

Rails のテストの仕組み

y-yagi 著

2019-04-01 版 発行

はじめに

本書の内容

本書は、Ruby on Rails(以降 Rails) が提供しているテストの仕組みについて説明した本です。

普段 Rails を使用してアプリケーションを作られている方は、一緒にテストも書かれているかと思います。Rails は Web アプリケーションを作る為のフレームワークですが、そのアプリケーションをテストする為の仕組みも合わせて提供しています。本書では、Rails がどのような仕組みを提供しているかについて説明します。

そのため、Rails を使用したアプリケーションでのテストについての話ではなく、Rails が提供しているテストについての話です。仕組みの使い方について詳細には触れていません。ご了承ください。

なお、特に注記が無い場合は、Rails は執筆時点で最新のバージョン(6.0.0.beta3, commit: 6a5c8b9)を対象にしています。

対象読者

本書は、既に Ruby、Rails を使った事がある人を対象としています。そのため、Ruby や Rails 自体の説明や、Ruby に関連するツールについての説明は端折っています。予めご了承ください。

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってくだ

さい。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いません。また、本書に記載されいてる情報は、執筆時点でのものです。時間経過によって情報が古くなっている可能性がありますのであらかじめご了承ください。

第 1 章

Rails のテスト

本章では、Rails が提供しているテストの概要について説明します。

1.1 テストフレームワーク

Ruby でテストフレームワークといえば、test-unit、minitest、RSpec の名前が挙がることが多いと思います。Rails は、かつて test-unit を使っていて、現在は minitest を使用しています。Rails 自体のテストも、Rails を使用するユーザーに向けたテストの仕組みも、どちらも minitest を使用しています。日本で Rails アプリケーションを開発されている場合、RSpec をお使いの方が多くのではないかと思います。しかし実は RSpec は Rails がデフォルトでサポートしているテストフレームワークではなく、Rails 自体に RSpec の為の機能は全くありません。

1.2 minitest と Rails

先に述べた通り、Rails では minitest を使用しています。が、minitest をそのまま使用している訳ではありません。色々と機能拡張をおこなっています。

例えば、minitest では否定の assert を行うのに、refute というメソッド、及び、refute で始まる各種 assert 用のメソッドを使用します。例えば値が一致しない事を確認したい場合、refute_equal というメソッドを使用します。

リスト 1.2: refute_equal

```
refute_equal 5, User.count
```

しかし Rails では refute を使うことを推奨していません。代わりに、assert_not というメソッド、及び、assert_not で始まる各種 assert 用が提供されており、そちらを使用することを推奨しています。先の例だと、代わりに assert_not_equal を使用する必要があります。

リスト 1.2: refute_equal

```
assert_not_equal 5, User.count
```

元々 test-unit には assert_not があり、test-unit から minitest に移行する際に互換性の為にこれらのメソッドに追加されました。その後、refute に移行する? というような提案もあったようなのですが、それは進まず、assert_not を使う形のままで落ち着いています^{*1}。

なお、refute ではなく assert_not を使用する事をチェックする為の RuboCop の設定^{*2}もあります。Rails のリポジトリではこの cop が有効化されており、refute は一切使われないようになっています。

他にも、Rails でテストを書く際、test というメソッドを使用してテストを定義します。

リスト 1.3: test

```
test "should get index" do
  get users_url
  assert_response :success
end
```

^{*1} 当時のちゃんとした議論が見つからなかったので推測混じりなのですが、どうも refute という名前をあまり好ましく思わない人がいたため、移行は行われなかったようです。恐らくレーサー。

^{*2} <https://www.rubydoc.info/gems/rubocop/RuboCop/Cop/Rails/RefuteMethods>

これも Rails が提供しているメソッドで、minitest だけを使用している場合、このメソッドは使用する事は出来ません。

そのため、「Rails は mintiest を使用している」を正確ではなく、「Rails は minitest を拡張した独自の仕組みを使用している」が正確な表現になります。

1.3 テスト用のクラス

Rails というフレームワークは機能ごとにライブラリがわかれています。例えば O/R マッパーの Active Record、template の表示を行う Action View、メール送信の為に Action Mailer、という具合です。

Rails では、各ライブラリ毎に、そのライブラリの機能を提供する為のクラスを提供しています。クラス図は次のようになっています。

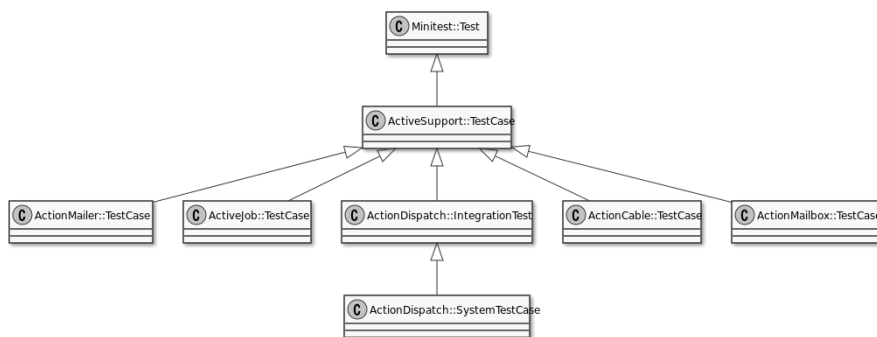


図 1.1: クラス図

紙面の都合上大分割愛していますが、minitest のテスト用のクラスである Minitest::Test を継承した ActiveSupport::TestCase というクラスがあり、各ライブラリのテストクラスはその ActiveSupport::TestCase を継承している、という点だけ覚えておいて下さい。各クラスの詳細については次章で説明します。

1.4 まとめ

本章では、Rails が提供しているテストの概要について説明しました。Rails は minitest を使用している、しかしそのまま使用しているのではなく、minitest を拡張して使用している、という点だけご理解頂けると幸いです。

第 2 章

テスト用のクラス

本章では、Rails が提供しているテスト用のクラスについて説明します。

2.1 クラスの一覧

Rails では、テスト用のクラスとして、次のようなクラスを提供しています。

表 2.1: Rails が提供しているテスト用のクラス

クラス名
ActiveSupport::TestCase
ActionMailer::TestCase
ActionView::TestCase
ActionController::TestCase
ActionDispatch::IntegrationTest
ActionDispatch::SystemTestCase
ActiveJob::TestCase
ActionCable::TestCase
ActionCable::Channel::TestCase
ActionMailbox::TestCase
Rails::Generators::TestCase

名前から何となく何の為のクラスが想像出来るのではないかと思います。基本的には各ライブラリ毎にクラスが提供されているので、そのライブラリを使用した機能のテストをしたい場合、それぞれ向けに提供されているテストを使えば良

いようになっています。

なお、Active Record や Active Model のような一部ライブラリでは専用のクラスは提供されていません。これは ActiveSupport::TestCase を使用すれば十分で、特別な処理を提供する必要が無い、と判断されている為です。専用のクラスが無い場合は、ActiveSupport::TestCase を使用するようして下さい。

2.2 ActiveSupport::TestCase

前章でも述べた通り、Rails が提供しているテストクラスのベースになっているクラスです。各テストクラス共通で使いたい機能があった場合、このクラスに機能を追加する、このクラスで module を include する、等を行うようになっています。

例えば、assert_not 等の assert_not_x で始まるメソッドは ActiveSupport::TestCase に定義されており、全てのテスト用のクラスで使用出来るようになっています。他にも、ActiveSupport::Testing::TimeHelpers という時間を処理する為の travel、travel_to 等のメソッドが定義されている module が include されています。

リスト 2.1: travel_to

```
Time.current # => Wed, 27 Mar 2019 11:56:31 JST +09:00

# 1 日前の移動
travel 1.day do
  Time.current # => Tue, 26 Mar 2019 11:56:35 JST +09:00
end

Time.current # => Wed, 27 Mar 2019 11:56:38 JST +09:00
```

また、ActiveSupport::TestCase は Rails 自体のテストでも使用されています。そのため、Rails 自体のテストをする為に必要なメソッドもこのクラスで使用出来るようになっています。

例えば、何か既存の機能を deprecate にしたい場合に、その機能が deprecate になっている事を確認する為の assert_deprecated というアサーションがあります。これは Rails 全てのライブラリで使いたいアサーションなので、

ActiveSupport::TestCase で必要な module が include されるようになっています。

リスト 2.2: assert_deprecated

```
# update_attributes メソッドを使用すると deprecate メッセージが出る
topic = Topic.find(1)
assert_deprecated do
  topic.update_attributes("title" => "The First Topic Updated")
end
```

クラスやメソッドについての詳細は、<https://edgeapi.rubyonrails.org/classes/ActiveSupport/TestCase.html> を参照してください。

2.3 ActionMailer::TestCase

メール (送信) のテストの為のクラスです。当然のことですが、テストで実際にメールを送信するわけにはいきません。ActionMailer::TestCase では、メール送信処理が実行されても実際のメールの送信は行わず、代わりに送信処理が実行されたメールを配列で管理するようにしています。合わせて、送信処理が呼ばれた (または呼ばれていない) 事を確認する為のアサーションを提供しています。

リスト 2.3: assert_emails

```
test "invite friend" do
  # invite_friend_url に POST したら招待用のメールが送信される
  assert_emails 1 do
    post invite_friend_url, params: { email: 'friend@example.com' }
  end
end
```

assert_emails は同期処理 (@code{deliver_now})、非同期処理 (@code{deliver_later}) どちらで送信されたメールもチェックの対象になります。非同期で送信されたメールだけチェックしたい場合は、assert_enqueued_emails を使用する必要があります。

アサーションについての詳細は、<https://edgeapi.rubyonrails.org/>

[classes/ActionMailer/TestHelper.html](https://edgeapi.rubyonrails.org/classes/ActionMailer/TestHelper.html) を参照してください。

2.4 ActiveRecord::TestCase

ジョブのテストのためのクラスです。メールと事なり、ジョブはテストで実際に実行しても問題無い事が多いでしょう。しかし例えば、「1 時間後に実行されるジョブ」があった場合、テストで実際に 1 時間待つ訳にはいきません。ActiveRecord::TestCase では、ジョブの登録処理が行われたらそのジョブを内部で保持し、どのようなジョブが登録されたかを確認出来るようにしています。当然、その登録された内容を確認する為のアサーションも提供されています。

リスト 2.4: assert_enqueued_jobs

```
test "withdrawal" do
  user = User.last
  # ユーザが退会したら LoggingJob が登録される
  assert_enqueued_with(job: LoggingJob) do
    user.withdrawal
  end
end
```

どのような引数が指定されたかチェックしたり、登録されたジョブを実行したりする事も出来るようになっています。アサーションについての詳細は、<https://edgeapi.rubyonrails.org/classes/ActiveJob/TestHelper.html> を参照してください。

2.5 ActionController::TestCase

名前から Action View のテンプレートに関するテスト用のクラスかと推測されるかと思うのですが、実際はちょっと異なり view の helper のテスト用のクラスです。ActionController::TestCase では helper を使用する為に必要な controller や view の生成処理を行ってくれます。

リスト 2.5: ActionController::TestCase

```
module UsersHelper
  def link_to_user(user)
    link_to "#{user.first_name} #{user.last_name}", user
  end
end

class UsersHelperTest < ActionView::TestCase
  test "link_to_user returns link with user name" do
    user = User.find_by(first_name: "Yamada", last_name: "Taro")
    assert_dom_equal %<a href="/user/#{user.id}">Yamada Taro</a>,
      link_to_user(user)
  end
end
```

しかし、Rails 4.2 より helper のテストはそもそも生成されなくなり^{*1}、このクラスが使用される事は無くなりました。また、helper 単体でのテストはあまり意味が無い^{*2}のでは、という声もあり、このクラスを使用する事は基本的に無いかと思います。

2.6 ActionController::TestCase

コントローラーのテストのためのクラスです。特定のコントローラーのメソッドに対して、HTTP リクエストの送信及びレスポンスの確認ができるようになっています。

リスト 2.6: ActionController::TestCase

```
test "should get index" do
  get :index
  assert_response :success
end

test "should create user" do
  assert_difference('User.count') do
    post(:create, params: { user: { email: @user.email, name: @user.name } })
  end
end
```

^{*1} <https://github.com/rails/rails/commit/a34b6649d061977026db7124d834faccdf5bd8ef>

^{*2} Integration テストや System テストビューのテストと合わせてやるのが良いのでは、という意見が多いです。

```
end

assert_redirected_to user_url(User.last)
end
```

しかし同様にコントローラーのテストを行う為のクラスとして `ActionDispatch::IntegrationTest` があります。`ActionDispatch::IntegrationTest` だとルーティングもセットでテストが出来る^{*3}、HTTP リクエストがより実際のリクエストに近い形で送信される等のメリットがあるのですが、実行は `ActionController::TestCase` の方が高速だった為、コントローラーのテストには長らく `ActionController::TestCase` が使われるようになっていました。

しかし、Rails 5.0 で `ActionDispatch::IntegrationTest` のパフォーマンスが大幅に改善され、実行速度の差は大幅に縮まりました。結果、コントローラーのテストでも `ActionDispatch::IntegrationTest` が使用される事が推奨されるようになり、`scaffold` で生成するコントローラーのテストでも `ActionDispatch::IntegrationTest` が使用されるようになりました。

なお、その際に `ActionController::TestCase` は `gem` に切り出して Rails 本体から削除する、という話があったのですが、何だかんだまだコードは残ったままになっています。とはいえ、機能追加等が行われる事は(恐らく)無いので、新規に追加するテストについては `ActionDispatch::IntegrationTest` を使用する事をおすすめします。

^{*3} `ActionController::TestCase` は送信先にコントローラーのアクション名を指定する為、ルーティングのテストは出来なかったのです。

2.7 ActionDispatch::IntegrationTest

2.8 ActionDispatch::SystemTestCase

2.9 ActionCable::TestCase

2.10 ActionCable::Channel::TestCase

2.11 ActionMailbox::TestCase

2.12 Rails::Generators::TestCase

Rails のテストの仕組み

2019 年 4 月 1 日 初版第 1 刷 発行

著 者 y-yagi
