Sinatra

Sinatra 1.4.0, 1.3.6, 1.2.9 released!

Posted by Konstantin Haase on **Friday, March 15, 2013** #

I've just released Sinatra 1.4.0, 1.3.6 and 1.2.9.

Find out what's new in 1.4 on my blog.

New feature release, Contrib and Recipes

Posted by Konstantin Haase on **Friday, September 30, 2011** #

We're proud to announce two new releases today: 1.3.0 and 1.2.7. We're also simultaneously releasing sinatra-contrib and would like to officially announce the recently launched Sinatra Recipes project. Read on for more goodness!

Sinatra 1.3.0

We have a new feature release!

From our perspective, one of the biggest additions is a streaming API. A simple version of it ships with Sinatra directly and is extended in sinatra-contrib. The vanilla version looks like this:

```
get '/' do
  stream do |out|
    out << "It's gonna be legen -\n"
    sleep 0.5
    out << " (wait for it) \n"
    sleep 1
    out << "- dary!\n"
  end
end
```

The cool thing about this is that it abstracts away the differences between all the different Rack servers, no matter if those are evented (like Thin, Rainbows! or Ebb) or sequential (like Unicorn, Passenger or Mongrel). Only WEBRick has issues with it at the moment (you'll get the response body all at once), but we are looking into this.

What is really interesting about this: If you run on an evented server, like Thin, you can keep the connection open and easily implement messaging services, Server-Sent Events and so on:

```
set :server, :thin
connections = []

get '/' do
  # keep stream open
  stream(:keep_open) { |out| connections << out }
end

post '/' do
  # write to all open streams
  connections.each { |out| out << params[:message] << "\n" }
  "message sent"
end
```

It is worth mentioning that all this works without fibers (which, amongst other things, would significantly limit the stack size).

We added support for the PATCH HTTP verb, which you might be familiar with from the new GitHub API:

```
patch '/' do
  # ... modify a resource ...
end
```

A `logger` helper for logging is available now, and if it writes to the logs depends on whether you enabled or disabled logging:

```
configure(:test) { disable :logging }

get '/' do
  logger.info "I just want to let you know: I'll take care of the request!"
  "Hello World!"
end
```

The `erubis` method has been deprecated. Sinatra will automatically use Erubis for rendering ERB templates if available.

There is more coming with this release, but the rest is mainly bug fixes and improving the behavior. For instance, Sinatra treats ETags and Last-Modified headers properly when they are used as optimistic locking for unsafe HTTP verbs. For more changes, see the 1.3.0 change log.

Special thanks for helping with the 1.3.0 release go to:

Gabriel Andretta, michelc, burningTyger, Tim Felgentreff, Vasily Polovnyov, nashby, kenichi nakamura, Gaku Ueda, Gabriel Horner, Sylvain Desvé, Jacob Burkhart & Josh Lane, aibo (irc), Selman ULUG, Aviv Ben-Yosef, Paolo “Nusco” Perrotta, Marcos Toledo, Matthew Schinckel, Tim Preston, Davide D’Agostino, Simone Carletti, Peter Suschlik, Postmodern, F. Zhang, Rémy Coutable, and David Waite

Sinatra 1.2.7

We will drop support for Rack prior to 1.3 and Ruby 1.8.6 with the Sinatra 1.3.0 release. However, we will continue to supply Sinatra 1.2 with bug fixes until the End-Of-Life of Rack 1.2.x and Sinatra 1.2 will always remain compatible with the latest released 1.8.6 patchlevel.

For the backported patches, see the 1.2.7 change log.

We'd like to thank everyone helping with the 1.2.7 release:

Emanuele Vicentini, Takanori Ishikawa, David Kellum, Gaku Ueda, Lee Reilly, Iain Barnett, pete, Luke Jahnke, John Wolfe, Andrew Armenia, Tim Felgentreff, and Alessandro Dal Grande

Sinatra-Contrib

There are a lot of Sinatra extensions out there, and some of those are used by a large number of apps, like sinatra-content-for or sinatra-reloader. The maintainers of these extensions have to keep up with new Sinatra releases to make sure everything works fine. This poses an issue from time to time, especially since some of these extensions were relying on Sinatra internals. To fix this, we created the sinatra-contrib project (source code at https://github.com/sinatra/sinatra-contrib). This project contains a number of extensions that are probably useful for most Sinatra applications. Some are old extensions that have largely been rewritten to ensure the code quality developers are used to from Sinatra itself and to no longer rely on internals, like sinatra-namespace, and some are new additions of general interest, for example sinatra-respond-with.

Here is the promise: For every Sinatra release, starting with 1.3.0, there will *always* be a fully compatible sinatra-contrib release. Documentation for all the extensions shipping with sinatra-contrib will be made available on the Sinatra website. I would not have managed to take care of this all by my self, so I'm really grateful that Gabriel Andretta jumped in to help with this project.

Documentation is available at sinatrarb.com/contrib.

Sinatra Recipes

Zachary Scott recently launched the Recipes project. It contains community contributed recipes and techniques for Sinatra.

Sinatra Loves Dancer

Posted by Konstantin Haase on **Thursday, July 21, 2011** #

"The only cool PR is provided by one's enemies. They toil incessantly and for free." - Marshall McLuhan

There are a lot of frameworks out there inspired by Sinatra. One of them is Dancer. Some unknown individual or group was posing as different members of the Sinatra core team on CPAN ratings, a website to comment on the quality of Perl libraries.

Those comments were extremely rude and do not represent the opinion of any of the Sinatra developers. Quite the opposite, we love how Sinatra inspired so many of the libraries and are even proud to see Dancer being adopted by Perl developers.

We do not know what the agenda of this one individual is, and honestly, we do not care. The harm done by this person to the Dancer project, the Sinatra project and the people he posed as is unacceptable. The incident has been reported to the CPAN ratings team and we hope that those comments will be removed in due time.

If you are a Perl developer, we suggest you check out Dancer.

What's New in Sinatra 1.2?

Posted by Konstantin Haase on **Thursday, March 3, 2011** #

As announced on the mailing list, we have just released Sinatra 1.2.0. Let's have a closer look at the new features.

Slim support

Sinatra now supports the Slim Template Engine:

```
require 'sinatra'
require 'slim'

get('/') { slim :index }

__END__

@@ index
! doctype html
html
  head
    title Sinatra With Slim
  body
    h1 Slim Is Fun!
    a href="http://haml-lang.com/" A bit like Haml, don't you think?
```

Inline Markaby

Like Builder and Nokogiri templates, Markaby can now be used directly inline:

```
require 'sinatra'
require 'markaby'

get '/' do
  markaby do
    html do
      head { title "Sinatra With Markaby" }
      body { h1 "Markaby Is Fun!" }
    end
  end
end
```

Layout Engines

Whenever you render a template, like `some_page.haml`, Sinatra will use a corresponding layout, like `layout.haml` for you. With the `:layout` option it has always been possible to use a different layout, but it still had to be written in the same template language, Haml in our example. In 1.1 we introduced `markdown`, `textile` and `rdoc` templates. It is not possible to use those for layouts. We therefore added the `:layout_engine` option, which easily allows you to combine one two different template engines:

```
require 'sinatra'
require 'rdiscount'

# for all markdown files, use post.haml as layout
set :markdown, :layout_engine => :haml, :layout => :post

get '/' do
  # use index.haml for readme
  markdown :README, :layout => :index
end

get '/:post' do
  markdown params[:post].to_sym
end
```

This feature should also be handy when migrating from one template language to another, as it allows you to combine Erb with Haml, for instance.

Conditional Filters

We introduced pattern matching filters in 1.1. Now they also support conditions:

```
before :agent => /Song Bird/ do
  # ...
end
```

Those can also be combined with patterns, of course:

```
after '/api/*', :provides => :json do
  # ...
end
```

URL helper

Usually Sinatra does not provide any view helper methods. Those are provided by extensions and would not suit Sinatra's approach of a small but robust core. However, constructing URLs is a use case most people run into sooner or later. It is a bit complicated to construct URLs right. Consider this example:

```
get('/foo') { "<a href='/bar'>Will you make it?</a>" }
get('/bar') { "You made it!" }
```

Feel free to run it. Works, doesn't it? So, what is wrong with it?

Imagine your app is "mounted" by another Rack application, for instance in a `config.ru` like this:

```
map('/there') { run Sinatra::Application }
map('/') { run MyRailsApp::Application }
```

Now the link to `/bar` would end up in a request send to `MyRailsApp` rather than to Sinatra. Injecting `request.script_name` would fix this, but be honest, how often do you do that?

Now, imagine these links are presented out of context, in an RSS feed or embedded on another host. In that case you might want to construct absolute URLs. This is even more cumbersome, as you most certainly either forget to handle reverse proxies, alternative ports/protocols or you end up with lots of URL related code all over the place, while what you should do is use the `url` helper:

```
get('/foo') { "<a href='#{url '/bar'}'>You will make it!</a>" }
get('/bar') { "You made it!" }
```

Since you are likely going to use this with redirects, we also aliased the helper to `to`:

```
get('/foo') { redirect to('/bar') }
get('/bar') { "You made it!" }
```

Named Captures on 1.9

Ruby 1.9 introduced named captures for regular expressions. Sinatra accepts regular expressions for matching paths. Now named captures will automatically end up populating `params`:

```
get %r{/(?<year>\d{4})/(?<month>\d{2})/(?<day>\d{2})/?} do
  date = Date.new params[:year].to_i, params[:month].to_i, params[:day].to_i
  @posts = Post.pubished_on date
  erb :posts
end
```

Templates with different scopes

All rendering methods now accept a `:scope` options:

```
get '/:id' do |id|
  @post = Post.find id

  # without scope
  erb "<%= @post.name %>"

  # with scope
  erb "<%= name %>", :scope => @post
end
```

Note that all Sinatra helper methods and instance variables will *not* be available.

Configurable redirects

In 1.1 we made sure all redirects were absolute URIs, to conform with RFC 2616 (HTTP 1.1). This will result in issues for you if you have a broken Reverse Proxy configuration. If so, you should really fix your configuration. If you are unable to do so, a simple `disable :absolute_redirects` will now give you back the 1.0 behavior. As shown above, you can now use the `to` helper with redirect. If all your redirects are application local, you can now `enable :prefixed_redirects` and skip the `to` altogether:

```
enable :prefixed_redirects
get('/foo') { redirect '/bar' }
get('/bar') { "You made it!" }
```

We did not enable this per default to not break compatibility and to allow you redirects to other Rack endpoints.

Overriding template lookup

One popular feature request is supporting multiple view folders. But everyone wants different semantics. So, instead of choosing one way to go, we gave you means to implement your own lookup logic:

```
helpers do
  def find_template(*)
    puts "looking for index.txt"
    yield "views/index.txt"
    puts "apparently, index.txt doesn't exist, let's try index.html"
    yield "views/index.html"
  end
end

get "/" do
  haml :foo
end
```

Sinatra will call `find_template` to discover the template file. In the above example, we don't care about what template engine to use or what name the template has. It will use `views/index.txt` or `views/index.html` for every template. Let's have a look at the standard implementation:

```
def find_template(views, name, engine)
  Tilt.mappings.each do |ext, klass|
    next unless klass == engine
    yield ::File.join(views, "#{name}.#{ext}")
  end
end
```

As you can see, it will look in the views folder for a file named like the template with any of the file extensions registered for the template engine.

If all you want to change is the folder, you probably should just call `super`:

```
def find_template(views, *a, &b)
  super("#{views}/a", *a, &b)
  super("#{views}/b", *a, &b)
end
```

More examples can be found in the readme.

Other changes

- `send_file` now takes a `:last_modified` option
- improved error handling

Blog revived

Posted by Konstantin Haase on **Thursday, March 3, 2011** #

It has been quiet on this blog recently. Announcements, like the 1.0 and 1.1 releases, have only been made on the mailing list. Plans are to change this and accompany mailing list announcements with in-depth articles about new feature releases.

We Have a FAQ Now

Posted by Ryan Tomayko on **Thursday, January 29, 2009** #

Just getting started with Sinatra and wondering how to do something that should be obvious? Check out The FAQ. We culled a few weeks worth of IRC logs and mailing list threads to compile the initial set of entries and we'll be adding to the list considerably over the coming weeks.

Have an idea for a FAQ entry? Fork the sinatra.github.com repository, add the entry to `faq.markdown`, and push. We'll take care of the rest.

What's New in Sinatra 0.9.0

Posted by Ryan Tomayko on **Sunday, January 18, 2009** #

This is the first in a series of 0.9.x releases designed to move Sinatra toward a rock solid 1.0. While we were able to add a touch of new hotness in this release, the major focus has been on getting the codebase shaped up for future development. Many longstanding bugs and minor annoyances were corrected along the way.

Sinatra's internal classes and methods have changed significantly in this release. Most apps written for 0.3.x will work just fine under the new codebase but we've begun adding deprecation warnings for things slated to be ripped out completely in 1.0. **Please test your apps before upgrading production environments to the 0.9.0 gem.** We're committed to keeping existing apps running through 0.9.x so report compatibility issues through the normal channels.

With that out of the way, here are some of the new features to look out for in 0.9.0.

Sinatra::Base and Proper Rack Citizenship

Sinatra can now be used to build modular / reusable Rack applications and middleware components. This means that multiple Sinatra applications can now be run in isolation and co-exist peacefully with other Rack based frameworks.

Requiring `'sinatra/base'` instead of `'sinatra'` causes a subset of Sinatra's features to be loaded. No methods are added to the top-level and the command-line / auto-running features are disabled. Subclassing `Sinatra::Base` creates a Rack component with the familiar Sinatra DSL methods available in class scope. These classes can then be run as Rack applications or used as middleware components.

Proper documentation on this feature is in the works but here's a quick example for illustration:

```
require 'sinatra/base'

class Foo < Sinatra::Base
  # all options are available for the setting:
  enable :static, :session
  set :root, File.dirname(__FILE__)

  # each subclass has its own private middleware stack:
  use Rack::Deflator

  # instance methods are helper methods and are available from
  # within filters, routes, and views:
  def em(text)
    "<em>#{text}</em>"
  end

  # routes are defined as usual:
  get '/hello/:person' do
    "Hello " + em(params[:person])
  end
end
```

That thing can be plugged in anywhere along a Rack pipeline. For instance, once Rails 2.3 ships, you'll be able to use Sinatra apps to build Rails Metal.

Jesse Newland and Jon Crosby are already experimenting. Very hot.

Nested Params

Form parameters with subscripts are now parsed into a nested/recursive Hash structure. Here's a form:

```
<form method='POST' action='/guestbook/'>
  <input type='text' name='person[name]'>
  <input type='text' name='person[email]'>
  <textarea name='note'></textarea>
</form>
```

It looks like this on the wire:

```
person[name]=Frank&person[email]=frank@theritz.com&message=Stay%20cool
```

Sinatra turns it into a nested Hash structure when accessed through the `params` method:

```
post '/guestbook/' do
  params[:person]    # => { :name => 'Frank', :email => 'frank@theritz.com' }
  "Hi #{person[:name]}! Thanks for signing my guestbook."
end
```

This was a massively popular feature requests. Thanks to Nicolás Sanguinetti for the patch and Michael Fellinger for the original implementation.

Routing with Regular Expressions

The route declaration methods (`get`, `put`, `post`, `put`, `delete`) now take a Regexp as a pattern. Captures are made available to the route block at `params[:captures]`:

```
get %r{/foo/(bar|baz)/(\d+)} do
  # assuming: GET /foo/bar/42
  params[:captures]  # => ['bar', 42]
end
```

Passing on a Route

We added a new request-level `pass` method that immediately exits the current block and passes control to the next matching route. For example:

```
get '/shoot/:person' do
  pass unless %w[Kenny Sherrif].include?(params[:person])
  "You shot #{params[:person]}."
end

get '/shoot/*' do
  "Missed!"
end
```

If no matching route is found after a `pass`, a `NotFound` exception is raised and the application 404s as if no route had matched in the first place.

Refined Test Framework

Sinatra's testing support no longer depends on `Test::Unit` (or any specific test framework for that matter). Requiring `'sinatra/test'` brings in the `Sinatra::Test` module and the `Sinatra::TestHarness` class, which can be used as necessary to simulate requests and make assertions about responses.

You can also require `sinatra/test/unit`, `sinatra/test/spec`, `sinatra/test/rspec`, or `sinatra/test/bacon` to setup a framework-specific testing environment. See the section on "Testing"

in the README for examples.

More

See the CHANGES file for a comprehensive list of enhancements, bug fixes, and deprecations.