Sinatra

Configuring Settings

Sinatra includes a number of built-in settings that control whether certain features are enabled. Settings are application-level variables that are modified using one of the `set`, `enable`, or `disable` methods and are available within the request context via the `settings` object. Applications are free to set custom settings as well as the default, built-in settings provided by the framework.

Using `set`, `enable`, and `disable`

In its simplest form, the `set` method takes a setting name and value and creates an attribute on the application. Settings can be accessed within requests via the `settings` object:

```
set :foo, 'bar'

get '/foo' do
  "foo is set to " + settings.foo
end
```

Deferring evaluation

When the setting value is a `Proc`, evaluation is performed when the setting is read so that other settings may be used to calculate the value:

```
set :foo, 'bar'
set :baz, Proc.new { "Hello " + foo }

get '/baz' do
  "baz is set to " + settings.baz
end
```

The `/baz` response should come as “baz is set to Hello bar” unless the `foo` setting is modified.

Configuring multiple settings

Multiple settings can be set by passing a Hash to `set`. The previous example could be rewritten with:

```
set :foo => 'bar', :baz => Proc.new { "Hello " + foo }
```

Setting multiple boolean settings with `enable` and `disable`

The `enable` and `disable` methods are sugar for setting a list of settings to `true` or `false`, respectively. The following two code examples are equivalent:

```
enable  :sessions, :logging
disable :dump_errors, :some_custom_option
```

Using `set`:

```
set :sessions, true
set :logging, true
set :dump_errors, false
set :some_custom_option, false
```

Built-in Settings

`:environment` - configuration/deployment environment

A symbol specifying the deployment environment; typically set to one of `:development`, `:test`, or `:production`. The `:environment` defaults to the value of the `RACK_ENV` environment variable (`ENV['RACK_ENV']`), or `:development` when no `RACK_ENV` environment variable is set.

The environment can be set explicitly:

```
set :environment, :production
```

`:sessions` - enable/disable cookie based sessions

Support for encrypted, cookie-based sessions are included with Sinatra but are disabled by default. Enable them with:

```
set :sessions, true
```

Sessions are implemented by inserting the `Rack::Session::Cookie` component into the application's middleware pipeline.

`:logging` - log requests to STDERR

Writes a single line to `STDERR` in Apache common log format when enabled. This setting is enabled by default in classic style apps and disabled by default in `Sinatra::Base` subclasses.

Internally, the `Rack::CommonLogger` component is used to generate log messages.

`:method_override` - enable/disable the POST `_method` hack

Boolean specifying whether the HTTP POST `_method` parameter hack should be enabled. When `true`, the actual HTTP request method is overridden by the value of the `_method` parameter included in the POST body. The `_method` hack is used to make POST requests look like other request methods (e.g., `PUT`, `DELETE`) and is typically only needed in shitty environments – like HTML form submission – that do not support the full range of HTTP methods.

The POST `_method` hack is implemented by inserting the `Rack::MethodOverride` component into the middleware pipeline.

`:root` – The application's root directory

The directory used as a base for the application. By default, this is assumed to be the directory containing the main application file (`:app_file` setting). The root directory is used to construct the default `:public_folder` and `:views` settings. A common idiom is to set the `:root` setting explicitly in the main application file as follows:

```
set :root, File.dirname(__FILE__)
```

`:static` – enable/disable static file routes

Boolean that determines whether static files should be served from the application's public directory (see the `:public_folder` setting). When `:static` is truthy, Sinatra will check if a static file exists and serve it before checking for a matching route.

The `:static` setting is enabled by default when the `public` directory exists.

`:public_folder` – static files directory

A string specifying the directory where static files should be served from. By default, this is assumed to be a directory named "public" within the root directory (see the `:root` setting). You can set the public directory explicitly with:

```
set :public_folder, '/var/www'
```

The best way to specify an alternative directory name within the root of the application is to use a deferred value that references the `:root` setting:

```
set :public_folder, Proc.new { File.join(root, "static") }
```

`:views` – view template directory

A string specifying the directory where view templates are located. By default, this is assumed to be a directory named "views" within the application's root directory (see the `:root` setting). The best way to specify an alternative directory name within the root of the application is to use a deferred value that references the `:root` setting:

```
set :views, Proc.new { File.join(root, "templates") }
```

:run – enable/disable the built-in web server

Boolean specifying whether the built-in web server is started after the app is fully loaded. By default, this setting is enabled only when the `:app_file` matches `$0`. i.e., when running a Sinatra app file directly with `ruby myapp.rb`. To disable the built-in web server:

```
set :run, false
```

:server – handler used for built-in web server

String or Array of Rack server handler names. When the `:run` setting is enabled, Sinatra will run through the list and start a server with the first available handler. The `:server` setting is set as follows by default:

```
set :server, %w[thin mongrel webrick]
```

:bind – server hostname or IP address

String specifying the hostname or IP address of the interface to listen on when the `:run` setting is enabled. The default value in the development environment is `'localhost'` which means the server is only available from the local machine. In other environments the default is `'0.0.0.0'`, which causes the server to listen on all available interfaces.

To listen on all interfaces in the development environment (for example if you want to test from other computers in your local network) use:

```
set :bind, '0.0.0.0'
```

This can also be set from the command line with the `-o` option. If you set the bind option in your application it will override anything set on the command line.

:port – server port

The port that should be used when starting the built-in web server when the `:run` setting is enabled. The default port is `4567`. To set the port explicitly:

```
set :port, 9494
```

:app_file – main application file

The `:app_file` setting is used to calculate the default `:root`, `:public_folder`, and `:views` setting values. A common idiom is to override the default detection heuristic by setting the `:app_file` explicitly from within the main application file:

```
set :app_file, __FILE__
```

It's also used to detect whether Sinatra should boot a web server when using classic-style applications.

:dump_errors - log exception backtraces to STDERR

Boolean specifying whether backtraces are written to STDERR when an exception is raised from a route or filter. This setting is enabled by default in classic style apps. Disable with:

```
set :dump_errors, false
```

:raise_errors - allow exceptions to propagate outside of the app

Boolean specifying whether exceptions raised from routes and filters should escape the application. When disabled, exceptions are rescued and mapped to error handlers which typically set a 5xx status code and render a custom error page. Enabling the :raise_errors setting causes exceptions to be raised outside of the application where it may be handled by the server handler or Rack middleware, such as Rack::ShowExceptions or Rack::MailExceptions.

:lock - ensure single request concurrency with a mutex lock

Sinatra can be used in threaded environments where more than a single request is processed at a time. However, not all applications and libraries are thread-safe and may cause intermittent errors or general weirdness. Enabling the :lock setting causes all requests to synchronize on a mutex lock, ensuring that only a single request is processed at a time.

The :lock setting is disabled by default.

:show_exceptions - enable classy error pages

Enable error pages that show backtrace and environment information when an unhandled exception occurs. Enabled in development environments by default.