpack/lib/action_controller/cgi_process.rb

```ruby
class CgiRequest < AbstractRequest #:nodoc:
  attr_accessor :cgi, :session_options
  DEFAULT_SESSION_OPTIONS = {
    :database_manager => CGI::Session::PStore,
    :prefix           => "ruby_sess.",
    :session_path     => "/"
  } unless const_defined?(:DEFAULT_SESSION_OPTIONS)
  def initialize(cgi, session_options = {})
    @cgi = cgi
    @session_options = session_options
    @env = @cgi.send(:env_table)
    super()
  end
  ...

class CgiResponse < AbstractResponse #:nodoc:
  def initialize(cgi)
    @cgi = cgi
    super()
  end
```

Rails Dispatcher.dispatch

- `railties/lib/dispatcher.rb`

```ruby
def dispatch(cgi = nil, session_options = ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS, output = $stdout)
  controller = nil
  if cgi ||= new_cgi(output)
    request, response = ActionController::CgiRequest.new(cgi, session_options), ActionController::CgiResponse.new(cgi)
    prepare_application
    controller = ActionController::Routing::Routes.recognize(request)
    controller.process(request, response).out(output)
  end
rescue Exception => exception  # errors from CGI dispatch
  failsafe_response(output, '500 Internal Server Error', exception) do
    controller ||= ApplicationController rescue LoadError nil
    controller ||= ActionController::Base
    controller.process_with_exception(request, response, exception).out(output)
  end
ensure
  # Do not give a failsafe response here.
  reset_after_dispatch
end
```

Dispatcher.prepare_application

- `railties/lib/dispatcher.rb`

```ruby
def prepare_application
  if Dependencies.load?
    ActionController::Routing::Routes.reload
    self.preparation_callbacks_run = false
  end

  prepare_breakpoint
  require_dependency 'application' unless Object.const_defined?(:ApplicationController)
  ActiveRecord::Base.verify_active_connections! if defined?(ActiveRecord)
  run_preparation_callbacks
end
```

Rails Dispatcher.dispatch

- `railties/lib/dispatcher.rb`

```
def dispatch(cgi = nil, session_options = ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS, output = $stdout)
  controller = nil
  if cgi ||= new_cgi(output)
    request, response = ActionController::CgiRequest.new(cgi, session_options), ActionController::CgiResponse.new(cgi)
    prepare_application
    controller = ActionController::Routing::Routes.recognize(request)
    controller.process(request, response).out(output)
  end
rescue Exception => exception  # errors from CGI dispatch
  failsafe_response(output, '500 Internal Server Error', exception) do
    controller ||= ApplicationController rescue LoadError nil
    controller ||= ActionController::Base
    controller.process_with_exception(request, response, exception).out(output)
  end
ensure
  # Do not give a failsafe response here.
  reset_after_dispatch
end
```

ActionController::Routing::Routes.recognize

- action_pack/lib/action_controller/routing.rb

```ruby
def recognize(request)
  params = recognize_path(request.path, extract_request_environment(request))
  request.path_parameters = params.with_indifferent_access
  "#{params[:controller].camelize}Controller".constantize
end

def recognize_path(path, environment={})
  path = CGI.unescape(path)
  routes.each do |route|
    result = route.recognize(path, environment) and return result
  end
  raise RoutingError, "no route found to match #{path.inspect} with #{environment.inspect}"
end
```

Rails Dispatcher.dispatch

- `railties/lib/dispatcher.rb`

```ruby
def dispatch(cgi = nil, session_options = ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS, output = $stdout)
  controller = nil
  if cgi ||= new_cgi(output)
    request, response = ActionController::CgiRequest.new(cgi, session_options), ActionController::CgiResponse.new(cgi)
    prepare_application
    controller = ActionController::Routing::Routes.recognize(request)
    controller.process(request, response).out(output)
  end
rescue Exception => exception  # errors from CGI dispatch
  failsafe_response(output, '500 Internal Server Error', exception) do
    controller ||= ApplicationController rescue LoadError nil
    controller ||= ActionController::Base
    controller.process_with_exception(request, response, exception).out(output)
  end
ensure
  # Do not give a failsafe response here.
  reset_after_dispatch
end
```

ActionController::Base#process

- actionpack/lib/action_controller/base.rb

```ruby
# Extracts the action_name from the request parameters and performs that action.
def process(request, response, method = :perform_action, *arguments) #:nodoc:
  initialize_template_class(response)
  assign_shortcuts(request, response)
  initialize_current_url
  assign_names
  forget_variables_added_to_assigns

  log_processing
  send(method, *arguments)

  assign_default_content_type_and_charset
  response
ensure
  process_cleanup
end
```

Filter processing added with alias_method_chain

- `actionpack/lib/action_controller/base.rb`

```ruby
module InstanceMethods # :nodoc:
  def self.included(base)
    base.class_eval do
      alias_method_chain :perform_action, :filters
      alias_method_chain :process, :filters
      alias_method_chain :process_cleanup, :filters
    end
  end

  def perform_action_with_filters
    #result = perform_filters do
    #  perform_action_without_filters unless performed?
    #end
    @before_filter_chain_aborted = (call_filter(self.class.filter_chain, 0) == false)
  end

  def process_with_filters(request, response, method = :perform_action, *arguments) #:n
    @before_filter_chain_aborted = false
    process_without_filters(request, response, method, *arguments)
  end
```

Module#alias_method_chain

active_support/lib/active_support/core_ext/module/aliasing.rb

```ruby
class Module
  # Encapsulates the common pattern of:
  #
  #   alias_method :foo_without_feature, :foo
  #   alias_method :foo, :foo_with_feature
  #
  # With this, you simply do:
  #
  #   alias_method_chain :foo, :feature
  #
  # And both aliases are set up for you.
  def alias_method_chain(target, feature)
    # Strip out punctuation on predicates or bang methods since
    # e.g. target?_without_feature is not a valid method name.
    aliased_target, punctuation = target.to_s.sub(/([?!=])$/, ''), $1
    yield(aliased_target, punctuation) if block_given?
    alias_method "#{aliased_target}_without_#{feature}#{punctuation}", target
    alias_method target, "#{aliased_target}_with_#{feature}#{punctuation}"
  end
```

Module#alias_method_chain

- Example

```ruby
class Foo
  def bar
    puts "Foo#bar called"
  end
end

class Baz < Foo
  def bar_with_puts
    puts "About to call bar_without_puts"
    bar_without_puts
    puts "After bar_without_puts"
  end
  alias_method_chain :bar, :puts
end

Foo.new.bar
Baz.new.bar
```

```
RubyMate r5712 running Ruby v1.8.5 (/usr/local/bin/ruby)
>>> untitled

Foo#bar called
About to call bar_without_puts
Foo#bar called
After bar_without_puts
```

ActionController::Base#process

- actionpack/lib/action_controller/base.rb

```ruby
# Extracts the action_name from the request parameters and performs that action.
def process(request, response, method = :perform_action, *arguments) #:nodoc:
  initialize_template_class(response)
  assign_shortcuts(request, response)
  initialize_current_url
  assign_names
  forget_variables_added_to_assigns

  log_processing
  send(method, *arguments)

  assign_default_content_type_and_charset
  response
ensure
  process_cleanup
end
```

ActionController::Base#initialize_template_class ActionController::Base#assign_shortcuts ActionController::Base#initialize_current_url

- `actionpack/lib/action_controller/base.rb`

```ruby
def initialize_template_class(response)
  response.template = self.class.view_class.new(self.class.view_root, {}, self)
  response.redirected_to = nil
  @performed_render = @performed_redirect = false
end

def assign_shortcuts(request, response)
  @_request, @_params, @_cookies = request, request.parameters, request.cookies
  @_response         = response
  @_response.session = request.session
  @_session = @_response.session
  @template = @_response.template
  @assigns  = @_response.template.assigns
  @_headers = @_response.headers
  assign_deprecated_shortcuts(request, response)
end

def initialize_current_url
  @url = UrlRewriter.new(request, params.clone)
end
```

ActionController::Base#process

- actionpack/lib/action_controller/base.rb

```
# Extracts the action_name from the request parameters and performs that action.
def process(request, response, method = :perform_action, *arguments) #:nodoc:
  initialize_template_class(response)
  assign_shortcuts(request, response)
  initialize_current_url
  assign_names
  forget_variables_added_to_assigns

  log_processing
  send(method, *arguments)

  assign_default_content_type_and_charset
  response
ensure
  process_cleanup
end
```

ActionController::Base#assign_names
ActionController::Base#forget_variables_added_to_assigns
ActionController::Base#log_processing

- actionpack/lib/action_controller/base.rb

```ruby
def assign_names
  @action_name = (params['action'] || 'index')
end

def forget_variables_added_to_assigns
  @variables_added = nil
end

def log_processing
  if logger
    logger.info "\n\nProcessing #{controller_class_name}\##{action_name}\
                (for #{request_origin}) [#{request.method.to_s.upcase}]"
    logger.info "  Session ID: #{@_session.session_id}" if \
                    @_session and @_session.respond_to?(:session_id)
    logger.info "  Parameters: #{respond_to?(:filter_parameters) ? \
                  filter_parameters(params).inspect : params.inspect}"
  end
end
```

ActionController::Base#process

- actionpack/lib/action_controller/base.rb

```
# Extracts the action_name from the request parameters and performs that action.
def process(request, response, method = :perform_action, *arguments) #:nodoc:
  initialize_template_class(response)
  assign_shortcuts(request, response)
  initialize_current_url
  assign_names
  forget_variables_added_to_assigns

  log_processing
  send(method, *arguments)

  assign_default_content_type_and_charset
  response
ensure
  process_cleanup
end
```

ActionController::Base#perform_action

- actionpack/lib/action_controller/base.rb

```
def perform_action
  if self.class.action_methods.include?(action_name)
    send(action_name)
    render unless performed?
  elsif respond_to? :method_missing
    send(:method_missing, action_name)
    render unless performed?
  elsif template_exists? && template_public?
    render
  else
    raise UnknownAction, "No action responded to #{action_name}", caller
  end
end

def assign_default_content_type_and_charset
  response.content_type ||= Mime::HTML
  response.charset      ||= self.class.default_charset
end
```

ActionController::Base#process

- actionpack/lib/action_controller/base.rb

```ruby
# Extracts the action_name from the request parameters and performs that action
def process(request, response, method = :perform_action, *arguments) #:nodoc:
  initialize_template_class(response)
  assign_shortcuts(request, response)
  initialize_current_url
  assign_names
  forget_variables_added_to_assigns

  log_processing
  send(method, *arguments)

  assign_default_content_type_and_charset
  response
ensure
  process_cleanup
end
```

Rails Dispatcher.dispatch

- `railties/lib/dispatcher.rb`

```
def dispatch(cgi = nil, session_options = ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS, output = $stdout)
  controller = nil
  if cgi ||= new_cgi(output)
    request, response = ActionController::CgiRequest.new(cgi, session_options), ActionController::CgiResponse.new(cgi)
    prepare_application
    controller = ActionController::Routing::Routes.recognize(request)
    controller.process(request, response).out(output)
  end
rescue Exception => exception  # errors from CGI dispatch
  failsafe_response(output, '500 Internal Server Error', exception) do
    controller ||= ApplicationController rescue LoadError nil
    controller ||= ActionController::Base
    controller.process_with_exception(request, response, exception).out(output)
  end
ensure
  # Do not give a failsafe response here.
  reset_after_dispatch
end
```

CgiResponse#out

- actionpack/lib/action_controller/cgi_process.rb

```ruby
def out(output = $stdout)
  convert_content_type!(@headers)
  output.binmode      if output.respond_to?(:binmode)
  output.sync = false if output.respond_to?(:sync=)

  begin
    output.write(@cgi.header(@headers))

    if @cgi.send(:env_table)['REQUEST_METHOD'] == 'HEAD'
      return
    elsif @body.respond_to?(:call)
      # Flush the output now in case the @body Proc uses
      # #syswrite.
      output.flush if output.respond_to?(:flush)
      @body.call(self, output)
    else
      output.write(@body)
    end

    output.flush if output.respond_to?(:flush)
  rescue Errno::EPIPE, Errno::ECONNRESET
    # lost connection to parent process, ignore output
  end
end
```

Mongrel Processes Request

- mongrel/lib/mongrel/rails.rb

```
begin
  cgi = Mongrel::CGIWrapper.new(request, response)
  cgi.handler = self
  # We don't want the output to be really final until we're out of the lock
  cgi.default_really_final = false

  log_threads_waiting_for(@active_request_path || request.params["PATH_INFO"]) if $mongrel_debug_client

  @guard.synchronize {
    @active_request_path = request.params["PATH_INFO"]
    Dispatcher.dispatch(cgi, ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS, response.body)
    @active_request_path = nil
  }

  # This finalizes the output using the proper HttpResponse way
  cgi.out("text/html",true) {""}
rescue Errno::EPIPE
  response.socket.close
rescue Object => rails_error
  STDERR.puts "#{Time.now}: Error calling Dispatcher.dispatch #{rails_error.inspect}"
  STDERR.puts rails_error.backtrace.join("\n")
end
```

Questions?