WEB+DBシステム（応用編）

第5回(2016年10月20日)

TDD(Test Driven Development)

テスト駆動開発

今日の目標

- TDDとは、どんな手法かを理解する。
  - TDDの具体的な方法を試し、用語についてなじむ。
  - 「テスト駆動開発」の考え方になじむ。
  - TDD環境のRSpecを導入し、merchandisesテーブルでRSpecによる手順を試み、「検証」や「テスト」などの位置づけを把握する。
  - 「用語」の意味を理解する。

本日のスライドについて

Rubyist Magaine の記事

スはスペックのス【第 1 回】RSpec の概要と、RSpec on Rails（モデル編）

スはスペックのス【第 2 回】RSpec on Rails（コントローラとビュー編）

http://jp.rubyist.net/magazine/?0021-Rspec

を大々的に引用します。

まともに（本気で）読むと2日がかりのWEBページですが、Ruby on Railsの応用編という位置付けの授業の一連の流れの中で、約1時間の授業の中で扱いきれる範囲でポイントを伝えるため、ベテランのお力を拝借します。

という訳で、今日の分の教科書は、上記のサイトです。

プログラマを目指す人には、長い付き合いになるはずの話題です。今日の授業だけではたぶん不十分です。Gitと同様に、様々な場面を活用して、しっかり勉強して下さい。

TDDとは？

- Test Driven Development(テスト駆動開発）

- Rubyを使う上で、最も理解して欲しい項目の一つ
  - 「簡単に作成できる」

だけではなく、

  - 「品質の高い物を作ることができる」

という部分に着目する。

TDDとは？

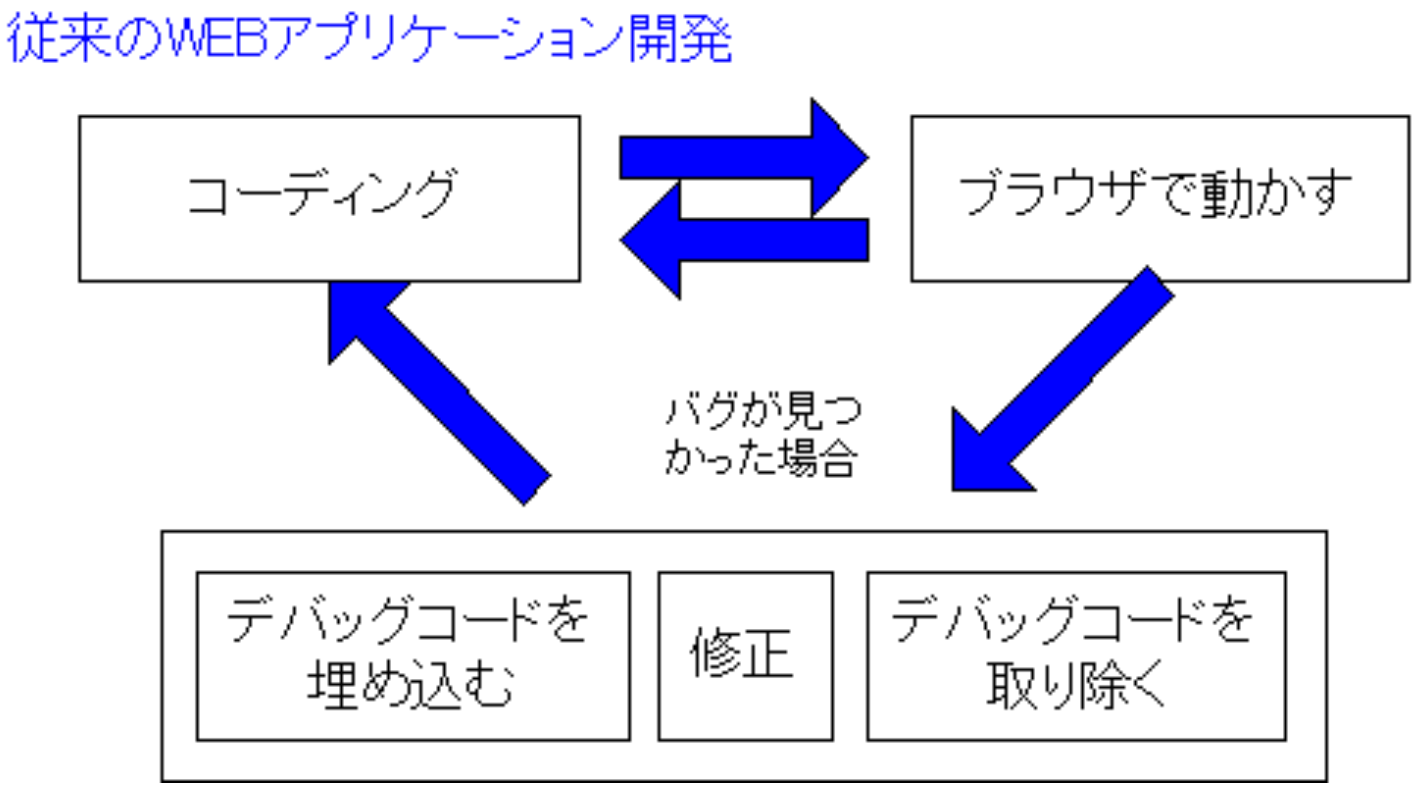
以下、引用:

TDD では、プログラム (プロダクトコード) を書くにあたって、まずテストコード を書きます。そして、テストの実行が失敗することを確認してから、テストコー ドをパスするようにプロダクトコードを書く、というプログラミングの進め方 です。

TDD は分析手法および設計技法であり、実際には開発のすべてのアクティビティを構造化するための技法である。

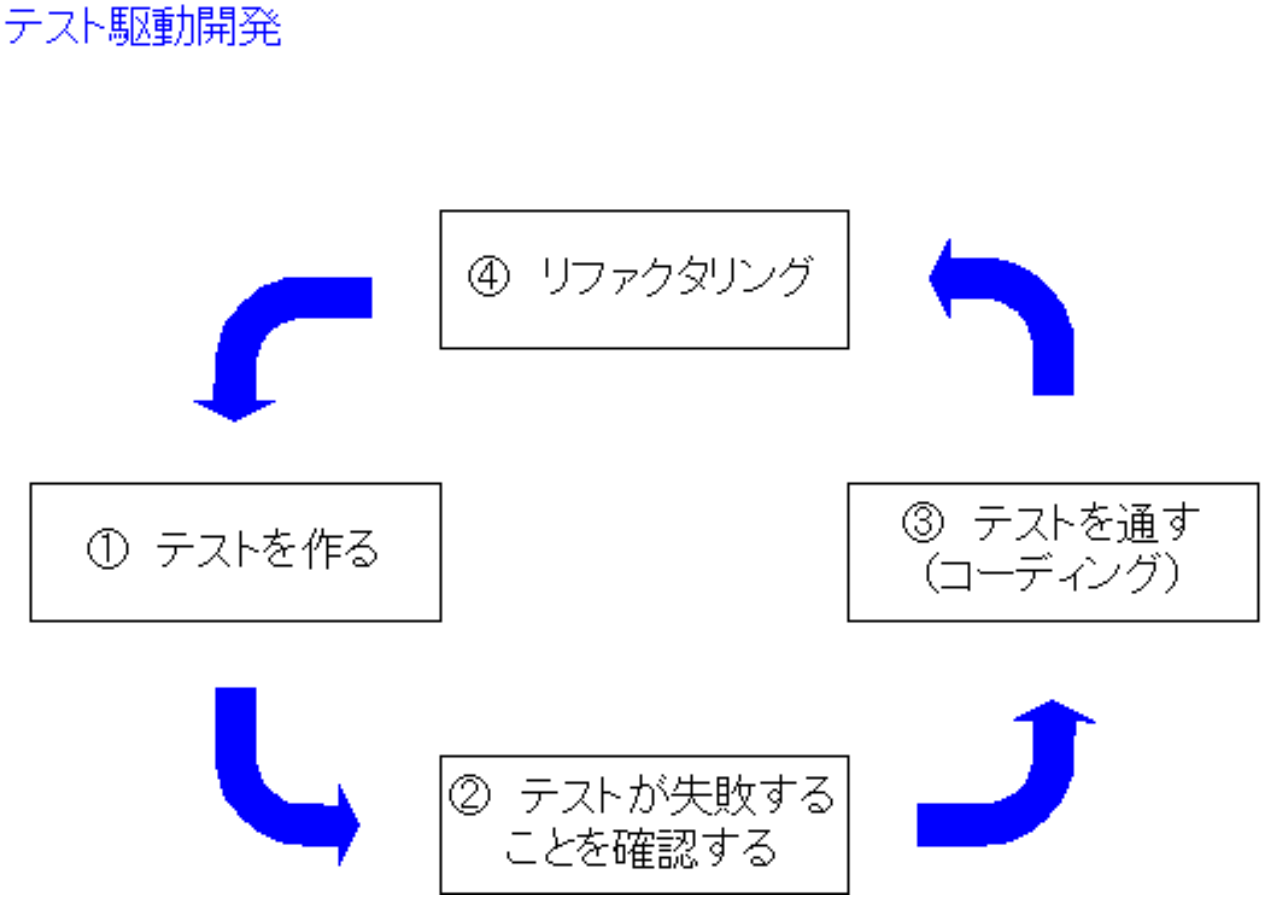
従来のWEBアプリケーション開発

- 実際に動かしてみて、動作を確認しながら試す。



テスト駆動開発の流れ

- 最初にテストの手順を決めて作る。



テストでは「失敗」にも意味がある

- コーディングを開始する前から「テスト」を用意する。
- コーディングを行う前に「テスト」を実行したなら、テストで「失敗した」ことが、「正しく動作している」ことの証明になる！
- ここでいう「失敗」は、ErrorではなくFailure

テスト導入の手順（4ステップ）

- Step 1: 「テスト」を書く
  - 「どんな動作をすべきか」、仕様を明確にしてテストを書く
- Step 2: 「テスト」が失敗することを確認する。(RED)
  - プログラムを書く前に、「テストスクリプト」を実行して、
    テストが失敗することを確認する。（全部が、ErrorではなくFailure）
    （テストスクリプトのバグを取る）
- Step 3: コーディングを行う。
  - テストに成功するようにプロダクトコードを書く (**GREEN**)
- Step 4: リファクタリング
  - 「テストが成功する」状態のまま、コードをきれいにする。

テストの二つの環境

Ruby on Railsでは、Test:Unitというテスト環境があります。

ですが、RSpecが多く使われているようですので、RSpecでTDDを導入します。

理由です。以下引用:

Test::Unit が「テスティングフレームワーク」であるのに対して、Rspec が「統合テスティング環境」であることを目指しているという点です。

具体的にいえば、RSpec は、振舞定義用の DSL の提供に加えて、様々なテスト関連ライブラリや周辺ツールを統合したり、標準サポートしたりといった試みを 積極的に続けています。

テスト用のデータ

テストでは、「成功するためのデータ」と、「エラー処理を確認するためのデータ」の２種類があります。

こうしたデータが入力されたり、データベースにそうしたデータがあったりした場合の、回復処理などを確認するためには、一貫したデータが与えられていなければなりません。

Rspec + Factory Girl

テスト時のモデルデータを用意するための仕組み

特徴としては

- Rubyコードで定義（YAMLやCSVではない）
- 関連のメンテナンスが楽
- 定義の継承もできる
- 同じ定義から連続的なデータを生成できる（シーケンス）

リレーションのメンテナンスが楽、という一言に惹かれました。

http://www.func09.com/wordpress/archives/532

TDDの原則

1. テストに失敗しない限り、プロダクトコードを書いてはいけない。

2. プロダクトコードはテストを通るように書く。

3. テストは少しずつ書き進めていく。

最初は、失敗しないテストから・・・

授業では、ruby on railsのプログラムの動作を
順に画面で見られるようにしながら、導入しました。

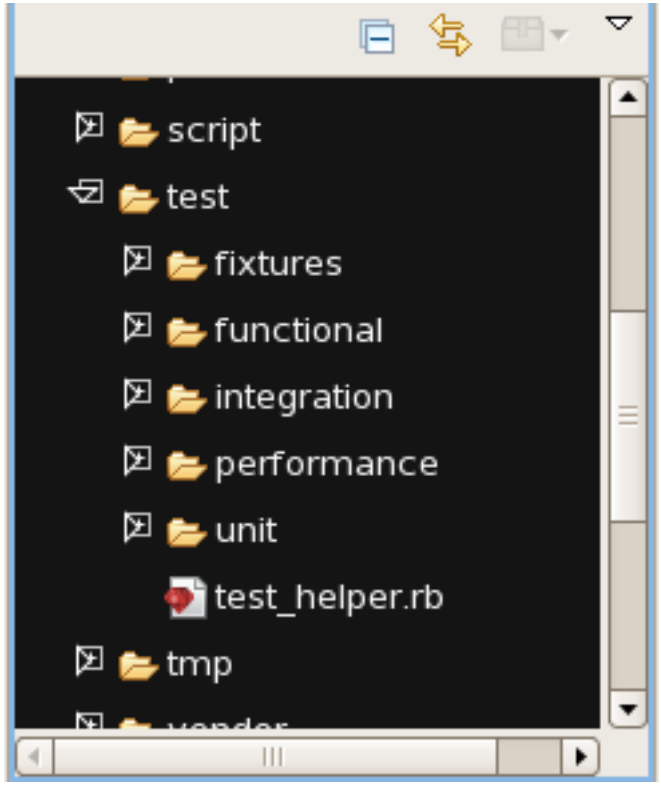
現在既に、userとroleの間にリレーションも貼ってあったりします。

ですので、一度書いたプログラムを消さない限り、テストには失敗しません・・・。最初は、この部分のテストをどう書くか、から着手します。

今後新規に書くプログラムでは、「まず失敗させる」という手順を守り、導入済みのモデルでは例外とするということで、理解して下さい。

参考資料:テスト用のファイル

scaffoldingした際に、自動生成された
testディレクトリの下に生成されます。
今回はこれらのファイルは使用しません。



今日の実習課題

- 今日の実習で行うこと。
  - Merchandises (商品)のテーブルの、Unitに対するRSpecでのテストを記述し、どんな書き方をして、どんな結果が得られるか、確認します。
    - 皆さんは、各自で設定したテーブルで読み替えて下さい。
      - 今後、新規にテーブルを登録する時は、テストから書くように、教材を用意して行きます。この場合、最初は必ず「失敗」(Failure)します。(心意気として・・・)
      - ERRORとFailureは違う、ということを理解していって下さい。

何をテストするか。

商品名、価格の登録などが正しく行われるか。

項目が一部記載されていないデータが、エラーとなるか。

roleのコードが0の場合はエラーとするか。

データの生成後の画面の流れが正しいか。

などのテストコードを書いて、走らせます。

RSpec のインストール

インストールされていない場合、

```
rspec –v
```

に対して、「コマンドがない」と言われます。

そこで、GNOME端末で

```
gem install rspec-rails
```

と入力します。

```
rspec –v
```

と入力されて、バージョン表示が得られればOKです。

```
[WebDB@cisnote vegetable-market]$ rspec -v
3.5.2
[WebDB@cisnote vegetable-market]$
```

Rspecのインストール

```
[root@cisnote vegetable-market]# rspec -v
bash: rspec: コマンドが見つかりません
[root@cisnote vegetable-market]# gem install rspec-rails
Fetching: rspec-support-3.3.0.gem (100%)
Successfully installed rspec-support-3.3.0
Fetching: diff-lcs-1.2.5.gem (100%)
Successfully installed diff-lcs-1.2.5
Fetching: rspec-mocks-3.3.2.gem (100%)
Successfully installed rspec-mocks-3.3.2
Fetching: rspec-expectations-3.3.1.gem (100%)
Successfully installed rspec-expectations-3.3.1
Fetching: rspec-core-3.3.2.gem (100%)
Successfully installed rspec-core-3.3.2
Fetching: rspec-rails-3.3.3.gem (100%)
Successfully installed rspec-rails-3.3.3
Parsing documentation for rspec-support-3.3.0
Installing ri documentation for rspec-support-3.3.0
Parsing documentation for diff-lcs-1.2.5
Installing ri documentation for diff-lcs-1.2.5
Parsing documentation for rspec-mocks-3.3.2
Installing ri documentation for rspec-mocks-3.3.2
Parsing documentation for rspec-expectations-3.3.1
Installing ri documentation for rspec-expectations-3.3.1
Parsing documentation for rspec-core-3.3.2
Installing ri documentation for rspec-core-3.3.2
Parsing documentation for rspec-rails-3.3.3
Installing ri documentation for rspec-rails-3.3.3
Done installing documentation for rspec-support, diff-lcs, rspec-mocks, rspec-ex
pectations, rspec-core, rspec-rails after 7 seconds
6 gems installed
[root@cisnote vegetable-market]# rspec -v
3.3.2
[root@cisnote vegetable-market]#
```

RSpecのGemfileへの追加

(プロジェクトフォルダ)/Gemfileを開いて以下の記述を追加します。

```
group :test, :development do
    gem "rspec-rails"
    gem "rails-controller-testing"
    gem "factory_girl_rails"
    gem "capybara"
    gem "database_cleaner"
end
```

```
17
18 gem 'devise'
19 gem 'bcrypt', '~>3.1.11'
20
21 group :test, :development do
22     gem "rails-controller-testing"
23     gem "rspec-rails"
24     gem "factory_girl_rails"
25     gem "capybara"
26     gem "database_cleaner"
27 end
28
29 # Use jquery as the JavaScript library
30 gem 'jquery-rails'
```

何をインストールしているか

RSpec　--　テストの自動化

FactoryGirl　--　テストデータの合成

Capybara　--　WEB画面でのユーザ入力を置換

DatabaseCleaner – テストに使ったデータを戻す

Bundle install

Gemfileを編集したら、いつもの通り

bundle install

を実行します。

bundle install

```
Using rack-test 0.6.3
Using warden 1.2.6
Using sprockets 3.7.0
Using websocket-driver 0.6.4
Using mime-types 3.1
Using coffee-script 2.4.1
Using uglifier 3.0.2
Using rb-inotify 0.9.7
Using therubyracer 0.12.2
Installing rspec-core 3.5.4
Using rspec-expectations 3.5.0
Using rspec-mocks 3.5.0
Using turbolinks 5.0.1
Using activesupport 5.0.0.1
Using loofah 2.0.3
Using xpath 2.0.0
Using mail 2.6.4
Using listen 3.0.8
Using rails-dom-testing 2.0.1
Using globalid 0.3.7
Using activemodel 5.0.0.1
Using factory_girl 4.7.0
Using jbuilder 2.6.0
Using rails-html-sanitizer 1.0.3
Installing capybara 2.10.1
Using spring-watcher-listen 2.0.0
Using activejob 5.0.0.1
Using activerecord 5.0.0.1
Using actionview 5.0.0.1
Using actionpack 5.0.0.1
Using actioncable 5.0.0.1
Using actionmailer 5.0.0.1
Using railties 5.0.0.1
Using sprockets-rails 3.2.0
Using coffee-rails 4.2.1
Using responders 2.3.0
Using factory_girl_rails 4.7.0
Using jquery-rails 4.2.1
Installing rspec-rails 3.5.2
Using web-console 3.3.1
Using rails 5.0.0.1
Using sass-rails 5.0.6
Using devise 4.2.0
Bundle complete! 22 Gemfile dependencies, 83 gems now installed.
Use `bundle show [gemname]` to see where a bundled gem is installed.
[WebDB@cisnote vegetable-market]$
```

RSpecの環境を準備

次に、以下のコマンドを実行します。

rails generate rspec:install

```
[WebDB@cisnote vegetable-market]$ rails g rspec:install
Running via Spring preloader in process 3206
      create  .rspec
      create  spec
      create  spec/spec_helper.rb
      create  spec/rails_helper.rb
[WebDB@cisnote vegetable-market]$
```

Specファイルの生成

rspec:install以後は、rails generateで生成されたモデルや、view、controllerなどに対してspecファイルも生成されますが、rspec:install以前に生成されたファイルに対しては、個別に指定する必要があります。

RSpec用フォルダ + modelテスト用

プロジェクトrootの下に、specというフォルダが出来ていることを確認します。

ここで、モデルのテストファイルを作成します。

rails g rspec:model merchandise

こうすると、specの下にサブフォルダmodelsとmerchandise_spec.rbファイルが生成されます。

role, userについても同様のコマンドを反復します。

```
[WebDB@cisnote vegetable-market]$ rails g rspec:model merchandise
Running via Spring preloader in process 3228
      create  spec/models/merchandise_spec.rb
      invoke  factory_girl
      create    spec/factories/merchandises.rb
[WebDB@cisnote vegetable-market]$ rails g rspec:model user
Running via Spring preloader in process 3247
      create  spec/models/user_spec.rb
      invoke  factory_girl
      create    spec/factories/users.rb
[WebDB@cisnote vegetable-market]$ rails g rspec:model role
Running via Spring preloader in process 3258
      create  spec/models/role_spec.rb
      invoke  factory_girl
      create    spec/factories/roles.rb
[WebDB@cisnote vegetable-market]$
```

現状での注意点

rails g rspec:model merchandise

のコマンドでは、rspecのgeneratorではmerchandisesというクラスが存在しているかどうかは確認せずに、「ひな形」を生成します。

ですから、hogehogeなどというmodelが存在しなくても、

rails g rspec:model hogehoge

と入力すると、models/hogehoge_spec.rbというファイルが生成されてしまいます。システムはチェックしてくれないので、入力ミスには注意して下さい。

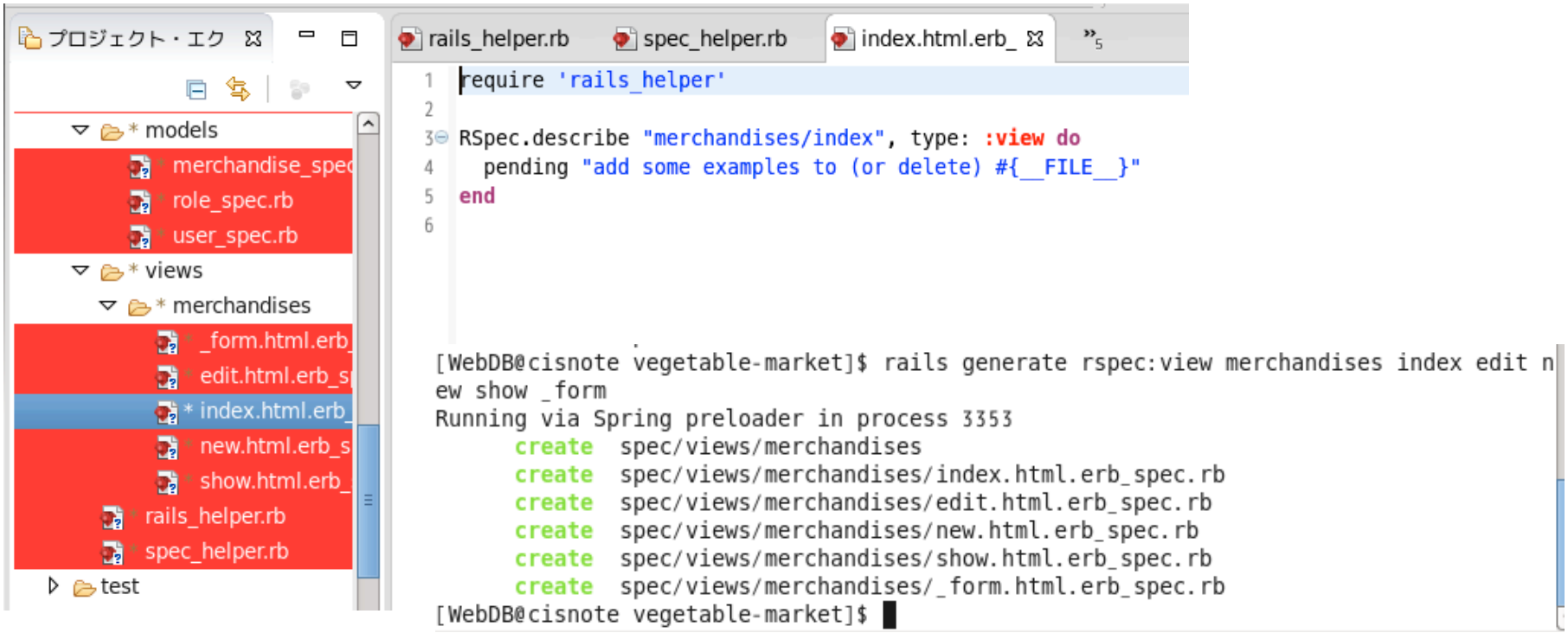
（特に、入力のスペルミス）

Viewsテスト用

次に、viewを生成します。Viewでは

rails g rspec:view merchandises index edit new show _form

と入力すると、index.html.erbや_form.html.erbなど用のテストファイルが作成されます。また、必要なフォルダも作成されます。



Viewsテスト用

同様の手順で、usersのindex, edit, new, show, _form, rolesのindex, edit, new, show, _formについても、生成します。

モデルのときは「単数形」だったのが、ここでは「複数形」なのに気をつけて下さい。

```
[WebDB@cisnote vegetable-market]$ rails generate rspec:view users index edit new show
 _form
Running via Spring preloader in process 3378
      create  spec/views/users
      create  spec/views/users/index.html.erb_spec.rb
      create  spec/views/users/edit.html.erb_spec.rb
      create  spec/views/users/new.html.erb_spec.rb
      create  spec/views/users/show.html.erb_spec.rb
      create  spec/views/users/_form.html.erb_spec.rb
[WebDB@cisnote vegetable-market]$ rails generate rspec:view roles index edit new show
 _form
Running via Spring preloader in process 3401
      create  spec/views/roles
      create  spec/views/roles/index.html.erb_spec.rb
      create  spec/views/roles/edit.html.erb_spec.rb
      create  spec/views/roles/new.html.erb_spec.rb
      create  spec/views/roles/show.html.erb_spec.rb
      create  spec/views/roles/_form.html.erb_spec.rb
[WebDB@cisnote vegetable-market]$
```

Views テスト用

```
[WebDB@cisnote vegetable-market]$ rails generate rspec:view merchandises index edit n
ew show _form
Running via Spring preloader in process 3353
      create  spec/views/merchandises
      create  spec/views/merchandises/index.html.erb_spec.rb
      create  spec/views/merchandises/edit.html.erb_spec.rb
      create  spec/views/merchandises/new.html.erb_spec.rb
      create  spec/views/merchandises/show.html.erb_spec.rb
      create  spec/views/merchandises/_form.html.erb_spec.rb
[WebDB@cisnote vegetable-market]$ rails generate rspec:view users index edit new show
 _form
Running via Spring preloader in process 3378
      create  spec/views/users
      create  spec/views/users/index.html.erb_spec.rb
      create  spec/views/users/edit.html.erb_spec.rb
      create  spec/views/users/new.html.erb_spec.rb
      create  spec/views/users/show.html.erb_spec.rb
      create  spec/views/users/_form.html.erb_spec.rb
[WebDB@cisnote vegetable-market]$ rails generate rspec:view roles index edit new show
 _form
Running via Spring preloader in process 3401
      create  spec/views/roles
      create  spec/views/roles/index.html.erb_spec.rb
      create  spec/views/roles/edit.html.erb_spec.rb
      create  spec/views/roles/new.html.erb_spec.rb
      create  spec/views/roles/show.html.erb_spec.rb
      create  spec/views/roles/_form.html.erb_spec.rb
[WebDB@cisnote vegetable-market]$
```

Controllerテスト用

Controllerでは

rails g rspec:controller merchandises

と入力します。Usersやrolesも同様です。これは、複数形です。

```
[WebDB@cisnote vegetable-market]$ rails generate rspec:controller merchandises
Running via Spring preloader in process 3550
      create  spec/controllers/merchandises_controller_spec.rb
[WebDB@cisnote vegetable-market]$ rails generate rspec:controller roles
Running via Spring preloader in process 3570
      create  spec/controllers/roles_controller_spec.rb
[WebDB@cisnote vegetable-market]$ rails generate rspec:controller users
Running via Spring preloader in process 3578
      create  spec/controllers/users_controller_spec.rb
[WebDB@cisnote vegetable-market]$
```

RSpecでの考え方

プログラムコードを書く前に、テストを書くのが原則

ここで、「**まずプログラムの振る舞いを動作可能なサンプルとして書くこと**」が最初になります。

このテストケースの論理的なまとまりを「**振舞(behaviour)**」と呼び、コードの検証を「**エクスペクテーション(expectation)**」と呼びます。

テスト用のテーブルを作る

Rails のmigrationは、開発用のファイルしか作っていません。そこで、テスト用のデータベースファイルを作ります。コマンドプロンプトで

rake db:migrate RAILS_ENV=test

と入力します。

エラーが出たら、何度か反復してみて下さい。

最終的に、全項目upならOKです。

```
[WebDB@cisnote vegetable-market]$ rake db:migrate:status RAILS_ENV=test

database: /home/WebDB/workspace/vegetable-market/db/test.sqlite3

 Status   Migration ID    Migration Name
--------------------------------------------------
   up     20161005120857  Create merchandises
   up     20161012091851  Create roles
   up     20161012094205  Create users
   up     20161012094340  Add devise to users

[WebDB@cisnote vegetable-market]$
```

テストが終わったら、元に戻す

テストが終わって、「テストラン」しての開発に戻る際は、

```
rake db:migrate RAILS_ENV=development
```

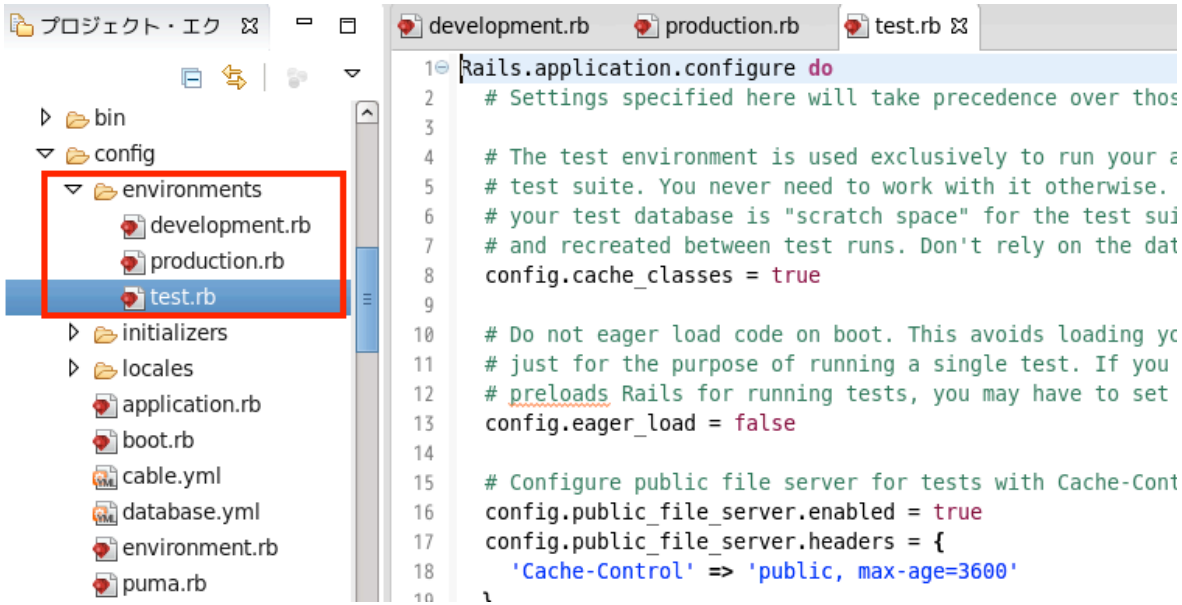
と入力して、データベースを開発用にもどします。

（今は、まだこれは打ち込みませんが、覚えておいてください。）

テストを終了して、通常の開発モードに戻った後、「migrateがうまくいかない！」質問が多くありますが、原因はこれです。

テスト用データベース生成

Railsの「実行環境」には、三つあります。
Development, Test, そしてProductionです。
Produce（商用）には、この授業では辿り着きません。
前期は、Developmentだけでした。
ただ、後期は、Testも行います。



テスト用データベース生成

```
[WebDB@cisnote vegetable-market]$ rake db:migrate RAILS_ENV=test
== 20161005120857 CreateMerchandises: migrating ===============================
-- create_table(:merchandises)
   -> 0.0034s
== 20161005120857 CreateMerchandises: migrated (0.0035s) ======================

== 20161012091851 CreateRoles: migrating =======================================
-- create_table(:roles)
   -> 0.0027s
== 20161012091851 CreateRoles: migrated (0.0028s) ==============================

== 20161012094205 CreateUsers: migrating =======================================
-- create_table(:users)
   -> 0.0041s
== 20161012094205 CreateUsers: migrated (0.0042s) ==============================

== 20161012094340 AddDeviseToUsers: migrating ==================================
-- change_table(:users)
   -> 0.0040s
-- add_index(:users, :email, {:unique=>true})
   -> 0.0037s
-- add_index(:users, :reset_password_token, {:unique=>true})
   -> 0.0012s
== 20161012094340 AddDeviseToUsers: migrated (0.0092s) =========================

[WebDB@cisnote vegetable-market]$
```

後で、元に戻すのを忘れずに・・・

rake db:migrate RAILS_ENV=test

のコマンドは、データベースを testに切り替えます。

テストを行う際は、こちらの環境で行います。

これまで作成した「開発」用に戻す場合には

rake db:migrate RAILS_ENV=development

と入力します。

そうしないと、「これまで入力したデータが消えた！」という騒ぎになります。（消えた訳ではなく、DBを切り替えただけです。）

Eclipseが真っ赤なので・・・

更新が全く反映されていないので、gitを更新します。

git add .

git commit –a –m ‘rspecファイル生成’

```
[WebDB@cisnote vegetable-market]$ git add .
[WebDB@cisnote vegetable-market]$ git commit -a -m 'rspecファイル生成'
[master 157cb90] rspecファイル生成
 31 files changed, 353 insertions(+)
 create mode 100644 .rspec
 create mode 100644 spec/controllers/merchandises_controller_spec.rb
 create mode 100644 spec/controllers/roles_controller_spec.rb
 create mode 100644 spec/controllers/users_controller_spec.rb
 create mode 100644 spec/factories/merchandises.rb
 create mode 100644 spec/factories/roles.rb
 create mode 100644 spec/factories/users.rb
 create mode 100644 spec/helpers/roles_helper_spec.rb
 create mode 100644 spec/helpers/users_helper_spec.rb
 create mode 100644 spec/models/merchandise_spec.rb
 create mode 100644 spec/models/role_spec.rb
 create mode 100644 spec/models/user_spec.rb
 create mode 100644 spec/rails_helper.rb
 create mode 100644 spec/spec_helper.rb
 create mode 100644 spec/views/merchandises/_form.html.erb_spec.rb
 create mode 100644 spec/views/merchandises/edit.html.erb_spec.rb
 create mode 100644 spec/views/merchandises/index.html.erb_spec.rb
```

まず、specファイルに名乗らせる。

Spec/modelの merchandises_spec.rbは、

```
require 'rails_helper’
RSpec.describe Merchandise, type: :model do
  pending "add some examples to (or delete) #{__FILE__}”
end
```

となっています。このままだと、英語のメッセージで馴染みがないので、生成されたspecファイルは、必要に応じて

```
RSpec.describe (クラス名), type: (種類) do
  pending “まだ、テストを記述していません。#{__FILE__}”
end
```

にしておきます。

_specファイルの修正

test.rb | merchandise_spe | edit.html.erb_s

```
1  require 'rails_helper'
2
3  RSpec.describe Merchandise, type: :model do
4    pending "add some examples to (or delete) #{__FILE__}"
5  end
6
```

を修正し、

test.rb | *merchandise_sp | edit.html.erb_s

```
1  require 'rails_helper'
2
3  RSpec.describe Merchandise, type: :model do
4    pending "まだ、テストを記述していません。#{__FILE__}"
5  end
6
```

とします。

最初のテスト検証

プロジェクトルートで、

rspec spec/*/*_spec.rb spec/*/*/*_spec.rb

とコマンド入力します。

```
[WebDB@cisnote vegetable-market]$ rspec spec/*/*_spec.rb spec/*/*/*_spec.rb
DEPRECATION WARNING: before_filter is deprecated and will be removed in Rails 5.1. Us
e before_action instead. (called from <class:RolesController> at /home/WebDB/workspac
e/vegetable-market/app/controllers/roles_controller.rb:2)
********************

Pending: (Failures listed here are expected and do not affect your suite's status)

  1) RolesHelper add some examples to (or delete) /home/WebDB/workspace/vegetable-mar
ket/spec/helpers/roles_helper_spec.rb
     # Not yet implemented
     # ./spec/helpers/roles_helper_spec.rb:14

  2) UsersHelper add some examples to (or delete) /home/WebDB/workspace/vegetable-mar
ket/spec/helpers/users_helper_spec.rb
     # Not yet implemented
     # ./spec/helpers/users_helper_spec.rb:14

  3) Merchandise まだ、テストを記述していません。 /home/WebDB/workspace/vegetable-marke
t/spec/models/merchandise_spec.rb
     # Not yet implemented
     # ./spec/models/merchandise_spec.rb:4
```

全部保留の結果

Specテストは実施していますが、全部「保留」になります。

保留になったファイルは、この教材の手順だと20個あるはずです。(記載漏れがあっても、気にしないで進めてください。)

```
  19) users/new add some examples to (or delete) /home/WebDB/workspace/vegetable-mark
et/spec/views/users/new.html.erb_spec.rb
     # Not yet implemented
     # ./spec/views/users/new.html.erb_spec.rb:4

  20) users/show add some examples to (or delete) /home/WebDB/workspace/vegetable-mar
ket/spec/views/users/show.html.erb_spec.rb
     # Not yet implemented
     # ./spec/views/users/show.html.erb_spec.rb:4


Finished in 0.01743 seconds (files took 5.41 seconds to load)
20 examples, 0 failures, 20 pending

[WebDB@cisnote vegetable-market]$
```

FactoryGirlを使う

specというディレクトリが生成されています。

このディレクトリ内の rails_helper.rb で、

```
Dir[Rails.root.join('spec/support/**/*.rb')].each { |f| require f }
```

のコメントを外し、

```
config.include FactoryGirl::Syntax::Methods
```

と、

```
require 'capybara/rails'
require 'capybara/rspec'
```

の合計3行を追加します。

WEBサイトによっては、spec_helper.rbに書き込むとしているケースもありますが、生成されているspecファイルに、require "rails_hellper"が生成されているので、rails_helper.rbの方に書き加えます。

spec/rails_helper.rb 設定

```
require 'spec_helper'
require 'rspec/rails'

require 'capybara/rails'
require 'capybara/rspec'
# Add additional requires below this line. Rails is not loaded until this point!

# Requires supporting ruby files with custom matchers and macros, etc, in
# spec/support/ and its subdirectories. Files matching `spec/**/*_spec.rb` are
# run as spec files by default. This means that files in spec/support that end
# in _spec.rb will both be required and run as specs, causing the specs to be
# run twice. It is recommended that you do not name files matching this glob to
# end with _spec.rb. You can configure this pattern with the --pattern
# option on the command line or in ~/.rspec, .rspec or `.rspec-local`.
#
# The following line is provided for convenience purposes. It has the downside
# of increasing the boot-up time by auto-requiring all files in the support
# directory. Alternatively, in the individual `*_spec.rb` files, manually
# require only the support files necessary.
#
Dir[Rails.root.join('spec/support/**/*.rb')].each { |f| require f }

# Checks for pending migration and applies them before tests are run.
# If you are not using ActiveRecord, you can remove this line.
ActiveRecord::Migration.maintain_test_schema!

RSpec.configure do |config|
  config.include FactoryGirl::Syntax::Methods

  # Remove this line if you're not using ActiveRecord or ActiveRecord fixtures
  config.fixture_path = "#{::Rails.root}/spec/fixtures"

  # If you're not using ActiveRecord, or you'd prefer not to run each of your
  # examples within a transaction, remove the following line or assign false
```

データベースクリーナの準備

spec/spec_helper.rb

に、以下の記述を追加します。

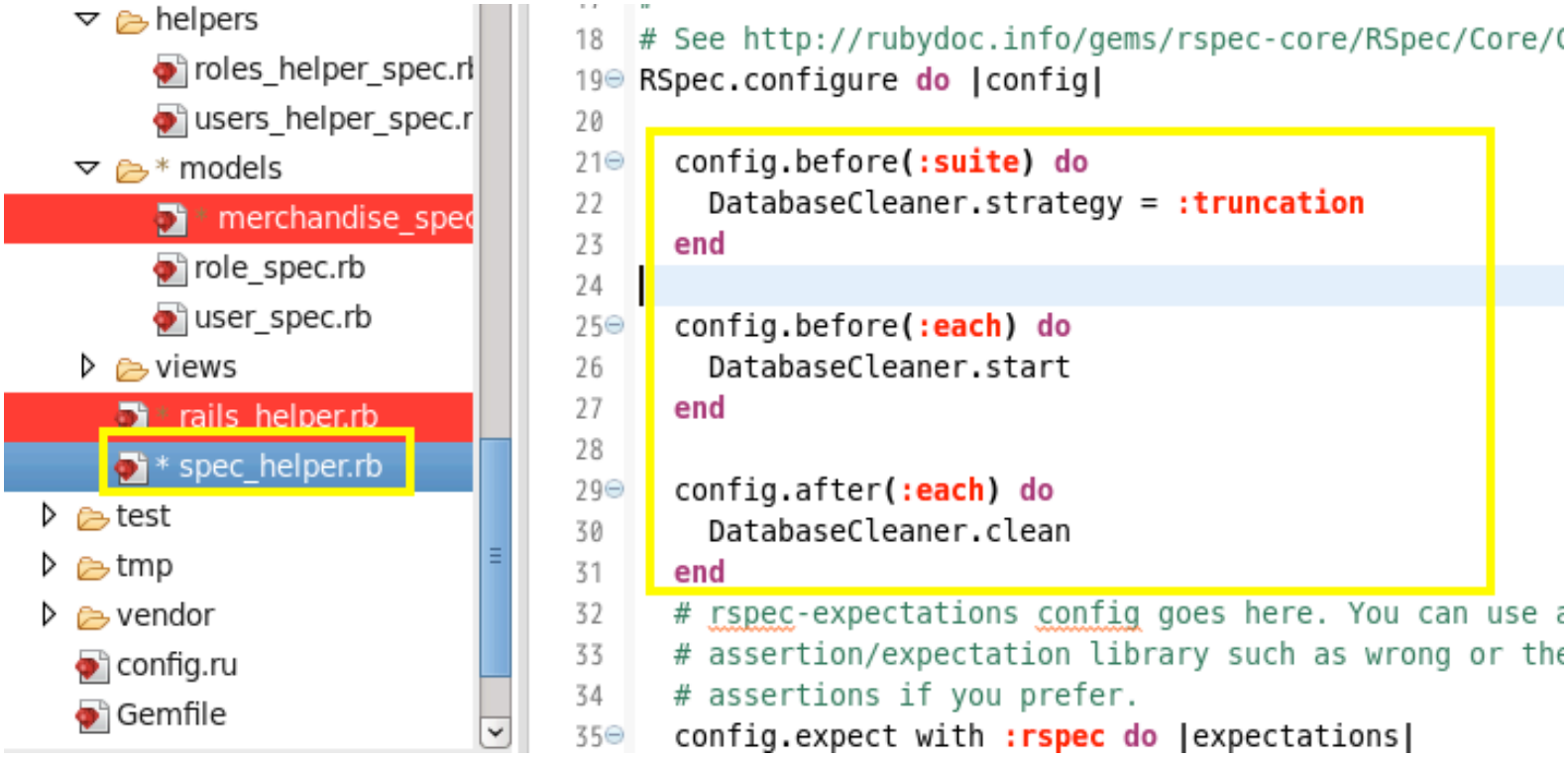
```
config.before(:suite) do
  DatabaseCleaner.strategy = :truncation
end

config.before(:each) do
  DatabaseCleaner.start
end

config.after(:each) do
  DatabaseCleaner.clean
end
```

spec/spec_helper.rb

Rspec.configure doのブロック内に書きます。



どういう「動作」をさせたいか

プログラムに、どんな「動作」をさせたいかを、最初に
　決めて、それを記述します。

これが「テスト駆動」です。

「どういう動作をさせるか、考えて決める」
　　ということは、

「設計」そのものです。

誰かが「設計」したものを、その指示に従って作るのが
　プログラマで、設計を行うのがSEです。

Roleに関する動作を列挙する

今回の例題は、以下の二つを書いてみます。

- Users::editでは、Userのrole_idは、必ずroleに存在する値である。
- Users::editでは、自分自身のhandleや、emailは、role_idが何であっても、変更できる。

前ページの2項目を、記述する。

最初は、「文章」として動作を考えます。
次に、これを「テスト項目」として記述します。

「Userのrole_idは、必ずroleに存在する値である。」
→「Userを登録する際に、role_idの値が、rolesテーブルに存在しない場合は、登録が失敗する。」
と読み替える

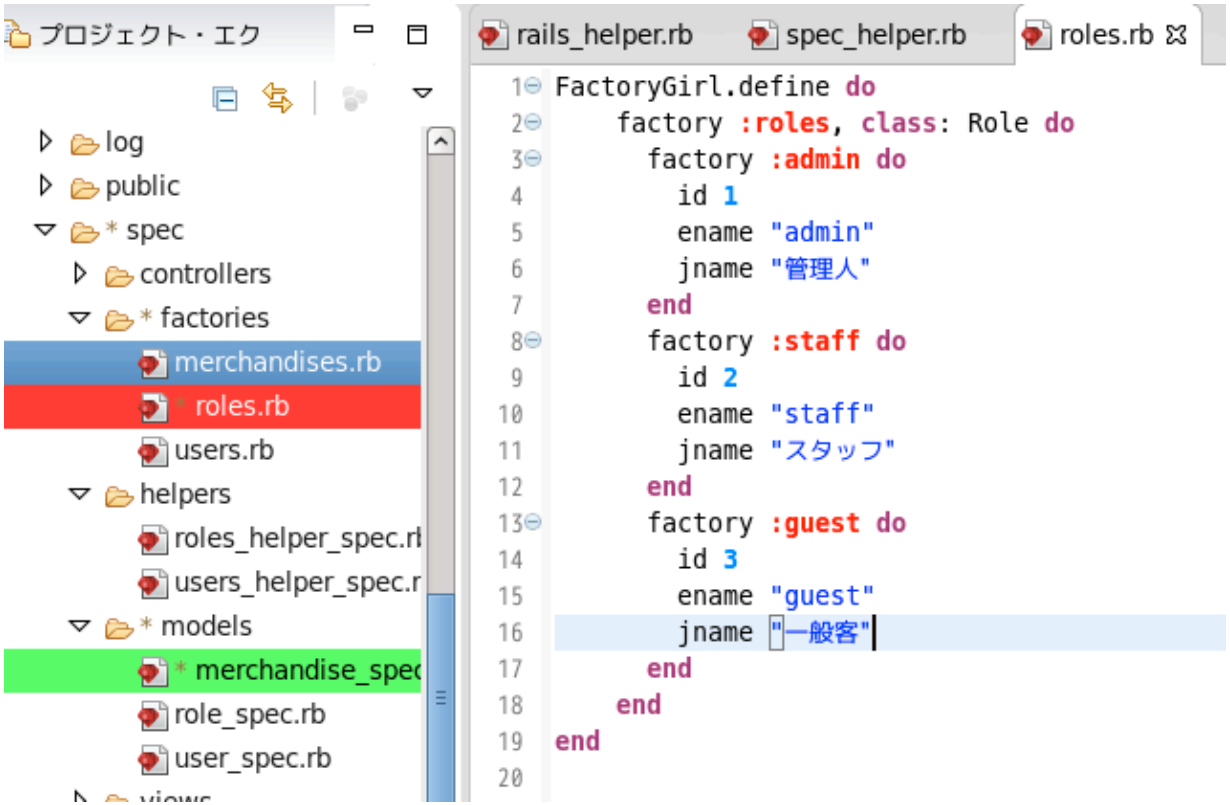
Step 1 データを準備する

FactoryGirlを用いて、データを用意します。

Spec/factories/roles.rb と users.rb
を編集します。

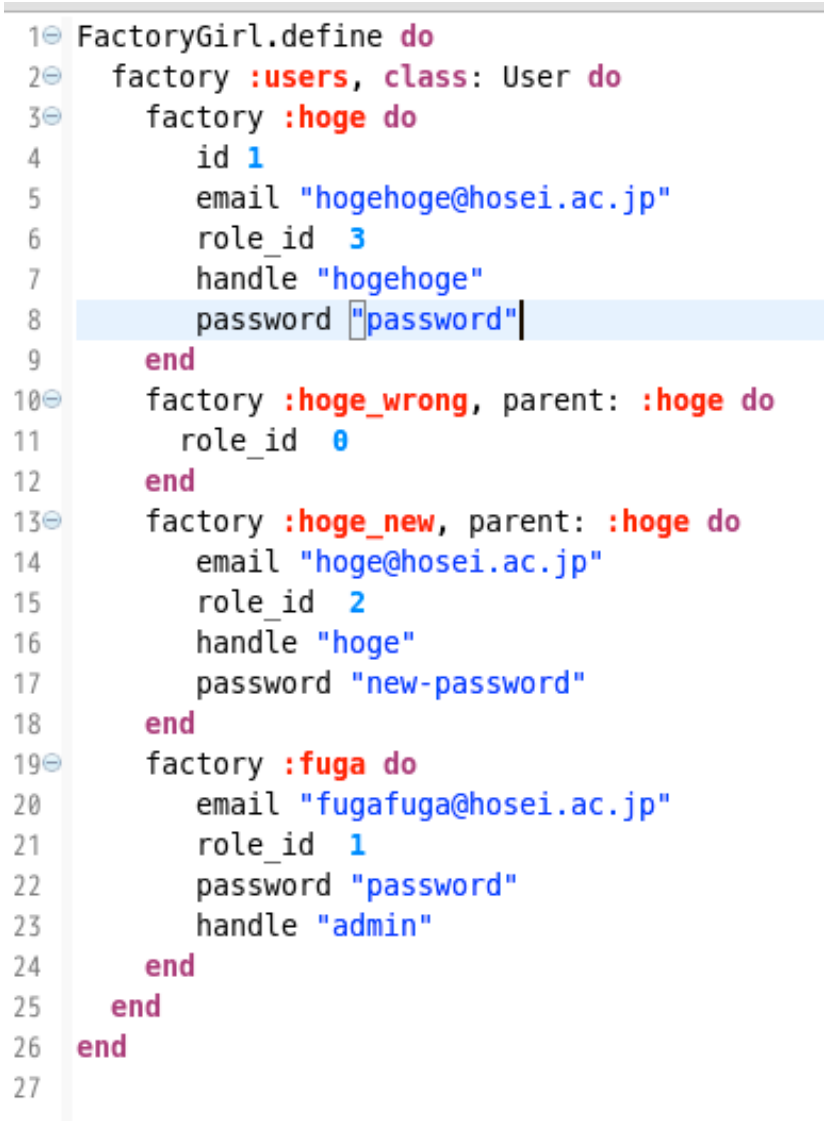
spec/factories/roles.rb

```
FactoryGirl.define do
    factory :roles, class: Role do
      factory :admin do
        id 1
        ename "admin"
        jname "管理人"
      end
      factory :staff do
        id 2
        ename "staff"
        jname "スタッフ"
      end
      factory :guest do
        id 3
        ename "guest"
        jname "一般客"
      end
    end
end
```



spec/factories/users.rb

```
FactoryGirl.define do
  factory :users, class: User do
    factory :hoge do
      id 1
      email "hogehoge@hosei.ac.jp"
      role_id  3
      handle "hogehoge"
      password "password"
    end
    factory :hoge_wrong, parent: :hoge do
      role_id  0
    end
    factory :hoge_new, parent: :hoge do
      email "hoge@hosei.ac.jp"
      role_id  2
      handle "hoge"
      password "new-password"
    end
    factory :fuga do
      email "fugafuga@hosei.ac.jp"
      role_id  1
      password "password"
      handle "admin"
    end
  end
end
```



Step 2 テストを記述する

beforeメソッドでroleなどを事前に生成します。

Afterメソッドで、データベースを空にもどします。

Describeで個々のメソッド(Action)を指定し、

It と doの間に、テストの内容をできるだけ具体的に書きます。

spec/controllers/ users_controller_spec.rb (1/2)

```
require 'rails_helper'

RSpec.describe UsersController, type: :controller do
  before(:all) do
    @admin = FactoryGirl.create(:admin)
    @staff = FactoryGirl.create(:staff)
    @guest = FactoryGirl.create(:guest)
    @hoge  = FactoryGirl.create(:hoge)
  end
  after(:all) do
    DatabaseCleaner.clean
  end

  describe "Usersのindex画面では、" do
    it "indexテンプレートで表示する。" do
      get :index
      expect(response).to render_template :index
    end
  end
```

spec/controllers/ users_controller_spec.rb (2/2)

```
  describe "#update" do
    context( "role_idの値が存在しない場合") do
      it "データは更新されず、editの画面に戻る。" do
        get :edit, id: @hoge
        patch :update, id: @hoge, user: attributes_for(:hoge_wrong)
        expect(response).to render_template :edit
      end
    end
    context( "role_idの値が存在するなら") do
      it "データが更新される。" do
        get :edit, id: @hoge
        patch :update, id: @hoge, user: attributes_for(:hoge_new)
        @hoge.reload
        expect(@hoge.email).to eq("hoge@hosei.ac.jp")
        expect(@hoge.role_id).to eq(2)
        expect(@hoge.handle).to eq("hoge")
        expect(@hoge.password).to eq("new-password")
      end
      it "更新後に user_path にリダイレクトされる。" do
        get :edit, id: @hoge
        patch :update, id: @hoge, user: attributes_for(:hoge_new)
        expect(response).to redirect_to user_path
      end
    end
  end
end
```

spec/controllers/ users_controller_spec.rb (1/2)

```ruby
require 'rails_helper'

RSpec.describe UsersController, type: :controller do
  before(:all) do
    @admin = FactoryGirl.create(:admin)
    @staff = FactoryGirl.create(:staff)
    @guest = FactoryGirl.create(:guest)
    @hoge  = FactoryGirl.create(:hoge)
  end
  after(:all) do
    DatabaseCleaner.clean
  end

  describe "Usersのindex画面では、" do
    it "indexテンプレートで表示する。" do
      get :index
      expect(response).to render_template :index
    end
  end

  describe "Usersのupdate画面では、" do
    context( "role_idの値が存在しない場合") do
      it "データは更新されず、editの画面に戻る。" do
        get :edit, id: @hoge
        patch :update, id: @hoge, user: attributes_for(:hoge_wrong)
        expect(response).to render_template :edit
      end
    end
    context( "role_idの値が存在するなら") do
      it "データが更新される。" do
        get :edit, id: @hoge
        patch :update, id: @hoge, user: attributes_for(:hoge_new)
```

spec/controllers/ users_controller_spec.rb (2/2)

```
21  describe "Usersのupdate画面では、" do
22    context( "role_idの値が存在しない場合") do
23      it "データは更新されず、editの画面に戻る。" do
24        get :edit, id: @hoge
25        patch :update, id: @hoge, user: attributes_for(:hoge_wrong)
26        expect(response).to render_template :edit
27      end
28    end
29    context( "role_idの値が存在するなら") do
30      it "データが更新される。" do
31        get :edit, id: @hoge
32        patch :update, id: @hoge, user: attributes_for(:hoge_new)
33        @hoge.reload
34        expect(@hoge.email).to eq("hoge@hosei.ac.jp")
35        expect(@hoge.role_id).to eq(2)
36        expect(@hoge.handle).to eq("hoge")
37        expect(@hoge.password).to eq("new-password")
38      end
39      it "更新後に user_path にリダイレクトされる。" do
40        get :edit, id: @hoge
41        patch :update, id: @hoge, user: attributes_for(:hoge_new)
42        expect(response).to redirect_to user_path
43      end
44    end
45  end
46
47 end
```

Rspecの仕様の変更

ネットで検索していると、shouldや、should_notを用いた例が多くありますが、rspecの最新版では

expect().to

の書き方に変わっています。

レンダリングとリダイレクション

- レンダリングとは？　（前期の復習）
  - 画面でテンプレートが選ばれた後、コントローラの値をHTMLソースの中に埋め込むこと。
- リダイレクションとは？
  - 新しいURLに強制的に移動する。
- Render_templateと、redirect_toのテストの例を書いてみました。参考にして下さい。

Step 3: テストの一括実行

- ユニットテスト、機能テストのテスト記述が
  書きあがっていると仮定してテストを一括実行する。

  rspec spec/*/*_spec.rb spec/views/*/*_spec.rb
- と入力する。

テストの結果（Pendingの部分）

```
Pending: (Failures listed here are expected and do not affect your suite's status)

  1) RolesHelper add some examples to (or delete) /home/WebDB/workspace/vegetable-mar
ket/spec/helpers/roles_helper_spec.rb
     # Not yet implemented
     # ./spec/helpers/roles_helper_spec.rb:14

  2) UsersHelper add some examples to (or delete) /home/WebDB/workspace/vegetable-mar
ket/spec/helpers/users_helper_spec.rb
     # Not yet implemented
     # ./spec/helpers/users_helper_spec.rb:14

  3) Merchandise まだ、テストを記述していません。 /home/WebDB/workspace/vegetable-marke
t/spec/models/merchandise_spec.rb
     # Not yet implemented
     # ./spec/models/merchandise_spec.rb:4

  4) Role add some examples to (or delete) /home/WebDB/workspace/vegetable-market/spe
c/models/role_spec.rb
     # Not yet implemented
```

テストの結果（Failureの部分）

```
Failures:

  1) UsersController Usersのupdate画面では、 role_idの値が存在するなら データが更新さ
れる。
     Failure/Error: expect(@hoge.password).to eq("new-password")

       expected: "new-password"
            got: "password"

       (compared using ==)
     # ./spec/controllers/users_controller_spec.rb:37:in `block (4 levels) in <top (r
equired)>'

Finished in 0.49952 seconds (files took 3.22 seconds to load)
24 examples, 1 failure, 20 pending

Failed examples:

rspec ./spec/controllers/users_controller_spec.rb:30 # UsersController Usersのupdate
画面では、 role_idの値が存在するなら データが更新される。

[WebDB@cisnote vegetable-market]$
```

テスト結果の解釈

Users_controllerの、indexなどの部分は「成功」したために、何も表示されていません。

（試しに、render先のテンプレート名などを書き換えると、そのテストは「失敗」します。）

この作業も、大切です。テストが実行されていることが確認できます。

それ以外では、1つFailureが出るはずです。

Password更新のエラー

1) UsersController#update role_idの値が存在するなら データが更新される。

```
Failure/Error: expect(@hoge.password).to eq("new-password")

  expected: "new-password"
       got: "password"

  (compared using ==)
# ./spec/controllers/users_controller_spec.rb:37:in `block (4 levels) in
<top (required)>'
```

実は、users/editでは、userの「新規登録」と「パスワード変更」は実行できません。Deviseの管理下にあります。従って、ここではテストから「パスワード」の照合を除く必要があります。

パスワードの更新はチェックしない

改めて、deviseを操作した時に、Capybaraを用いたテストでパスワードなどのチェックを行うこととし、ここでは以下の1行を除外します。

＃でコメントアウトしています。

expect(@hoge.password).to eq("new-password")

```
    end
    context( "role_idの値が存在するなら") do
      it "データが更新される。" do
        get :edit, id: @hoge
        patch :update, id: @hoge, user: attributes_for(:hoge_new)
        @hoge.reload
        expect(@hoge.email).to eq("hoge@hosei.ac.jp")
        expect(@hoge.role_id).to eq(2)
        expect(@hoge.handle).to eq("hoge")
        # expect(@hoge.password).to eq("new-password")
      end
      it "更新後に user_path にリダイレクトされる。" do
        get :edit, id: @hoge
        patch :update, id: @hoge, user: attributes_for(:hoge_new)
        expect(response).to redirect_to user_path
      end
    end
```

テストの再実行

Failureがゼロになりました。

```
  16) users/_form add some examples to (or delete) /home/WebDB/workspace/vegetable-m
rket/spec/views/users/_form.html.erb_spec.rb
     # Not yet implemented
     # ./spec/views/users/_form.html.erb_spec.rb:4

  17) users/edit add some examples to (or delete) /home/WebDB/workspace/vegetable-ma
ket/spec/views/users/edit.html.erb_spec.rb
     # Not yet implemented
     # ./spec/views/users/edit.html.erb_spec.rb:4

  18) users/index add some examples to (or delete) /home/WebDB/workspace/vegetable-m
rket/spec/views/users/index.html.erb_spec.rb
     # Not yet implemented
     # ./spec/views/users/index.html.erb_spec.rb:4

  19) users/new add some examples to (or delete) /home/WebDB/workspace/vegetable-mar
et/spec/views/users/new.html.erb_spec.rb
     # Not yet implemented
     # ./spec/views/users/new.html.erb_spec.rb:4

  20) users/show add some examples to (or delete) /home/WebDB/workspace/vegetable-ma
ket/spec/views/users/show.html.erb_spec.rb
     # Not yet implemented
     # ./spec/views/users/show.html.erb_spec.rb:4


Finished in 0.49029 seconds (files took 4.69 seconds to load)
24 examples, 0 failures, 20 pending

[WebDB@cisnote vegetable-market]$
```

テスト駆動とは・・・

最初に、「テスト」（仕様）を書きます。

次に、そのテストを、Rspecで「テスト記述」として書いていきます。

そのテストが実行されること（実行されて、Failure）になることを確認します。あるいは、そのテスト自体にエラーがないか、修正します。

最後に、プログラムを修正（新規書き下ろし）して、プログラムの正常動作を確認します。

今日はここまで

先週行った実装の続きから始めましたので、今日は
rspecのセットアップと、動作確認までとします。

Versionが噛み合ないと、再三エラーが出ます。
Gemfileを修正し、bundle update / bundle installを何度も試す、という試行錯誤が必要になる人も、出て来る可能性があります。

Rspecは、バージョンが上がったため、「全て最新版」の組み合わせで実行する必要があります。

欠席課題

- 本日の授業を欠席した学生は、以下の画面を添付して下さい。
- 多少面倒な部分もありますが、自宅でテストを走らせて、Examples と Pendingが表示されている実行画面(スライドP66)と、その時に走らせたテストファイルの中身を、「レポートとして」報告してください。
  - 画面コピーのgifファイルをそのままむき出して提出する、ということはやめて下さい。

来週の予定

Capybaraを用いた「画面入力」のテストと、モデルのテストなどを追加していきます。

また、Contollerで、一通りすべてのメソッドのテストが実行できるように、追記していきます。