LEARNING sinatra

#sinatra

Table of Contents

About

Chapter 1: Getting started with sinatra

Remarks

Sinatra is a simple Ruby DSL for quickly creating web applications.

It has built in features for routing, using templates, serving static files, helpers, error handling and many other topics.

http://www.sinatrarb.com/intro.html

Versions

Version	Release Date
2.0.0beta2	2016-08-22
2.0.0beta1	2016-08-22
1.4.7	2016-01-24
1.4.0	2013-03-15
1.3.0	2011-10-01
1.2.0	2011-03-02
1.1.0	2010-10-23
1.0.0	2010-03-23
0.9.0	2009-01-18
0.3.0	2008-08-31
0.2.0	2008-04-11
0.1.0	2007-10-04

Examples

Installation

You can install Sinatra as a global gem:

```
gem install sinatra
```

or add it to a project's Gemfile

```
# in Gemfile:
gem 'sinatra'
```

and run `bundle install`.

Your first Sinatra app

```
# app.rb
require 'sinatra'

get '/' do
    'Hello, Universe!'
end
```

Install Sinatra:

```
gem install sinatra
```

Run the app:

```
ruby app.rb
```

That's it! Access your app at http://localhost:4567

Read Getting started with sinatra online: https://riptutorial.com/sinatra/topic/4533/getting-started-with-sinatra

Chapter 2: Routing

Examples

What is routing

In Sinatra, routing is how your app responds to requests, by the path of the request (e.g. `/welcome`) and by the `HTTP` verb used (e.g. `GET` or `POST`). The way a request is written is as follows:

```
<http-verb> <path> do
    <code block to execute when this route is requested>
end
```

Here is an example that responds to `GET` requests to the path `/hello` by returning a page that says "Hi, whats up":

```
get "/hello" do
    return "Hi, whats up"
end
```

Sinatra only responds to routes that you define. **If you do not define a route, Sinatra returns a `404 Page Not Found` error page.**

Sinatra responds to routes in the order they are defined. If you have several routes that can match a given request (see "Regexp based path matching"), the first route that fits the request is returned.

NOTE: Sinatra treats routes with and without trailing forward-slash (`/`) as 2 different and distinct routes. That is, `get '/hello'` and `get '/hello/'` by default match different blocks of code. If you want to ignore the trailing forward-slash and treat both routes as the same, you can add `?` after the forward-slash to make it optional, like so: `get '/hello/?'`. This uses Sinatra's ability to use regular expressions for route matching (more on this below).

Regexp based path matching

When matching the path of a route, you can do it explicitly, matching only one path, like so:

```
get "/hello" do
    return "Hello!"
end
```

You can also use a regular expression to match complex routes. Any route which matches the regular expression will run that code block. If multiple routes can potentially match the request, the first-matched route is executed.

Here's a typical example of a route that matches paths that include `/user/` followed by one or more digits (presumably, user IDs) i.e. `GET /user/1`:

```
get /\/user\/\d+/ do
  "Hello, user!"
end
```

The example above matches `/user/1`, but will also match `/delete/user/1` and `/user/1/delete/now`, since our regular expression is not very restrictive and allows for a partial match against any part of the path.

We can be more explicit with the regexp and tell it to match the route exactly, using `\A` and `\z` directives to anchor the match to the beginning and the end of the path:

```
get /\A\/user\/\d+\z/ do
  "Hello, user!"
end
```

This route will not match `/delete/user/1` or `/user/1/delete/now` because of match anchoring.

Ignoring Trailing `/`

Our example route above will also **not** match `/user/1/` (with trailing forward-slash). If you want to ignore trailing slash at the end of the route, adjust the regexp to make the slash optional (*note the `\/?` at the end*):

```
get /\A\/user\/\d+\/?\z/ do
  "Hello, user! You may have navigated to /user/<ID> or /user/<ID>/ to get here."
end
```

Capturing Route Matches

So far, we've matched against regexp routes, but what if we want to use the matched values in our code block? Following up on our example, how do we know what the user's ID is when the route is executed?

We can **capture** the desired part of the path and use Sinatra's `param[:captures]` variable to work with the data inside the route:

```
get /\A\/user\/(\d+)\/?\z/ do
  "Hello, user! Your ID is #{params['captures'].first}!"
end
```

Available Sinatra Routing verbs

There are a number of available routing verbs in Sinatra, they correspond directly to http verbs

```
get '/' do
  .. get some data, a view, json, etc ..
end

post '/' do
  .. create a resource ..
end
```

```
put '/' do
  .. replace a resource ..
end

patch '/' do
  .. change a resource ..
end

delete '/' do
  .. delete something ..
end

options '/' do
  .. appease something ..
end

link '/' do
  .. affiliate something ..
end

unlink '/' do
  .. separate something ..
end
```

Sinatra Route Parameters

Of course you can pass data to Sinatra routes, to accept data in your routes you can add route paremeters. You can then access a params hash:

```
get '/hello/:name' do
  # matches "GET /hello/foo" and "GET /hello/bar"
  # params['name'] is 'foo' or 'bar'
  "Hello #{params['name']}!"
end
```

You can also assign parameters directly to variables like we usually do in Ruby hashes:

```
get '/hello/:name' do |n|
  # matches "GET /hello/foo" and "GET /hello/bar"
  # params['name'] is 'foo' or 'bar'
  # n stores params['name']
  "Hello #{n}!"
end
```

You can also add wildcard parameters without any specific names by using asteriks. They can then be accessed by using params['splat']:

```
get '/say/*/to/*' do
  # matches /say/hello/to/world
  params['splat'] # => ["hello", "world"]
end
```

Read Routing online: https://riptutorial.com/sinatra/topic/4692/routing

Credits

S. No	Chapters	Contributors
1	Getting started with sinatra	Arman H, Community, mhutter, tpei
2	Routing	aisflat439, Arman H, thesecretmaster, tpei