Ruby on Rails: Start your Engines

Luís Ferreira

October 9, 2012

Contents

Chapter 1

A little bit of History

Rails applications have evolved a whole lot throughout the years so, before even talking about engines let's take a step back and look at that nasty beast that were plugins.

Plugins and Engines are very much alike in terms of the problem then intend to solve, that is when you want to experiment on new stuff without hurting the stable code base or just modularize your code so that it is more easily tested and maintained. They do, however, have some differences in how the solve this.

Prior to rails 3 plugins would live in the `vendor/plugin` directory and they would work fine if what you wanted was to extend `ActiveRecord` or `String` or any other class. On the other if you would like to add an actual model, controller, asset or even migration, brace yourself, your in for a boilerplate nightmare.

Adding a controller would involve creating a generator that would copy it to the correct folder in the app, and the same goes for assets and migrations. This not only is a tedious job as it defeats one of the reasons we are using plugins in the first place, since it merges the plugin code with the application code. What a mess!

As you might imagine these are just some of the more obvious problems with rails 2 plugins and quite frankly I think this is why nobody really used them.

When rails 3 came out the concept of engines came with it, these were plugins but were bundled in gem. They were not quite as spectacular as they are today and I will not give them too much attention, I just want to mention a gem that was written by José Valim, a core Rails contributor, called Enginex. What this gem does is scaffold an engine with all the structure you need, as well as some very helpful boilerplate code.

At the time rails 3.1 was released, Enginex was bundled together with some other nice tweaks. This was the dawn of the age of the Engines!

Chapter 2

Engines

Here is were we get into the good stuff, creating an engine.

2.1 How to create an engine

First of all you need to understand the difference between the two very distinct types of engines, the full engine and the mountable app.

The first is what most closely resembles a plugin as the parent application inherits the routes from the engine. It is not necessary to specify anything in parent app's routes file, since specifying the gem in Gemfile is enough for the parent app to inherit the models, routes, controllers, etc. This means that these models and controllers will not be namespaced and it's names may collide with the app's.

The later is namespaced by default and the engine must be mounted on the parent app, so all the models, controllers, etc, are included and namespaced as EngineName::ClassName.

So, with this in mind you can now choose which is the engine type that best suits your need, but from my experience a mountable app is the right fit for most of the cases, since it can be included into any project without the problem of names clashing.

2.1.1 Full Engine

To create a full engine just go to your favorite shell and run:

```
rails plugin new forum --full
```

This will create what looks like a stripped rails app, with an app, config, lib and test folders. For now we'll focus on the lib folder as the other folders are pretty much the same as in a normal rails app.

In the lib folder you have another one with the name of the engine, in this example it is forum, which has two very important files. One called engine.rb where the engine's specific configurations live, as well as the requirement of all the needed gems to startup the app. The other one is the version.rb where the version of the engine is defined, remember that an engine is also a gem and this version is needed if you want to actually release it.

The lib folder also has another one called tasks for the engine's rake tasks. And this is pretty much it for a full engine, now it is up to you to write the code you want and them include it in the app (Chapter 4)

2.2 Common patterns and best practices

2.3 Extending the app with metaprogramming

Chapter 3

Testing

How to test an engine

3.1 Setting it up

3.2 RSpec

3.3 Cucumber

Chapter 4

Deploying

How to merge with the app, routes, load seeds, migrations

4.1 Precompile Assets

If there is a dependency between js in the engine, must precompile assets in order not to have conflicts.