Django使用心得（二） 使用TestCase测试接口

> 使用TestCase测试接口，养成良好的编码测试习惯

在接触开源社区 Github 之后，发现特别多的开源项目都会有单元测试 TestCase 。但是在步入工作后，从业了两个创业公司，发现大多数程序员都没有养成写单元测试的习惯。

在目前的公司面试了一些程序员，他们的工作经验平均都有三年以上，但是都没有编写单元测试的习惯。 问到 "为什么不去编写单元测试呢？"，无非就是回答 "没有时间" 、 "写的都是接口，直接用客户端工具测试一下就可以了" 。

在笔者使用了 Django 框架自带的 TestCase 之后，发现用 TestCase 测试接口不仅比一些 客户端工具方便，而且还能降低在对代码进行修改之后出现 BUG 的几率， 特别是一些对代码有严重的洁癖喜欢优化代码的程序员来说真的非常有用。

而且运用框架的 TestCase 编写单元测试，还能结合一些 CI 工具来实现自动化测试，这个我也会专门写一篇文章来介绍我利用 Gitlab CI 结合 Django 的 TestCase 实现自动化测试的一些心得。

Toc

h2

类的结构

过 TestCase 的读者，先简单介绍一下 TestCase 的类结构。

常见的 TestCase 由 setUp 函数、 tearDown 函数和 test_func 组成。

这里 test_func 是指你编写了测试逻辑的函数，而 setUp 函数则是在 test_func 函数之前执行的函数， tearDown 函数则是在 test_func 执行之后执行的函数。

Top
Foot

development_of_test_habits/tests/test_demo.py view raw This Github Sample is by elfgzp

```python
from django.test import TestCase


class Demo(TestCase):
    def setUp(self):
        print('setUp')

    def tearDown(self):
        print('tearDown')

    def test_demo(self):
        print('test_demo')

    def test_demo_2(self):
        print('test_demo2')
```

我们可以通过在 Django 项目的根目录运行以下命令来运行这个单元测试

```
python manage.py test development_of_test_habits.tests.test_demo.Demo
```

如果使用 Pycharm 来运行的话可以直接点击类左侧的运行箭头，更加方便地运行或者 Debug 这个单元测试。

结果清晰的看到这个单元测试的执行顺序。

```
 database for alias 'default'...
identified no issues (0 silenced).
.setUp
test_demo2
tearDown
.
----------------------------------------------------------------------
Ran 2 tests in 0.001s
```

Toc

Top
Foot

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

此外还可以从运行结果看到，在测试之前单元测试创建了一个测试数据库。

> Creating test database for alias ‘default’…

然后在测试结束将数据库摧毁。

> Destroying test database for alias ‘default’…

这个也就是在继承了 Django 框架中的 TestCase ，它已经帮你实现的一些逻辑方便用于测试，所以我们不需要在 setUp 和 tearDown 函数中实现这些逻辑。

利用TestCase测试接口

h2

接下来讲一下我们如何使用 TestCase 来测试接口的，首先我们编写一个简单的接口，这里笔者是用 Django Rest Framework 的 APIView 来编写的，读者也可以使用自己管用的方法来编写。

development_of_test_habits/views/hello_test_case.py view raw This Github Sample is by elfgzp

```python
from rest_framework.views import APIView
from rest_framework.response import Response


class HelloTestCase(APIView):
    def get(self, request, *args, **kwargs):
        return Response({
            'msg': 'Hello %s I am a test Case' % request.query_params.get('nam
        })
```

然后这个接口类加到我们的路由中。

development_of_test_habits/urls.py view raw This Github Sample is by elfgzp

```python
django.urls import path
development_of_test_habits import views

tterns = [
ath('hello_test_case', views.HelloTestCase.as_view(), name='hello_test_case'),
```

接下来我们编写一个 HelloTestCase 的单元测试类来测试我们的测试用例。

f_test_habits/tests/test_hello_test_case.py view raw This Github Sample is by elfgzp

```python
ngo.urls import resolve, reverse
ngo.test import TestCase


class HelloTestCase(TestCase):
    def setUp(self):
        self.name = 'Django'

    def test_hello_test_case(self):
        url = '/test_case/hello_test_case'
        # url = reverse('hello_test_case')
```

Toc

Top
Foot

```
        # Input: print(resolve(url))
        # Output: ResolverMatch(func=development_of_test_habits.views.hello_te
        response = self.client.get(url)
        self.assertEqual(response.status_code, 200)  # 期望的Http相应码为200
        data = response.json()
        self.assertEqual(data['msg'], 'Hello , I am a test Case')  # 期望的msg

        response = self.client.get(url, {'name': self.name})
        self.assertEqual(response.status_code, 200)  # 期望的Http相应码为200
        data = response.json()
        self.assertEqual(data['msg'], 'Hello Django I am a test Case')  # 期望
```

在 setUp 函数中，我定义了一个 name 属性并且赋值为 Django 便于后面使用。

单元测试测试接口主要分为下面几个重要的内容。

请求的路由地址

h3

在测试接口时无非就是发起请求，检查返回的状态嘛和响应内容是否正确。发请求肯定少不了 url 地址，这里有两种方式来配置请求地址。
1.直接设置请求地址

```
url = '/test_case/hello_test_case'
```

2.透过 django.urls.reverse 函数和在路由设置的 name 来得到请求的地址

```
url =  reverse('hello_test_case')
```

这里在介绍以下我们还可以通过 django.urls.resolve 和 url 得到对应的接口类或者接口函数`。

请求的客户端

h3

发起请求我们除了需要路由外，我们还需要一个发起请求的客户端。python的 requests 库就是很好的客户端工具，只不过 Django 在它的 TestCase 类 中已经集成了一个客户端工具，我们只需要调用 TestCase 的 client 属性就可以得到一个客户端。

```
client = self.client
```

发起请求

h3

发起请求非常简单只需要一行代码，我们就可以通过请求得到它的响应体。

```
lf.client.get(url)
```

数只需要传入 data 参数。

```
response = self.client.get(url, {'name': self.name})
```

验证响应体

h3

Toc

Top
Foot

在单元测试中， TestCase 的 assertEqual 有点类似 python 的 assert 函数，除了 assertEqual 外还有 assertNotEqual 、 assertGreater 、 assertIn 等等。 这里笔者主要做了两个检查，一个是检查 status_code 是否等于 200 。

```
self.assertEqual(response.status_code, 200)  # 期望的Http相应码为200
```

另一个是检查响应内容是否正确。

```
data = response.json()
self.assertEqual(data['msg'], 'Hello , I am a test Case')  # 期望的msg返回结果为'Hello ,
```

这个就是最简单的测试请求的单元测试，但是在实际的接口中，我们是需要数据的，所以我们还需要生成测试数据。
这里介绍一个非常方便的库 mixer ， 可以方便在我们的单元测试中生成测试数据。

利用mixer在TestCase中生成测试数据

h2

首先我们定一个场景，比如说我们记录了学校班级的学生的作业，需要一个接口来返回学生的作业列表，并且这个接口是需要用户登陆后才可以请求的，定义的 models 和接口类如下。

development_of_test_habits/models.py view raw This Github Sample is by elfgzp

```
from django.db import models


class School(models.Model):
    name = models.CharField(max_length=32)


class Class(models.Model):
    school_id = models.ForeignKey(to=School, on_delete=models.PROTECT)
    name = models.CharField(max_length=32)


class Student(models.Model):
    class_id = models.ForeignKey(to=Class, on_delete=models.PROTECT)
    name = models.CharField(max_length=32)


class HomeWork(models.Model):
    student_id = models.ForeignKey(to=Student, on_delete=Student)
    name = models.CharField(max_length=32)
```

接口笔者用的是 Django rest framework 的 ReadOnlyModelViewSet 视图类实现的，实现的功能就是返回一个结果集， 并且 json 中有 HomeWork 的 School Name 、 Class Name 和 Student Name ，序列化代码如下。

f_test_habits/views/api/home_work.py view raw This Github Sample is by elfgzp

```
t_framework.viewsets import ReadOnlyModelViewSet
from rest_framework.permissions import IsAuthenticated

from development_of_test_habits.models import HomeWork
from development_of_test_habits.serializers import HomeWorkSerializer


class HomeWorkViewSet(ReadOnlyModelViewSet):
```

Toc

Top
Foot

```
    queryset = HomeWork.objects.all()
    serializer_class = HomeWorkSerializer
    permission_classes = (IsAuthenticated, )
```

development_of_test_habits/serializers.py view raw This Github Sample is by elfgzp

```
from rest_framework import serializers

from development_of_test_habits.models import HomeWork


class HomeWorkSerializer(serializers.ModelSerializer):
    class Meta:
        model = HomeWork
        fields = ('school_name', 'class_name', 'student_name', 'name')

    school_name = serializers.CharField(source='student_id.class_id.school_id.
    class_name = serializers.CharField(source='student_id.class_id.name', read
    student_name = serializers.CharField(source='student_id.name', read_only=T
```

最后把我们的接口类添加到路由中。

development_of_test_habits/serializers.py view raw This Github Sample is by elfgzp

```
urlpatterns = [
    path('hello_test_case', views.HelloTestCase.as_view(), name='hello_test_ca
    path('api/home_works', views.HomeWorkViewSet.as_view({'get': 'list'}), nam
]
```

完成接口的编写，可以开始写单元测试了，定义 `HomeWorkAPITestCase` 测试类并且在 `setUp` 中生成测试数据。

development_of_test_habits/tests/test_home_works_api.py view raw This Github Sample is by elfgzp

```
from django.test import TestCase
from django.urls import reverse

from django.contrib.auth.models import User

from mixer.backend.django import mixer

from development_of_test_habits import models


class HomeWorkAPITestCase(TestCase):
    def setUp(self):
        self.user = mixer.blend(User)

        self.random_home_works = [
            mixer.blend(models.HomeWork)
            for _ in range(11)
        ]
```


Top
Foot

`xer` 这个模块，这个模块会根据你定义的模型和模型的字段来随机生成测试数据，包括这个数据的外键数据 。 这样在我们这种层级非常多的关系型数据就非常的方便，否则需要一层一层的去生成数据。 代码中就利用 `mixer` 生成了一个随机的用户和11个随机的 `HomeWork` 数据。

接下来编写测试的逻辑代码。

development_of_test_habits/tests/test_home_works_api.py view This Github Sample is by elfgzp raw

```
class HomeWorkAPITestCase(TestCase):
    def setUp(self):
        self.user = mixer.blend(User)

        self.random_home_works = [
            mixer.blend(models.HomeWork)
            for _ in range(11)
        ]

    def test_home_works_list_api(self):
        url = reverse('home_works_list')

        response = self.client.get(url)
        self.assertEqual(response.status_code, 403)

        self.client.force_login(self.user)
        response = self.client.get(url)
        self.assertEqual(response.status_code, 200)
        data = response.json()
        self.assertEqual(len(data), len(self.random_home_works))

        data_fields = [key for key in data[0].keys()]

        self.assertIn('school_name', data_fields)
        self.assertIn('class_name', data_fields)
        self.assertIn('student_name', data_fields)
        self.assertIn('name', data_fields)
```

首先通过 `django.urls.reverse` 函数和接口的路由名称获得 `url` ， 第一步先测试用户在没有登陆的情况下请求接口，这里期望的请求响应码为 `403` 。

```
response = self.client.get(url)
self.assertEqual(response.status_code, 403)
```

我们通过 `client` 的一个 登陆 函数 `force_login` 来登陆我们随机生成的用户，再次请求接口，这次的期望的请求相应码就为 `200` 。

```
self.client.force_login(self.user)
response = self.client.get(url)
self.assertEqual(response.status_code, 200)
```

最后验证返回的结果数量是和结果中定义的字段是否正确。

```
data = response.json()
self.assertEqual(len(data), len(self.random_home_works))

data_fields = [key for key in data[0].keys()]

('school_name', data_fields)
('class_name', data_fields)
('student_name', data_fields)
('name', data_fields)
```

Toc

Top

Foot

中测试接口的最常见的流程。

h2

TestCase在使用中需要注意的一些问题

假设我们要在接口中增加 请求头 ， 以 HelloTestCase 接口为例，我们要增加一个 TEST_HEADER 的请求头，则在接口的逻辑处理中，就需要给这个请求头 加上 HTTP_ 前缀。

development_of_test_habits/views/hello_test_case.py view raw This Github Sample is by elfgzp

```
class HelloTestCase(APIView):
def get(self, request, *args, **kwargs):
    data = {
        'msg': 'Hello %s I am a test Case' % request.query_params.get('name',
    }
    test_header = request.META.get('HTTP_TEST_HEADER')
    if test_header:
        data['test_header'] = test_header
    return Response(data)
```

如果我们用客户端工具类似 Post Man 、 RestFul Client 等等，请求时只要在请求头中加上 TEST_HEADER 即可。但是在单元测试中，我们也需要把 HTTP_ 这个前缀加上，否则接口逻辑是无法获取的。

development_of_test_habits/tests/test_hello_test_case.py view raw This Github Sample is by elfgzp

```
def test_hello_test_case(self):
    url = '/test_case/hello_test_case'
    # url = reverse('hello_test_case')
    # Input: print(resolve(url))
    # Output: ResolverMatch(func=development_of_test_habits.views.hello_test_c
    response = self.client.get(url)
    self.assertEqual(response.status_code, 200)  # 期望的Http相应码为200
    data = response.json()
    self.assertEqual(data['msg'], 'Hello , I am a test Case')  # 期望的msg返回

    response = self.client.get(url, {'name': self.name})
    self.assertEqual(response.status_code, 200)  # 期望的Http相应码为200
    data = response.json()
    self.assertEqual(data['msg'], 'Hello Django I am a test Case')  # 期望的ms

    # 假设我们要在接口中增加请求头'TEST_HEADER'
    # 则在测试时需要加上前缀'HTTP_'最终的结果为'HTTP_TEST_HEADER'
    response = self.client.get(url, **{'HTTP_TEST_HEADER': 'This is a test hea
    data = response.json()
    self.assertEqual(data['test_header'], 'This is a test header.')
```

h2

总结

在用测试用例来测试接口后，笔者已经开始养成写完接口直接用单元测试来测试的习惯，这样不单是在给别人说明自己的接口的功能，还是减少线上环境的BUG都有明显的帮助， 希望读者也能用这成写单元测试的好习惯。

Toc

Top
Foot

撰写评论

发布

账号（邮件地址）

评论 1 时间正序 时间倒序 同感正序

黄智 2019.06.10 15:13

表情里没有大拇指

回复 0 0

Toc

Top

Foot