
Spring



Spring is a Rails application preloader. It speeds up development by keeping your application running in the background, so you don't need to boot it every time you run a test, rake task or migration.

Features

- Totally automatic; no need to explicitly start and stop the background process
- Reloads your application code on each run
- Restarts your application when configs / initializers / gem dependencies are changed

Compatibility

- Ruby versions: MRI 2.7, MRI 3.0, MRI 3.1, MRI 3.2
- Rails versions: 6.0, 6.1, 7.0
- Bundler v2.1+

Spring makes extensive use of `Process.fork`, so won't be able to provide a speed up on platforms which don't support forking (Windows, JRuby).

Walkthrough

Setup

Add Spring to your Gemfile:

```
1 gem "spring", group: :development
```

(Note: using `gem "spring", git: "..."` won't work and is not a supported way of using Spring.)

It's recommended to 'springify' the executables in your `bin/` directory:

```
1 $ bundle install
2 $ bundle exec spring binstub --all
```

This generates a `bin/spring` executable, and inserts a small snippet of code into relevant existing executables. The snippet looks like this:

```
1 begin
2   load File.expand_path('../spring', __FILE__)
3 rescue LoadError => e
4   raise unless e.message.include?('spring')
5 end
```

On platforms where Spring is installed and supported, this snippet hooks Spring into the execution of commands. In other cases, the snippet will just be silently ignored, and the lines after it will be executed as normal.

If you don't want to prefix every command you type with `bin/`, you can use `direnv` to automatically add `./bin` to your `PATH` when you `cd` into your application. Simply create an `.envrc` file with the command `PATH_add bin` in your Rails directory.

Enable reloading

Spring reloads application code, and therefore needs the application to have reloading enabled.

Ensure that `config.enable_reloading` is `true` in the environments that Spring manages. That setting is typically configured in `config/environments/*.rb`. In particular, make sure it is `true` for the `test` environment.

Note: in versions of Rails before 7.1, the setting is called `cache_classes`, and it needs to be `false` for Spring to work.

Usage

For this walkthrough I've generated a new Rails application, and run `rails generate scaffold post name:string`.

Let's run a test:

```
1 $ time bin/rake test test/controllers/posts_controller_test.rb
2 Running via Spring preloader in process 2734
3 Run options:
4
5 # Running tests:
6
7 .....
8
9 Finished tests in 0.127245s, 55.0121 tests/s, 78.5887 assertions/s.
10
11 7 tests, 10 assertions, 0 failures, 0 errors, 0 skips
12
13 real    0m2.165s
```

```
14 user    0m0.281s
15 sys     0m0.066s
```

That wasn't particularly fast because it was the first run, so Spring had to boot the application. It's now running:

```
1 $ bin/spring status
2 Spring is running:
3
4 26150 spring server | spring-demo-app | started 3 secs ago
5 26155 spring app    | spring-demo-app | started 3 secs ago | test mode
```

The next run is faster:

```
1 $ time bin/rake test test/controllers/posts_controller_test.rb
2 Running via Spring preloader in process 8352
3 Run options:
4
5 # Running tests:
6
7 .....
8
9 Finished tests in 0.176896s, 39.5714 tests/s, 56.5305 assertions/s.
10
11 7 tests, 10 assertions, 0 failures, 0 errors, 0 skips
12
13 real    0m0.610s
14 user    0m0.276s
15 sys     0m0.059s
```

If we edit any of the application files, or test files, the changes will be picked up on the next run without the background process having to restart. This works in exactly the same way as the code reloading which allows you to refresh your browser and instantly see changes during development.

But if we edit any of the files which were used to start the application (configs, initializers, your gemfile), the application needs to be fully restarted. This happens automatically.

Let's "edit" `config/application.rb`:

```
1 $ touch config/application.rb
2 $ bin/spring status
3 Spring is running:
4
5 26150 spring server | spring-demo-app | started 36 secs ago
6 26556 spring app    | spring-demo-app | started 1 sec ago | test mode
```

The application detected that `config/application.rb` changed and automatically restarted itself.

If we run a command that uses a different environment, then that environment gets booted up:

```
1 $ bin/rake routes
2 Running via Spring preloader in process 2363
3   posts GET    /posts(&:format)      posts#index
4       POST    /posts(&:format)      posts#create
5   new_post GET    /posts/new(&:format)  posts#new
6   edit_post GET    /posts/:id/edit(&:format) posts#edit
7       post GET    /posts/:id(&:format)  posts#show
8       PUT     /posts/:id(&:format)  posts#update
9       DELETE  /posts/:id(&:format)  posts#destroy
10
11 $ bin/spring status
12 Spring is running:
13
14 26150 spring server | spring-demo-app | started 1 min ago
15 26556 spring app    | spring-demo-app | started 42 secs ago | test mode
16 26707 spring app    | spring-demo-app | started 2 secs ago |
      development mode
```

There's no need to "shut down" Spring. This will happen automatically when you close your terminal. However, if you do want to do a manual shut down, use the `stop` command:

```
1 $ bin/spring stop
2 Spring stopped.
```

From within your code, you can check whether Spring is active with `if defined?(Spring)`.

Removal

To remove Spring:

- 'Unspring' your bin/ executables: `bin/spring binstub --remove --all`
- Remove spring from your Gemfile

Deployment

You must not install Spring on your production environment. To prevent it from being installed, run the `bundle config set without 'development test'` before `bundle install` command which is used to install gems on your production machines:

```
1 $ bundle config set without 'development test'
2 $ bundle install
```

Commands

rake

Runs a rake task. Rake tasks run in the `development` environment by default. You can change this on the fly by using the `RAILS_ENV` environment variable. The environment is also configurable with the `Spring::Commands::Rake.environment_matchers` hash. This has sensible defaults, but if you need to match a specific task to a specific environment, you'd do it like this:

```
1 Spring::Commands::Rake.environment_matchers["perf_test"] = "test"
2 Spring::Commands::Rake.environment_matchers[/^perf/]      = "test"
3
4 # To change the environment when you run `rake` with no arguments
5 Spring::Commands::Rake.environment_matchers[:default] = "development"
```

rails console, rails generate, rails runner

These execute the rails command you already know and love. If you run a different sub command (e.g. `rails server`) then Spring will automatically pass it through to the underlying `rails` executable (without the speed-up).

Additional commands

You can add these to your Gemfile for additional commands:

- `spring-commands-rspec`
- `spring-commands-cucumber`
- `spring-commands-spinach`
- `spring-commands-testunit` - useful for running `Test::Unit` tests on Rails 3, since only Rails 4 allows you to use `rake test path/to/test` to run a particular test/directory.
- `spring-commands-parallel-tests` - Adds the `parallel_*` commands from `parallel_tests`.
- `spring-commands-teaspoon`
- `spring-commands-m`
- `spring-commands-rubocop`
- `spring-commands-rackup`
- `spring-commands-rack-console`
- `spring-commands-standard`

Use without adding to bundle

If you don't want Spring-related code checked into your source repository, it's possible to use Spring without adding to your Gemfile. However, using Spring binstubs without adding Spring to the Gemfile is not supported.

To use Spring like this, do a `gem install spring` and then prefix commands with `spring`. For example, rather than running `bin/rake -T`, you'd run `spring rake -T`.

Temporarily disabling Spring

If you're using Spring binstubs, but temporarily don't want commands to run through Spring, set the `DISABLE_SPRING` environment variable.

Class reloading

Spring uses Rails' class reloading mechanism to keep your code up to date between test runs. This is the same mechanism which allows you to see changes during development when you refresh the page. However, you may never have used this mechanism with your `test` environment before, and this can cause problems.

It's important to realise that code reloading means that the constants in your application are *different objects* after files have changed:

```
1 $ bin/rails runner 'puts User.object_id'
2 70127987886040
3 $ touch app/models/user.rb
4 $ bin/rails runner 'puts User.object_id'
5 70127976764620
```

Suppose you have an initializer `config/initializers/save_user_class.rb` like so:

```
1 USER_CLASS = User
```

This saves off the *first* version of the `User` class, which will not be the same object as `User` after the code has been reloaded:

```
1 $ bin/rails runner 'puts User == USER_CLASS'
2 true
3 $ touch app/models/user.rb
4 $ bin/rails runner 'puts User == USER_CLASS'
5 false
```

So to avoid this problem, don't save off references to application constants in your initialization code.

Using Spring with a containerized development environment

As of Spring 1.7, there is some support for doing this. See this example repository for information about how to do it with Docker.

Configuration

Spring will read `~/ .spring.rb` and `config/spring.rb` for custom settings. Note that `~/ .spring.rb` is loaded *before* bundler, but `config/spring.rb` is loaded *after* bundler. So if you have any `spring-commands-*` gems installed that you want to be available in all projects without having to be added to the project's Gemfile, require them in your `~/ .spring.rb`.

`config/spring_client.rb` is also loaded before bundler and before a server process is started, it can be used to add new top-level commands.

Application root

Spring must know how to find your Rails application. If you have a normal app everything works out of the box. If you are working on a project with a special setup (an engine for example), you must tell Spring where your app is located:

```
1 Spring.application_root = './test/dummy'
```

Running code before forking

There is no `Spring.before_fork` callback. To run something before the fork, you can place it in `~/ .spring.rb` or `config/spring.rb` or in any of the files which get run when your application initializes, such as `config/application.rb`, `config/environments/*.rb` or `config/initializers/*.rb`.

Running code after forking

You might want to run code after Spring forked off the process but before the actual command is run. You might want to use an `after_fork` callback if you have to connect to an external service, do some general cleanup or set up dynamic configuration.

```
1 Spring.after_fork do
2   # run arbitrary code
3 end
```

If you want to register multiple callbacks you can simply call `Spring.after_fork` multiple times with different blocks.

Watching files and directories

Spring will automatically detect file changes to any file loaded when the server boots. Changes will cause the affected environments to be restarted.

If there are additional files or directories which should trigger an application restart, you can specify them with `Spring.watch`:

```
1 Spring.watch "config/some_config_file.yml"
```

By default, Spring polls the filesystem for changes once every 0.2 seconds. This method requires zero configuration, but if you find that it's using too much CPU, then you can use event-based file system listening by installing the `spring-watcher-listen` gem.

Quiet output

To disable the “Running via Spring preloader” message which is shown each time a command runs:

```
1 Spring.quiet = true
```

You can also set the initial state of the `quiet` configuration option to true by setting the `SPRING_QUIET` environment variable before executing Spring. This is useful if you want to set quiet mode when invoking the Spring executable in a subprocess, and cannot or prefer not to set it programmatically via the `Spring.quiet` option in `~/ .spring.rb` or the app's `config/spring.rb`.

Environment variables

The following environment variables are used by Spring:

- `DISABLE_SPRING` - If set, Spring will be bypassed, and your application will boot in a foreground process
- `SPRING_LOG` - The path to a file which Spring will write log messages to.

-
- `SPRING_TMP_PATH` - The directory where Spring should write its temporary files (a pidfile and a socket). By default, we use the `XDG_RUNTIME_DIR` environment variable, or else `Dir.tmpdir`, and then create a directory in that named `spring-$UID`. We don't use your Rails application's `tmp/` directory because that may be on a filesystem which doesn't support UNIX sockets.
 - `SPRING_APPLICATION_ID` - Used to identify distinct Rails applications. By default, it is an MD5 hash of the current `RUBY_VERSION`, and the path to your Rails project root.
 - `SPRING_SOCKET` - The path which should be used for the UNIX socket which Spring uses to communicate with the long-running Spring server process. By default, this is `SPRING_TMP_PATH/SPRING_APPLICATION_ID`.
 - `SPRING_PIDFILE` - The path which should be used to store the pid of the long-running Spring server process. By default, this is related to the socket path; if the socket path is `/foo/bar/spring.sock` the pidfile will be `/foo/bar/spring.pid`.
 - `SPRING_QUIET` - If set, the initial state of the `Spring.quiet` configuration option will default to `true`.
 - `SPRING_SERVER_COMMAND` - The command to run to start up the Spring server when it is not already running. Defaults to `spring _[version]_ server --background`.

Troubleshooting

If you want to get more information about what Spring is doing, you can run Spring explicitly in a separate terminal:

```
1 $ spring server
```

Logging output will be printed to stdout. You can also send log output to a file with the `SPRING_LOG` environment variable.