Rails on Rack

January 13, 2015

This guide covers Rails integration with Rack and interfacing with other Rack components.
After reading this guide, you will know:

- How to use Rack Middlewares in your Rails applications.
- Action Pack's internal Middleware stack.
- How to define a custom Middleware stack.

This guide assumes a working knowledge of Rack protocol and Rack concepts such as middlewares, url maps and `Rack::Builder`.

1 Introduction to Rack

Rack provides a minimal, modular and adaptable interface for developing web applications in Ruby. By wrapping HTTP requests and responses in the simplest way possible, it unifies and distills the API for web servers, web frameworks, and software in between (the so-called middleware) into a single method call.

- Rack API Documentation

Explaining Rack is not really in the scope of this guide. In case you are not familiar with Rack's basics, you should check out the Resources section below.

2 Rails on Rack

2.1 Rails Application's Rack Object

`Rails.application` is the primary Rack application object of a Rails application. Any Rack compliant web server should be using `Rails.application` object to serve a Rails application.

2.2 `rails server`

`rails server` does the basic job of creating a `Rack::Server` object and starting the webserver.
Here's how `rails server` creates an instance of `Rack::Server`

```
Rails::Server.new.tap do |server|
  require APP_PATH
  Dir.chdir(Rails.application.root)
  server.start
end
```

The `Rails::Server` inherits from `Rack::Server` and calls the `Rack::Server#start` method this way:

```
class Server < ::Rack::Server
  def start
    ...
    super
  end
end
```

Here's how it loads the middlewares:

```
def middleware
  middlewares = []
  middlewares << [Rails::Rack::Debugger] if options[:debugger]
  middlewares << [::Rack::ContentLength]
  Hash.new(middlewares)
end
```

`Rails::Rack::Debugger` is primarily useful only in the development environment. The following table explains the usage of the loaded middlewares:

Middleware	Purpose
`Rails::Rack::Debugger`	Starts Debugger
`Rack::ContentLength`	Counts the number of bytes in the response and set the HTTP Content-Length header

2.3 `rackup`

To use `rackup` instead of Rails' `rails server`, you can put the following inside `config.ru` of your Rails application's root directory:

```
# Rails.root/config.ru
require ::File.expand_path('../config/environment', __FILE__)

use Rails::Rack::Debugger
use Rack::ContentLength
run Rails.application
```

And start the server:

```
$ rackup config.ru
```

To find out more about different `rackup` options:

```
$ rackup --help
```

2.4 Development and auto-reloading

Middlewares are loaded once and are not monitored for changes. You will have to restart the server for changes to be reflected in the running application.

3 Action Dispatcher Middleware Stack

Many of Action Dispatcher's internal components are implemented as Rack middlewares. `Rails::Application` uses `ActionDispatch::MiddlewareStack` to combine various internal and external middlewares to form a complete Rails Rack application.

`ActionDispatch::MiddlewareStack` is Rails equivalent of `Rack::Builder`, but built for better flexibility and more features to meet Rails' requirements.

3.1 Inspecting Middleware Stack

Rails has a handy rake task for inspecting the middleware stack in use:

```
$ bin/rake middleware
```

For a freshly generated Rails application, this might produce something like:

```
use Rack::Sendfile
use ActionDispatch::Static
use Rack::Lock
use #<ActiveSupport::Cache::Strategy::LocalCache::Middleware:0x000000029a0838>
use Rack::Runtime
use Rack::MethodOverride
use ActionDispatch::RequestId
use Rails::Rack::Logger
use ActionDispatch::ShowExceptions
use ActionDispatch::DebugExceptions
use ActionDispatch::RemoteIp
use ActionDispatch::Reloader
use ActionDispatch::Callbacks
use ActiveRecord::Migration::CheckPending
use ActiveRecord::ConnectionAdapters::ConnectionManagement
use ActiveRecord::QueryCache
use ActionDispatch::Cookies
use ActionDispatch::Session::CookieStore
use ActionDispatch::Flash
```

```
use ActionDispatch::ParamsParser
use Rack::Head
use Rack::ConditionalGet
use Rack::ETag
run Rails.application.routes
```

The default middlewares shown here (and some others) are each summarized in the Internal Middlewares section, below.

3.2 Configuring Middleware Stack

Rails provides a simple configuration interface `config.middleware` for adding, removing and modifying the middlewares in the middleware stack via `application.rb` or the environment specific configuration file `environments/<environment>.rb`.

3.2.1 Adding a Middleware You can add a new middleware to the middleware stack using any of the following methods:

- `config.middleware.use(new_middleware, args)` - Adds the new middleware at the bottom of the middleware stack.
- `config.middleware.insert_before(existing_middleware, new_middleware, args)` - Adds the new middleware before the specified existing middleware in the middleware stack.
- `config.middleware.insert_after(existing_middleware, new_middleware, args)` - Adds the new middleware after the specified existing middleware in the middleware stack.

```
# config/application.rb

# Push Rack::BounceFavicon at the bottom
config.middleware.use Rack::BounceFavicon

# Add Lifo::Cache after ActiveRecord::QueryCache.
# Pass { page_cache: false } argument to Lifo::Cache.
config.middleware.insert_after ActiveRecord::QueryCache, Lifo::Cache, page_cache: false
```

3.2.2 Swapping a Middleware You can swap an existing middleware in the middleware stack using `config.middleware.swap`.

```
# config/application.rb

# Replace ActionDispatch::ShowExceptions with Lifo::ShowExceptions
config.middleware.swap ActionDispatch::ShowExceptions, Lifo::ShowExceptions
```

3.2.3 Deleting a Middleware Add the following lines to your application configuration:

```
# config/application.rb
config.middleware.delete "Rack::Lock"
```

And now if you inspect the middleware stack, you'll find that `Rack::Lock` is not a part of it.

```
$ bin/rake middleware
(in /Users/lifo/Rails/blog)
use ActionDispatch::Static
use #<ActiveSupport::Cache::Strategy::LocalCache::Middleware:0x00000001c304c8>
use Rack::Runtime
...
run Rails.application.routes
```

If you want to remove session related middleware, do the following:

```
# config/application.rb
config.middleware.delete "ActionDispatch::Cookies"
config.middleware.delete "ActionDispatch::Session::CookieStore"
config.middleware.delete "ActionDispatch::Flash"
```

And to remove browser related middleware,

```
# config/application.rb
config.middleware.delete "Rack::MethodOverride"
```

3.3 Internal Middleware Stack

Much of Action Controller's functionality is implemented as Middlewares. The following list explains the purpose of each of them:

`Rack::Sendfile`

- Sets server specific X-Sendfile header. Configure this via `config.action_dispatch.x_sendfile_header` option.

`ActionDispatch::Static`

- Used to serve static files. Disabled if `config.serve_static_files` is `false`.

`Rack::Lock`

- Sets `env["rack.multithread"]` flag to `false` and wraps the application within a Mutex.

`ActiveSupport::Cache::Strategy::LocalCache::Middleware`

- Used for memory caching. This cache is not thread safe.

`Rack::Runtime`

- Sets an X-Runtime header, containing the time (in seconds) taken to execute the request.

`Rack::MethodOverride`

- Allows the method to be overridden if `params[:_method]` is set. This is the middleware which supports the PUT and DELETE HTTP method types.

`ActionDispatch::RequestId`

- Makes a unique `X-Request-Id` header available to the response and enables the `ActionDispatch::Request#uuid` method.

`Rails::Rack::Logger`

- Notifies the logs that the request has began. After request is complete, flushes all the logs.

`ActionDispatch::ShowExceptions`

- Rescues any exception returned by the application and calls an exceptions app that will wrap it in a format for the end user.

`ActionDispatch::DebugExceptions`

- Responsible for logging exceptions and showing a debugging page in case the request is local.

`ActionDispatch::RemoteIp`

- Checks for IP spoofing attacks.

`ActionDispatch::Reloader`

- Provides prepare and cleanup callbacks, intended to assist with code reloading during development.

`ActionDispatch::Callbacks`

- Provides callbacks to be executed before and after dispatching the request.

`ActiveRecord::Migration::CheckPending`

- Checks pending migrations and raises `ActiveRecord::PendingMigrationError` if any migrations are pending.

`ActiveRecord::ConnectionAdapters::ConnectionManagement`

- Cleans active connections after each request, unless the `rack.test` key in the request environment is set to `true`.

`ActiveRecord::QueryCache`

- Enables the Active Record query cache.

`ActionDispatch::Cookies`

- Sets cookies for the request.

`ActionDispatch::Session::CookieStore`

- Responsible for storing the session in cookies.

`ActionDispatch::Flash`

- Sets up the flash keys. Only available if `config.action_controller.session_store` is set to a value.

`ActionDispatch::ParamsParser`

- Parses out parameters from the request into `params`.

`Rack::Head`

- Converts HEAD requests to `GET` requests and serves them as so.

`Rack::ConditionalGet`

- Adds support for "Conditional `GET`" so that server responds with nothing if page wasn't changed.

`Rack::ETag`

- Adds ETag header on all String bodies. ETags are used to validate cache.

It's possible to use any of the above middlewares in your custom Rack stack.

4 Resources

4.1 Learning Rack

- Official Rack Website
- Introducing Rack
- Ruby on Rack #1 - Hello Rack!
- Ruby on Rack #2 - The Builder

4.2 Understanding Middlewares

- Railscast on Rack Middlewares

5 Feedback

You're encouraged to help improve the quality of this guide.

Please contribute if you see any typos or factual errors. To get started, you can read our documentation contributions section.

You may also find incomplete content, or stuff that is not up to date. Please do add any missing documentation for master. Make sure to check Edge Guides first to verify if the issues are already fixed or not on the master branch. Check the Ruby on Rails Guides Guidelines for style and conventions.

If for whatever reason you spot something to fix but cannot patch it yourself, please open an issue.

And last but not least, any kind of discussion regarding Ruby on Rails documentation is very welcome in the rubyonrails-docs mailing list.
