Episode 242

Thor

All Rails developers will be familiar to some degree with Rake. Rake was originally written as an alternative to the `make` command but in Rails it's mostly used for small administrative scripts. As a scripting solution Rake can be a little limiting, for example passing arguments to a Rake task is not pretty and usually we have to resort to passing them as environment variables. Another problem with Rake is that tasks cannot easily be made global, they are always local to the current project. Tools such as Sake[1] get around this problem but then you require that additional tool.

In this episode we'll take a look a Thor[2]. Thor is an alternative to Rake which doesn't have the limitations mentioned above. It is included as a dependency in Rails so if you have Rails installed then you automatically have Thor as Rails' generators make use of it. Therefore learning Thor will help if you need to create any generators.

Thor can be run from the command line and if we run `thor help` we'll get a list of the options that it supports.

```
$ thor help
Tasks:
  thor help [TASK]     # Describe available tasks or one specific
task
  thor install NAME    # Install an optionally named Thor file
into your system tasks
  thor installed       # List the installed Thor modules and tasks
  thor list [SEARCH]   # List the available thor tasks (--
substring means .*SEARCH)
  thor uninstall NAME  # Uninstall a named Thor module
  thor update NAME     # Update a Thor file from its original
location
  thor version         # Show Thor version
```

We don't have any Thor scripts of our own yet. Let's create one now.

---

[1] http://rubygems.org/gems/sake

[2] https://github.com/wycats/thor

A Script to Copy Configuration Files

If we're going to create a Thor script we might as well create one that does something useful. It is normal in a Rails application to not put certain configuration files under source control as they contain sensitive information such as passwords. We're going to create some example configuration files in an `examples` subdirectory then create a Thor script that will copy these files over into the `config` directory.

Thor scripts can be put in an application's `lib/tasks` directory, like Rake tasks, so we'll create a new file in this directory called `setup.thor`. A Thor script is a class that inherits from `Thor` and the name of the class will become the namespace for the commands. Each method in the class becomes a command so long as it has a description. This is defined using the `desc` method, which takes two arguments: the name of the command and a description. We'll start by creating a simple command called `config` that will output a line of text.

`/lib/tasks/setup.thor`

```
class Setup < Thor

  desc "config", "copy configuration files"
  def config
    puts "running config"
  end
end
```

We can run the command by running `thor setup:config`. This calls the `config` method and we'll see the output printed in the terminal window.

```
$ thor setup:config
running config
```

To see the list of available commands we can run `thor list`.

```
$ thor list
setup
-----
thor setup:config  # copy configuration files
```

In the output is the command we've just written along with its description.

Copying Files

Let's make the `config` command do something useful. The code below loops through each file in the `/config/examples` directory and copies it to `/config`, skipping the file if it already exists there.

`/lib/tasks/setup.thor`

```
class Setup < Thor

  desc "config", "copy configuration files"
  def config
    Dir["config/examples/*"].each do |source|
      destination = "config/#{File.basename(source)}"
      if File.exist?(destination)
        puts "Skipping #{destination} because it already exists"
      else
        puts "Generating #{destination}"
        FileUtils.cp(source, destination)
      end
    end
  end
end
```

If we run the `config` command again now it should copy the files over.

```
$ thor setup:config
Generating config/database.yml
Generating config/private.yml
```

When we run the command again it will skip the files as they already exist in the `config` directory.

```
$ thor setup:config
Skipping config/database.yml because it already exists
Skipping config/private.yml because it already exists
```

It would be useful if we could pass in a `--force` option so that if the files exist in the destination directory they are still replaced. We can do this by calling `method_options` before the method definitions to define the options.

/lib/tasks/setup.thor

```
class Setup < Thor

  desc "config", "copy configuration files"
  method_options :force => :boolean
  def config
    Dir["config/examples/*"].each do |source|
      destination = "config/#{File.basename(source)}"
      FileUtils.rm(destination) if options[:force]
      if File.exist?(destination)
        puts "Skipping #{destination} because it already exists"
      else
        puts "Generating #{destination}"
        FileUtils.cp(source, destination)
      end
    end
  end
end
```

We can add as many options as we want here and there are a number of different types that are supported: strings, numbers and so on. To get the value for a given option we call `options` and in the code above we use `options[:force]` to read the `:force` option and delete the file if `force` has been set.

If we run the file with the `--force` option now the existing files are overwritten.

```
$ thor setup:config --force
Generating config/database.yml
Generating config/private.yml
```

Adding Options

Any additional arguments that we pass to `thor` are passed to the method. Let's say that want a way to customize the files that are copied over so that if we just want `private.yml` to be copied we could run

```
$ thor setup:config private.yml
```

This argument will be passed to the `config` method. We don't want to be forced to specify a file name so we'll give the argument a default value of `"*"` so that all of the files are copied. It's good to keep the documentation accurate so we'll also update the description so that it mentions the `NAME` argument.

`/lib/tasks/setup.thor`

```
class Setup < Thor

  desc "config [NAME]", "copy configuration files"
  method_options :force => :boolean
  def config(name = "*")
    Dir["config/examples/#{name}"].each do |source|
      destination = "config/#{File.basename(source)}"
      FileUtils.rm(destination) if options[:force]
      if File.exist?(destination)
        puts "Skipping #{destination} because it already exists"
      else
        puts "Generating #{destination}"
        FileUtils.cp(source, destination)
      end
    end
  end
end
```

We'll call the command with an argument now to see if it works.

```
$ thor setup:config private.yml
Skipping config/private.yml because it already exists
```

It does, and also works if we can also specify a name in conjunction with `--force`.

```
$ thor setup:config  private.yml --force
Generating config/private.yml
```

Installing Commands Globally

Our script is now pretty useful and we'd like to use it other Rails applications. Thor makes this easy: all we have to do is call `thor install <path_to_file>` and this will install the command into the system Thor commands.

```
$ thor install lib/tasks/setup.thor
Your Thorfile contains:
class Setup < Thor

  desc "config [NAME]", "copy configuration files"
  method_options :force => :boolean
  def config(name = "*")
    Dir["config/examples/#{name}"].each do |source|
      destination = "config/#{File.basename(source)}"
      FileUtils.rm(destination) if options[:force]
      if File.exist?(destination)
        puts "Skipping #{destination} because it already exists"
      else
        puts "Generating #{destination}"
        FileUtils.cp(source, destination)
      end
    end
  end
endDo you wish to continue [y/N]? y
Please specify a name for lib/tasks/setup.thor in the system
repository [setup.thor]:
Storing thor file in your system repository
```

Once it's been installed we can run `thor list` from any directory and we'll see the command listed.

```
$ cd ~
$ thor list
setup
-----
thor setup:config [NAME]  # copy configuration files
```

Our command is now globally available and we can use it in any of our Rails apps.

Accessing a Rails application from Thor

There are a couple of other things we'll demonstrate in Thor and for these we'll create a new command in our `Setup` class. This command will generate some records in our database so we'll call it `populate`. Our application is a blogging app with an `Article` model and `populate` will create ten articles for us.

`/lib/tasks/setup.thor`

```
class Setup < Thor

  desc "config [NAME]", "copy configuration files"
  method_options :force => :boolean
  def config(name = "*")
     # Config method body omitted.
  end

  desc "populate", "generate records"
  def populate
    10.times do |num|
      puts "Generating article #{num}"
      Article.create!(:name => "Article #{num}")
    end
  end
end
```

If we try to run this command now we'll get an error message saying that the `Article` class cannot be found and this is because the command hasn't loaded the application's models. The Rails application isn't loaded by default in the context of a Thor command and so we need to load the application before the command tries to create an `Article`. Fortunately this is not difficult to do, all we need to do is require the `config/environment` file.

/lib/tasks/setup.thor

```
class Setup < Thor

  desc "config [NAME]", "copy configuration files"
  method_options :force => :boolean
  def config(name = "*")
     # Config method body omitted.
  end

  desc "populate", "generate records"
  def populate
    require ‘./config/environment’
    10.times do |num|
      puts "Generating article #{num}"
      Article.create!(:name => "Article #{num}")
    end
  end
end
```

If we run the command again now it should create the ten articles, after a short delay while the environment loads.

```
$ thor setup:populate
Generating article 0
Generating article 1
Generating article 2
Generating article 3
Generating article 4
Generating article 5
Generating article 6
Generating article 7
Generating article 8
Generating article 9
```

It would be good if we could make the number of articles created configurable so that we can create any number we want. We can do this by using `method_options` as we did in the `config` method.

/lib/tasks/setup.thor

```
class Setup < Thor

  desc "config [NAME]", "copy configuration files"
  method_options :force => :boolean
  def config(name = "*")
     # Config method body omitted.
  end

  desc "populate", "generate records"
  method_options :count => 10
  def populate
    require './config/environment'
    options[:count].times do |num|
      puts "Generating article #{num}"
      Article.create!(:name => "Article #{num}")
    end
  end
end
```

This time instead of specifying a type for the options we've specified a default value and Thor will infer the type from this value. Now if we pass in a count of 5, five articles will be created.

```
$ thor setup:populate --count 5
Generating article 0
Generating article 1
Generating article 2
Generating article 3
Generating article 4
```

That's it for this episode on Thor. If you want more information about it the documentation[3] is a great place to start, especially if you want to know more about passing in options.

The big question is when should you use Thor over Rake? If you're creating a simple Rails application then it's better to stick with Rake as that is the most popular and well-known. If you find yourself creating a lot of administration tasks for your Rails applications then Thor is definitely worth considering.

---

[3] https://github.com/wycats/thor#readme