django.test.TestCase

Django 公式ドキュメント テストを書いて実行する Django 公式ドキュメント テストツール

Django 公式のユニットテストフレームワークとして、 `django.test.TestCase` があります。

`django.test.TestCase` は、 `unittest.TestCase` を継承しています。
そして、 `unittest.TestCase` に、ウェブフレームワークのテストに便利な機能が追加されています。

本資料は、読者が `unittest.TestCase` を使ったことがあることを前提にしています。

> pytest を使って Django プロジェクトのテストコードを書くこともできますが、そのための準備はやや複雑です。
> pytest を使って Django のテストコードを書くのに必要な手続きを習得するよりも unittest を習得してしまうほうが、学習コストが低く、また応用が効くノウハウになるでしょう。

テストランナーの機能

データベースの作成/マイグレーション/削除

Django のテストランナーは、起動時に、データベースの作成とマイグレーションを行います。

Django のテストランナーは、ひとつひとつのテストが終了する都度、テスト用データベースをマイグレーション直後の状態に戻します(ロールバックします)。
これにより、個々のテストは、中身がクリアされた(マイグレーション直後と同様の)状態のデータベースを使えます。
ですので、他のテストの結果に影響を受けない状態でテストを行えます。

そして、Django のテストランナーは、テストが終了すると、データベースを削除します。

> Django のユニットテストでは、データベースのロールバックは自動的に行われます。
> ですので、 `tearDown` メソッドや `tearDownClass` メソッド内でデータベースの初期化作業をする必要はありません。

テストデータベースの作成と破棄、所在と権限

テストデータベースは以下のように作成され、そして破棄されます。

SQLite を使っているときは、デフォルトでは、テストにはインメモリのデータベースを使います。
つまり、データベースはメモリ内に作成されるため、ファイルシステムへのアクセスを完全になくすことができます。

PostgreSQL, MySQL 等のデータベースを使用している場合は、テスト用のデータベースがテストの都度作られます。
そして、テスト終了時にこのテスト用のデータベースは破棄されます(*1)

このテストデータベースの名前は、 DATABASES 設定内の `NAME` の値の前に `test_` を付けたものになります。(*2)

> (*1) `python manage.py test` コマンドに `--keepdb` オプションをつけて実行すると、テスト用データベースは破棄されません。
> なので、次回以降のテストをより高速に実行することができます。
>
> (*2) テストデータベースの名前の変更は、 `settings'` の DATABASES で TEST セクションの NAME` キーを書き換えることで可能です。
>
> Django 公式ドキュメント テストを書いて実行する test データベース

Django がデータベースへの接続で利用するユーザがテスト用データベースに対して作成/削除等の権限を有していないとテストが失敗することがあるので注意してください。
この権限の付与手順は、概ね以下のとおりです。

1. テスト用データベースと同名のデータベースをいったん作成する(マイグレーション等は不要)
2. Django が使うデータベースユーザに、 1. で作ったデータベースの適切な権限を付与する

以下は、 PostgreSQL の場合のより詳細な手順例です。

1. Django の設定から、以下を調べる
    - Django が使っているデータベースのデータベース名
    - Django が使っているデータベースユーザ名
2. PostgreSQL に postgres ユーザでログインする
3. Django のデータベース名に test_ という prefix のついた名前のデータベースを作る
4. 3.で作ったデータベースに対する権限を、 Django の設定ファイルに記載のユーザに対して付与する
5. 作成したデータベースを削除する
6. PostgreSQL からログアウトする

1. Django の設定から、Django が使うデータベース名と postgres ユーザのユーザ名を調べる

config.settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'django_mysite_db<',
        'USER': 'django_user',
        'PASSWORD': 'mysiteUserPass123',
        'HOST': 'localhost',
        'PORT': '5432',
        'ATOMIC_REQUESTS': True,
    }
}
```

上記から、以下のことが分かりました。

項目	値
Django が使っているデータベースユーザ名	`django_user`
Django が使っているデータベースのデータベース名	`django_mysite_db`
Django がテストで使うデータベースのデータベース名	`test_django_mysite_db`

2. PostgreSQL に postgres ユーザでログインする

```
sudo -u postgres psql
```

3. Django のデータベース名に test_ という prefix のついた名前のデータベースを作る

```
CREATE DATABASE test_django_mysite_db;
```

4. 3.で作ったデータベースに対する権限を、 Django の設定ファイルに記載のユーザに対して付与する

```
GRANT ALL PRIVILEGES ON DATABASE test_django_mysite_db TO django_user;
```

5. 作成したデータベースを削除する

```
DROP DATABASE test_django_mysite_db;
```

6. PostgreSQL からログアウトする

```
\q
```

一度データベースを作成し、適切な権限を付与します。
そうすれば、そのデータベースを削除して以降に Django が同名のデータベースを作ったとしても、もうエラーは発生しません。

django.test.TestCase.client の機能

テストクライアント

テストメソッド内では、テスト用の HTTP テストクライアントを利用できます。

このテストクライアントでは、以下の要領で、 `GET` や `POST` などの HTTP メソッドを利用できます。
リクエストの戻り値としてレスポンスを得られます。
また、 View クラスや view 関数がテンプレートに渡した `context` 辞書の値など、様々な関連情報も得られます。

このレスポンスの内容を検証することで、 View の挙動をテストできます。

```
from django.test import TestCase


class TestMethodSample(TestCase):
    def test_get_method_sample(self):
        response = self.client.get('/sample/')
        self.assertEqual(response.status_code, 200)

        messages = list(response.context['messages'])
        self.assertEqual(len(messages), 1)
        self.assertEqual(str(messages[0]), 'ページの表示に成功しました。')

    def test_post_method_sample(self):
        response = self.client.post('/sample/', {'foo': 'bar'})
        self.assertEqual(response.status_code, 302)
```

Django 公式ドキュメント テストツール テストクライアント

以下のメソッドによって、特定のテストユーザとしてログインすることも可能です。
ログインに成功すれば、ログインユーザとしてウェブページで `GET` や `POST` を行ったときの View の挙動をテストできます。

- `django.test.client.login`
- `django.test.client.force_login`

以下では、未ログイン状態のユーザ、ログイン済のユーザでの挙動の違いをテストする例を示します。

```
from django.contrib.auth import get_user_model
from django.shortcuts import resolve_url
from django.test import TestCase

User = get_user_model()


class TestArticleListViewSample(TestCase):
    """ 未ログインユーザ、ログインユーザでの GET リクエストのテストの例 """

    def test_anonymous(self):
        """ 未ログインユーザとしてGETリクエストを実施 """
        path = resolve_url('log:article_list')
        response = self.client.get(path)

        self.assertEqual(response.status_code, 200)

    def test_authed_user(self):
        """ ログインユーザとしてGETリクエストを実施 """
        user = User.objects.create_user(username='testuser', email='foo@bar.com',
                                        password='testpassword')
        login_result = self.client.login(email=user.email, password='testpassword')
        self.assertTrue(login_result)

        path = resolve_url('log:article_list')
        response = self.client.get(path)

        self.assertEqual(response.status_code, 200)
```

アサーション

Django 公式ドキュメント テストツール Assertions

`django.test.TestCase` には、 `unittest.TestCase` クラスから継承したもの以外にもウェブフレームワークらしい様々な assert メソッドがあります。
まずは、レスポンス解析用の assert メソッドに慣れていきましょう。

メソッド名	概要	例
assertContains	レスポンスが指定された要素を含んでいることを確認する	self.assertContains(response, "Hello World")
assertNotContains	レスポンスが指定された要素を含んでいないことを確認する	self.assertNotContains(response, "Error")
assertHTMLEqual	2つのHTMLテキストが同じ意味合いになることを確認する	python self.assertHTMLEqual(html1, html2)
assertHTMLNotEqual	2つのHTMLテキストが同じ意味合いにならないことを確認する	python self.assertHTMLNotEqual(html1, html2)
assertTemplateUsed	特定のテンプレートが使用されたことを確認する	self.assertTemplateUsed(response, "template.html")
assertTemplateNotUsed	特定のテンプレートが使用されなかったことを確認する	self.assertTemplateNotUsed(response, "template.html")
assertRedirects	レスポンスが指定されたURLにリダイレクトされることを確認する	self.assertRedirects(response, "/redirected/")

`assertHTMLEqual`, `assertHTMLNotEqual` での `html1`, `html2` は、HTML テキストを文字列で指定します。
Django 公式ドキュメント テストツール assertHTMLEqual

https://github.com/k-brahma/django_photo_diary/blob/main/log/tests/test_views_sample_basic.py

```python
from django.test import TestCase


class TestHTMLEqualSample(TestCase):
    def test_html_equal_1(self):
        # 以下の html1 と html2 は HTMLタグの機能や意味においては等価です。
        html1 = '<p class="my-class" id="my-id">本文</p>'
        html2 = '<P id="my-id" class="my-class" >本文</P>'

        self.assertHTMLEqual(html1, html2)

    def test_html_equal_2(self):
        # 以下の html3 と html4 は HTMLタグの機能や意味においては等価です。
        html1 = '<input type="checkbox" class="my-class" id="my-id" checked>'
        html2 = '<INPUT TYPE="checkbox"  checked="checked" id="my-id" class="my-class">'

        self.assertHTMLEqual(html1, html2)
```

Django の View クラスのテストコードの例を以下に示します。

https://github.com/k-brahma/django_photo_diary/blob/main/log/tests/test_views_sample.py

```python
from django.shortcuts import resolve_url
from django.test import TestCase
from django.contrib.auth import get_user_model

from log.models import Article, Tag

User = get_user_model()


class TestArticleUpdateViewSample(TestCase):
    """
    ArticleUpdateView のテスト

    ページの表示/日記の更新ができるのが投稿者本人または is_staff ユーザのみということを確認する
    """

    @classmethod
    def setUpTestData(cls):
        cls.list_path = resolve_url('log:article_list')

        # テストユーザを作る。この作業は全テストを通じて一度で良いので setUpTestData で行う
        cls.user = User.objects.create_user(username='test', email='foo@bar.com',
                                            password='testpassword')

        # テスト用のタグを作成
        for i in range(1, 5):
            tag = Tag.objects.create(name=f'test_tag{i}', slug=f'test_tag{i}')
            setattr(cls, f'tag{i}', tag)

    def setUp(self):
        # テストの都度改めて article を作成する
        # 投稿編集テストがあるので、 setUpTestData で行うのは不適切
        # (更新された article のままだと他のテストに影響するため)
        self.article = Article.objects.create(title='base_test_title',
                                              body='base_test_body', user=self.user)
        self.article.tags.add(self.tag1, self.tag2)

        # aritcle の pk は生成される都度異なる場合があるので注意(データベース製品による)
        self.path = resolve_url('log:article_update', pk=self.article.pk)

    def result_redirect(self, response):
        """ 権限を持たないユーザが GET/POST でアクセスしたときの処理の検証用メソッド """
        redirect_path = self.list_path
        self.assertRedirects(response, redirect_path)

        messages = list(response.context['messages'])
        self.assertEqual(len(messages), 1)
        self.assertEqual(str(messages[0]), '日記を更新できるのは投稿者と管理者だけです。')

    def result_get_success(self, response):
        """ 権限を持つユーザが GET でアクセスしたときの処理の検証用メソッド """
        self.assertEqual(response.status_code, 200)
```

```
        self.assertTemplateUsed(response, 'log/article_update.html')
        self.assertContains(response, self.article.title)

    def result_post_success(self, response):
        """ 権限を持つユーザが POST で日記の更新を行ったときの処理の検証用メソッド """
        self.assertRedirects(response, self.list_path)

        messages = list(response.context['messages'])
        self.assertEqual(len(messages), 1)
        self.assertEqual(str(messages[0]), '日記を更新しました。')

    def check_article(self, title, body, tags):
        """ post 後の article オブジェクトの状態チェック用メソッド """
        articles = Article.objects.all()
        self.assertEqual(len(articles), 1)
        self.assertEqual(articles[0].title, title)
        self.assertEqual(articles[0].body, body)

        article_tags = articles[0].tags.all()
        self.assertEqual(len(article_tags), 2)
        self.assertIn(tags[0], article_tags)
        self.assertIn(tags[1], article_tags)

    def test_get_anonymous(self):
        """ AnonymousUser は一覧ページにリダイレクトされる """
        response = self.client.get(self.path, follow=True)
        self.result_redirect(response)

    def test_get_another_user(self):
        """ 投稿者本人でなくてスタッフでもない場合は一覧ページにリダイレクトされる """
        another_user = User.objects.create_user(username='another', email='foo2@bar.com',
                                                password='testpassword')
        result = self.client.login(email=another_user.email, password='testpassword')
        self.assertTrue(result)  # ログイン成功しているか確認

        response = self.client.get(self.path, follow=True)
        self.result_redirect(response)

    def test_get_article_user(self):
        """ 投稿者本人の場合は更新ページが表示される """
        result = self.client.login(email=self.user.email, password='testpassword')
        self.assertTrue(result)  # ログイン成功しているか確認

        response = self.client.get(self.path)
        self.result_get_success(response)

    def test_get_is_staff(self):
        """ 投稿者本人でなくてもスタッフの場合は更新ページが表示される """
        another_user = User.objects.create_user(username='another', email='foo2@bar.com',
                                                password='testpassword', is_staff=True)
        result = self.client.login(email=another_user.email, password='testpassword')
        self.assertTrue(result)  # ログイン成功しているか確認
```

```
        response = self.client.get(self.path)
        self.result_get_success(response)

    def test_post_failure_another_user(self):
        """ 投稿者本人でなくてスタッフでもない場合は投稿に失敗し一覧ページにリダイレクトされる """
        another_user = User.objects.create_user(username='another', email='foo2@bar.com',
                                                password='testpassword')
        result = self.client.login(email=another_user.email, password='testpassword')
        self.assertTrue(result)  # ログイン成功しているか確認

        # postメソッドで送信するデータを生成
        data = {
            'title': 'test_title_fail',
            'body': 'test_body_fail',
            'tags': [self.tag3.id, self.tag4.id, ]
        }
        response = self.client.post(self.path, data=data, follow=True)
        self.result_redirect(response)

        # データが更新されていないことを確認
        self.check_article('base_test_title', 'base_test_body', [self.tag1, self.tag2])

    def test_post_success_article_user(self):
        """ 投稿者本人の場合は投稿を更新できる """
        self.client.force_login(self.user)  # ログイン状態にする

        # postメソッドで送信するデータを生成
        data = {
            'title': 'test_title1',
            'body': 'test_body1',
            'tags': [self.tag3.id, self.tag4.id, ]
        }
        response = self.client.post(self.path, data=data, follow=True)

        self.result_post_success(response)

        # データが更新されていることを確認
        self.check_article('test_title1', 'test_body1', [self.tag3, self.tag4])

    def test_post_success_is_staff(self):
        """ 投稿者本人でなくてもスタッフの場合は投稿を更新できる """
        another_user = User.objects.create_user(username='another', email='foo2@bar.com',
                                                password='testpassword', is_staff=True)
        result = self.client.login(email=another_user.email, password='testpassword')
        self.assertTrue(result)  # ログイン成功しているか確認

        # postメソッドで送信するデータを生成
        data = {
            'title': 'test_title2',
            'body': 'test_body2',
            'tags': [self.tag3.id, self.tag4.id, ]
```

```
        }
        response = self.client.post(self.path, data=data, follow=True)

        self.result_post_success(response)

        # データが更新されていることを確認
        self.check_article('test_title2', 'test_body2', [self.tag3, self.tag4])
```

コマンドラインからの実行

Django 公式ドキュメント テストを書いて実行する テストの実行
Django の unittest を実行するには、以下のコマンドを実行します。

```
python manage.py test
```

テスト対象を絞りこんで実行することもできます。
Django のユニットテストは、 `discover` オプションを指定しないでもパッケージ以下のすべてのテストを再帰的に探索してテストスイートを作ります。その点、 unittest よりも簡単です。

```
# log アプリのテストだけを実行
python manage.py test log

# accounts アプリと log アプリののテストだけを実行
python manage.py test accounts log

# log アプリの tests パッケージ以下にあるテストだけを実行
python manage.py test log.tests

# 1つのテストモジュールだけを実行
python manage.py test log.tests.test_views

# 1つのテストクラスだけを実行
python manage.py test log.tests.test_views.TestArticleUpdateView

# 1つのテストメソッドだけを実行
python manage.py test log.tests.test_views.TestArticleUpdateView.test_get_anonymous
```

テストレポート

テストの実行結果は、以下の例のように表示されます。

```
(venv) PS D:\django_project> python manage.py test
Found 62 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..............................................................
----------------------------------------------------------------------
Ran 62 tests in 5.327s

OK
Destroying test database for alias 'default'...
```

以下は、失敗したテストがあった場合の表示例です。

```
(venv) PS D:\django_project> python manage.py test
Found 62 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....................................................FF..F..
======================================================================
FAIL: test_get_anonymous
(log.tests.test_views_sample.TestArticleUpdateViewSample.test_get_anonymous)
AnonymousUser は一覧ページにリダイレクトされる
----------------------------------------------------------------------
Traceback (most recent call last):
  File "D:\django_project\log\tests\test_views_sample.py", line 84, in test_get_anonymous
    self.result_redirect(response)
  File "D:\django_project\log\tests\test_views_sample.py", line 52, in result_redirect
    self.assertEqual(str(messages[0]), '日記を更新できるのは投稿者だけです。')
AssertionError: '日記を更新できるのは投稿者と管理者だけです。' != '日記を更新できるのは投稿者だけ
です。'
- 日記を更新できるのは投稿者と管理者だけです。
?              ----
+ 日記を更新できるのは投稿者だけです。


======================================================================
FAIL: test_get_another_user
(log.tests.test_views_sample.TestArticleUpdateViewSample.test_get_another_user)
投稿者本人でなくてスタッフでもない場合は一覧ページにリダイレクトされる
----------------------------------------------------------------------
Traceback (most recent call last):
  File "D:\django_project\log\tests\test_views_sample.py", line 94, in
test_get_another_user
    self.result_redirect(response)
  File "D:\django_project\log\tests\test_views_sample.py", line 52, in result_redirect
    self.assertEqual(str(messages[0]), '日記を更新できるのは投稿者だけです。')
AssertionError: '日記を更新できるのは投稿者と管理者だけです。' != '日記を更新できるのは投稿者だけ
です。'
- 日記を更新できるのは投稿者と管理者だけです。
?              ----
+ 日記を更新できるのは投稿者だけです。


======================================================================
FAIL: test_post_failure_another_user
(log.tests.test_views_sample.TestArticleUpdateViewSample.test_post_failure_another_user)
投稿者本人でなくてスタッフでもない場合は投稿に失敗し一覧ページにリダイレクトされる
----------------------------------------------------------------------
Traceback (most recent call last):
  File "D:\django_project\log\tests\test_views_sample.py", line 128, in
test_post_failure_another_user
    self.result_redirect(response)
  File "D:\django_project\log\tests\test_views_sample.py", line 52, in result_redirect
    self.assertEqual(str(messages[0]), '日記を更新できるのは投稿者だけです。')
AssertionError: '日記を更新できるのは投稿者と管理者だけです。' != '日記を更新できるのは投稿者だけ
```

```
です。'
- 日記を更新できるのは投稿者と管理者だけです。
?                ----
+ 日記を更新できるのは投稿者だけです。


----------------------------------------------------------------------
Ran 62 tests in 5.363s

FAILED (failures=3)
Destroying test database for alias 'default'...
```

そのほかの注意点

そのほか、Django のユニットテストでは、以下のような注意点があります。

1. リダイレクトのテストで使う2つのメソッドはそれぞれ独自にリダイレクト先へのリクエストを行う
2. `manage.py startapp <アプリ名>` で作成される `tests.py` はすぐに削除する

1. リダイレクトの検証方法

リダイレクトのテストには、慣れないと気づきにくいハマりポイントがあります。
なので、このタイミングでポイントをお伝えしておきます。

リダイレクトのテストに関係するのは、 `django.tests.TestCase` クラスの、以下の2つのインスタンス変数とメソッドです。

使い分けも含めてここで説明します。

属性	引数	初期値	説明
`client`	`follow=True`	`False`	テストクライアントでのリクエスト時に使うオブジェクト リダイレクト先で起きるもろもろを調べたいときに使う。
`assertRedirects`	`fetch_redirect_response=True`	`True`	リダイレクトレスポンスのテストに使うメソッド。 リダイレクト先のでの `GET` リクエストのステータスコードを調べたいときに使う。

まず、 `client` オブジェクトについて説明します。

`client` オブジェクトは、テストクライアントでのリクエスト時に使うオブジェクトです。
`get` や `get` などのリクエストメソッド実行時に、キーワード `follow` を指定できます。

`follow=True` を指定すると、リダイレクト先のページのレスポンスを取得します。
`follow=False` を指定すると、リダイレクト先のページのレスポンスを取得しません。
デフォルト値は `False` です。

https://github.com/k-brahma/django_photo_diary/blob/main/log/tests/test_views_sample_basic.py

```python
from django.contrib.auth import get_user_model
from django.shortcuts import resolve_url
from django.test import TestCase
from log.models import Article


class TestRedirectClient(TestCase):
    """
    client.get でのリダイレクトのテストの例
    follow=True/False での挙動の違いを確認する
    """

    def setUp(self):
        """ すべてのテストメソッドに共通の準備 """
        user = get_user_model().objects.create_user(
            username='test', email='foo@bar.com', password='testpassword')
        article = Article.objects.create(
            title='base_test_title', body='base_test_body', user=user)

        self.redirect_path = resolve_url('log:article_list')
        self.path = resolve_url('log:article_update', pk=article.pk)

    def test_follow_false(self):
        """ follow=False のとき、リダイレクト先のページを取得しない """
        response = self.client.get(self.path, follow=False)

        # リダイレクトコードを受け取るところまでしか処理を進めないので、ステータスコードは 302
        self.assertEqual(response.status_code, 302)

        # リダイレクト先ページのHTMLを取得しない
        html = response.content.decode('utf-8')
        self.assertEqual(html, "")

        # リダイレクト先ページの context を取得しない
        self.assertIsNone(response.context)

    def test_follow_true(self):
        """ follow=True  のとき、リダイレクト先のページを取得する """
        response = self.client.get(self.path, follow=True)

        # リダイレクト先ページを取得するので、ステータスコードは 200
        self.assertEqual(response.status_code, 200)

        # リダイレクト先ページのHTMLを取得するので、空のコンテンツではない
        html = response.content.decode('utf-8')
        self.assertNotEqual(html, "")

        # リダイレクト先ページの context を取得するので、Noneではない
        self.assertIsNotNone(response.context)

    def test_default(self):
        """ follow のデフォルト値は False """
```

```
        response = self.client.get(self.path, follow=False)

        self.assertEqual(response.status_code, 302)

        html = response.content.decode('utf-8')
        self.assertEqual(html, "")

        self.assertIsNone(response.context)
```

次に、 `assertRedirects` メソッドについて説明します。

`assertRedirects` メソッドは、リダイレクトレスポンスのテストに使うメソッドです。
リダイレクト先の情報のうち、以下の2つの項目についてはこのメソッドで検査することができます。
(逆に言うと、以下の2つ以外の項目については検査できません)

- リダイレクト先のパス
- リダイレクト先ページに `GET` リクエストを行ったときのレスポンスのステータスコード

ただし、 `fetch_redirect_response` 引数の値が `False` のときは、リダイレクト先のパスしか検査できません。
リダイレクト先ページのステータスコードも調べるには、 `fetch_redirect_response` 引数の値を `True` にする必要があります。

> もっとも、以下にあるとおり、 `fetch_redirect_response` 引数のデフォルト値は `True` です。> ですので、明示的に `fetch_redirect_response=True` と指定する必要はありません。

`assertRedirects` メソッドの引数

引数名	初期値	概要	例
`response`		テスト中のHTTPリクエストへのレスポンスオブジェクト	`response = self.client.get('/my-url/')`
`expected_url`		期待されるリダイレクト先のURLを表す文字列	`'/login/'`
`status_code`	`302`	期待されるリダイレクトのHTTPステータスコード	`status_code=301`
`target_status_code`	`200`	リダイレクト先URLへの `GET` リクエストの期待されるHTTPステータスコード	`target_status_code=200`
`msg_prefix`	`""`	アサーションエラーメッセージの接頭辞として使用する文字列	`msg_prefix='Assertion Failed:'`
`fetch_redirect_response`	`True`	リダイレクト先のレスポンスを取得するかどうかを制御するフラグ	`fetch_redirect_response=False`

`assertRedirects` メソッドの使用例:

`status_code` , `target_status_code` , `fetch_redirect_response` 引数の初期値を念頭に入れたうえで、以下のコードを読んでみてください。

```
response = self.client.get('/my-url/')

# 以下はいずれも同じ
self.assertRedirects(response, "/login/")
self.assertRedirects(response, "/login/", status_code=302)
self.assertRedirects(response, "/login/", status_code=302, target_status_code=200)
self.assertRedirects(response, "/login/", status_code=302, target_status_code=200,
fetch_redirect_response=True)
self.assertRedirects(response, "/login/", 302, 200, True)

# 以下はいずれも同じ
self.assertRedirects(response, "/login/", fetch_redirect_response=False)
self.assertRedirects(response, "/login/", status_code=302, fetch_redirect_response=False)
```

`assertRedirects` メソッドは、 `fetch_redirect_response=True` を指定すると、以下の動作をします。

1. 第一引数 `response` の内容を元にしてリダイレクト先のパスを取得する
2. 1.で取得したリダイレクト先に `GET` リクエストを送信する
3. 2.で取得したレスポンスのステータスコードを調べ、 `target_status_code` で指定された値と比較する

`assertRedirects` メソッドは、 `client` がメソッド実行時に `follow=True` を指定しているかどうかに関わりなく、リダイレクト先からのレスポンスを取得できます。
なぜなら、 `assertRedirects` メソッドは、内部で自分でリダイレクト先のパスに対するリクエストを送信しているからです。

ただし、すでに述べたとおり、リダイレクト先の情報のうち、このメソッドで検査することができるのは以下の2つの項目だけです。

- リダイレクト先のパス
- リダイレクト先ページに `GET` リクエストを行ったときのレスポンスのステータスコード

ですので、レスポンスに含まれる HTML やコンテキストの内容等を検査したいというときには、 `client.get(path, follow=True)` とし、レスポンスを `assert` メソッドで検証することになります。

https://github.com/k-brahma/django_photo_diary/blob/main/log/tests/test_views_sample_basic.py

```
    class TestRedirectAssertRedirect(TestCase):
        """
        assertRedirects() の挙動をテスト
        fetch_redirect_response=True/False での挙動の違いを確認する
        """

        def setUp(self):
            """ すべてのテストメソッドに共通の準備 """
            user = get_user_model().objects.create_user(
                username='test', email='foo@bar.com', password='testpassword')
            article = Article.objects.create(
                title='base_test_title', body='base_test_body', user=user)

            self.redirect_path = resolve_url('log:article_list')
            self.path = resolve_url('log:article_update', pk=article.pk)

        def test_fetch_redirect_response_false(self):
            """ fetch_redirect=False のとき、assertRedirects メソッドは、
                リダイレクト先の検証を行わない """
            response = self.client.get(self.path)

            self.assertRedirects(
                response, self.redirect_path,
                target_status_code=500,  # リダイレクト先の検証を行わないのでこの値は無視される
                fetch_redirect_response=False)

        def test_fetch_redirect_response_true(self):
            """ fetch_redirect=True のとき、assertRedirects メソッドは、
                内部でリダイレクト先へのリクエストを発行し、ステータスコードの検証を行う """
            response = self.client.get(self.path)

            # target_status_code は 正しい値なのでOK
            self.assertRedirects(response, self.redirect_path, target_status_code=200,
                                 fetch_redirect_response=True)

            # target_status_code は 正しい値ではないのでこれはNG
            # self.assertRedirects(response, self.redirect_path,
            #                      target_status_code=500,
            #                      fetch_redirect_response=True)

        def test_fetch_redirect_response_default(self):
            """ fetch_redirect のデフォルト値は True """
            response = self.client.get(self.path)

            # target_status_code は 正しい値なのでOK
            self.assertRedirects(response, self.redirect_path, target_status_code=200, )

            # target_status_codeの初期値は 200 なので以下でもOK
            self.assertRedirects(response, self.redirect_path)

            # 以下は、そのほかの初期値もあえて指定したもの
            self.assertRedirects(response, self.redirect_path,
```

```
                        status_code=302,
                        target_status_code=200)
        self.assertRedirects(response, self.redirect_path,
                        status_code=302,
                        target_status_code=200,
                        fetch_redirect_response=True)

        # target_status_code は 正しい値ではないのでこれはNG
        # self.assertRedirects(response, self.redirect_path, target_status_code=500, )
```

リダイレクトの検査についての話を別の角度からまとめます。

検査したい項目	使うオブジェクト	引数についての注意
リダイレクト先のパス	`assertRedirects`	`expected_url` 引数で検証する (必須)
リダイレクトレスポンスの ステータスコード	`assertRedirects`	`status_code` 引数で検証する (デフォルトは 302)
リダイレクト先でのレスポンスの ステータスコード	`assertRedirects`	`target_status_code` 引数で検証する (デフォルトは 200) `fetch_redirect_response` は `True` が必要 (デフォルトは `True`)
上記以外	`client`	`follow` は `True` にする

最後に、様々な検査項目を想定したサンプルコードを紹介します。

```
    class TestRedirectVariousCases(TestCase):
        """ 種々のニーズに対応したリダイレクトのテスト方法まとめ """

        def setUp(self):
            """ すべてのテストメソッドに共通の準備 """
            user = get_user_model().objects.create_user(
                username='test', email='foo@bar.com', password='testpassword')
            article = Article.objects.create(
                title='base_test_title', body='base_test_body', user=user)

            self.redirect_path = resolve_url('log:article_list')
            self.path = resolve_url('log:article_update', pk=article.pk)

        def test_assert_status_code_only(self):
            """ リダイレクト元でのスタータスコードを検査したいだけのとき"""
            response = self.client.get(self.path)
            self.assertEqual(response.status_code, 302)

        def test_assert_redirect_status_code(self):
            """ リダイレクト先のステータスコードを検査したいだけのとき """
            response = self.client.get(self.path, follow=True)
            self.assertEqual(response.status_code, 200)

        def test_assert_redirect_page_content(self):
            """ リダイレクト先のコンテンツを検査したいいとき """
            response = self.client.get(self.path, follow=True)

            html = response.content.decode('utf-8')
            self.assertNotEqual(html, "")

            self.assertContains(response, '記事一覧')

            messages = list(response.context['messages'])
            self.assertEqual(len(messages), 1)
            self.assertEqual(str(messages[0]), '日記を更新できるのは投稿者と管理者だけです。')

        def test_assert_redirect(self):
            """ リダイレクト元とリダイレクト先のスタータスコードの両方を検査したいとき """
            response = self.client.get(self.path, follow=True)
            self.assertRedirects(response, self.redirect_path,
                                 status_code=302, target_status_code=200)

            # status_code=302, target_status_code=200 はともに初期値なので省略可能
            self.assertRedirects(response, self.redirect_path)
```

ところで、 `assertRedirects` メソッドのキーワード引数 `fetch_redirect_response` のデフォルト値は `True` ですが、 `False` は、どのようなときに指定するのでしょうか。
このオプションは、リダイレクト先のパスが他サイトの `URL` になってしまう等、リダイレクト先のコンテンツを取得することができないときに指定します。
(たとえば、 `https://gogole.com/` にリダイレクトするといった場合です)

通常はデフォルトの `True` のままで問題ありません。

2. manage.py startapp <アプリ名> で作成される tests.py はすぐに削除する

Django では、 `manage.py startapp <アプリ名>` コマンドでアプリケーションを作成すると、自動的に `tests.py` ファイルが作成されます。

ところで、この `tests.py` を放置したまま同じディレクトリに `tests` パッケージを作ると、どうなるでしょうか。
実は、 `tests.py` と `tests` パッケージが同一ディレクトリにあると、 `python manage.py test` コマンド実行時に `ImportError` エラーが発生してテストそのものが実行できなくなることがあります。

Django テストコードを書くときは、テスト対象のモジュールごとに `tests` パッケージ内に `test_<モジュール名>` といった名称のモジュールを作り、その中にテストコードを書くのが一般的です。
`tests.py` ファイルを残しておいてもトラブルの元になるだけです。
`python manage.py startapp <アプリ名>` コマンドでアプリケーションを作成したら、すぐに削除するよう習慣づけると良いでしょう。

以下は、 log アプリ内に `tests.py` と `tests` パッケージの両方が存在する状態でテストを実行しようとしたときの、 `ImportError` 発生の例です。

```
(venv) PS D:\django_project> python manage.py test
Traceback (most recent call last):
  File "D:\django_project\manage.py", line 22, in <module>
    main()
  File "D:\django_project\manage.py", line 18, in main
    execute_from_command_line(sys.argv)
  File "D:\django_project\venv\Lib\site-packages\django\core\management\__init__.py", line
442, in execute_from_command_line
    utility.execute()
  File "D:\django_project\venv\Lib\site-packages\django\core\management\__init__.py", line
436, in execute
    self.fetch_command(subcommand).run_from_argv(self.argv)
  File "D:\django_project\venv\Lib\site-packages\django\core\management\commands\test.py",
line 24, in run_from_argv
    super().run_from_argv(argv)
  File "D:\django_project\venv\Lib\site-packages\django\core\management\base.py", line
412, in run_from_argv
    self.execute(*args, **cmd_options)
  File "D:\django_project\venv\Lib\site-packages\django\core\management\base.py", line
458, in execute
    output = self.handle(*args, **options)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "D:\django_project\venv\Lib\site-packages\django\core\management\commands\test.py",
line 68, in handle
    failures = test_runner.run_tests(test_labels)
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "D:\django_project\venv\Lib\site-packages\django\test\runner.py", line 1048, in
run_tests
    suite = self.build_suite(test_labels, extra_tests)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "D:\django_project\venv\Lib\site-packages\django\test\runner.py", line 898, in
build_suite
    tests = self.load_tests_for_label(label, discover_kwargs)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "D:\django_project\venv\Lib\site-packages\django\test\runner.py", line 872, in
load_tests_for_label
    tests = self.test_loader.discover(start_dir=label, **kwargs)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\kbrah\AppData\Local\Programs\Python\Python311\Lib\unittest\loader.py",
line 322, in discover
    tests = list(self._find_tests(start_dir, pattern))
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\kbrah\AppData\Local\Programs\Python\Python311\Lib\unittest\loader.py",
line 377, in _find_tests
    tests, should_recurse = self._find_test_path(full_path, pattern)
                            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\kbrah\AppData\Local\Programs\Python\Python311\Lib\unittest\loader.py",
line 429, in _find_test_path
    raise ImportError(
ImportError: 'tests' module incorrectly imported from 'D:\\django_project\\log\\tests'.
Expected 'D:\\django_project\\log'. Is this module globally installed?
```