

## 一、pytest中的fixture是什么

为可靠的和可重复执行的测试提供固定的基线（可以理解为测试的固定配置，使不同范围的测试都能够获得统一的配置），fixture提供了区别于传统单元测试（setup/teardown）风格的令人惊喜的功能，而且pytest做得更炫。

## 二、pytest中fixture的使用

1. fixture 可以作为一个函数的参数被调用

```
import pytest

@pytest.fixture
def smtp_connection():
    import smtplib
    return smtplib.SMTP("smtp.gmail.com", 587, timeout=5)

def test_ehlo(smtp_connection):
    response, msg = smtp_connection.ehlo()
    assert response == 250
    assert 0 #强制断言失败
```

知乎 @把苹果咬哭

如上图，这里的 `test_ehlo` 函数，需要参数值 `smtp_connection`，pytest就是找到并且调用这个用 `@pytest.fixture` 装饰的 `smtp_connection` 函数。

换句话讲，被装饰器装饰的函数或者方法，仍然可以被调用。步骤是这样的：

- pytest 找到 test\_ 开头的函数，于是找到了 test\_ehlo
- test\_ehlo 这个测试函数，需要一个参数 smtp\_connection，于是函数 smtp\_connection 被找到
- smtp\_connection 被调用来创建一个实例

2. fixture可以在一个类、或者一个模块、或者整个session中被共享，加上范围即可

```
import pytest

@pytest.fixture(scope="module")
def smtp_connection():
    import smtplib
    return smtplib.SMTP("smtp.gmail.com", 587, timeout=5)

def test_ehlo(smtp_connection):
    response, msg = smtp_connection.ehlo()
    assert response == 250
    assert 0      #强制断言失败

def test_noop(smtp_connection):
    response, msg = smtp_connection.noop()
    assert response == 250
    assert 0
```

知乎 @把苹果咬哭

如上图，加入 scope="module" 的参数，可以让fixture function在每次模块测试的时候只请求一次。这样不同的test function在同一个test module中接收到的 smtpfixture参数都是一样的。这里的 smtp\_connection，就可以在这个模块中，共享使用。

类似的：

如果想在类中使用，那么 `@pytest.fixture(scope="class")`；

如果想在全部会话中使用，那么 `@pytest.fixture(scope="session")`。

### 3. fixture也可以单独存放

有的时候为了方便配置和访问，也可以将这样的fixture放到`conftest.py`文件中单独存放。

```
# content of conftest.py
import pytest
import smtplib

@pytest.fixture(scope="module")
def smtp():
    return smtplib.SMTP("smtp.gmail.com")
```

知乎 @把苹果咬哭

注意：该文件要放在case同级目录下哦

### 4. 同一个模块里出现多个范围的装饰

当出现多个范围装饰的时候，优先实例化范围优先级高的。

也就是优先级从大到小：`session-->module-->class-->function`

### 5. fixture的如何实现teardown

```
import smtplib
import pytest

@pytest.fixture(scope="module")
def smtp():
    smtp = smtplib.SMTP("smtp.gmail.com")
    yield smtp # provide the fixture value
    print("teardown smtp")
    smtp.close()
```

知乎 @把苹果咬哭

关键字yield后面的代码，即是测试结束后执行的代码

上图代码中的 `print("teardown smtp")`和`smtp.close()`，会在module范围内的最后一个测试完成后执行，不管测试中有没有exception的状态。

如果我们在装饰器中指定`scope="function"`，那么smtp将会在每次单个测试中建立和清除。

## 6. fixture中的参数 autouse

关于autouse，默认是False，如果不加 `scope='session'`，的使用autouse，只在当前module下有效。

① 如果你想一个module下的都用上，那就打开改成True，如下，这样就不需要往每个函数里传入fixture，例如：

```
import pytest

@pytest.fixture(autouse=True)
def fixture_for_function():
    print("这是用在函数上的fixture")

def test_1():
    print("执行了test1")

def test_2():
    print("执行了test2")

def test_3():
    print("执行了test3")

if __name__ == '__main__':
    pytest.main(["-s", "-q", "./pytest_fixture_demo.py"])
```

知乎 @把苹果咬哭

你想一个module下的都用上，那就打开改成True

看下运行结果：

```
这是用在函数上的fixture  
执行了test1  
·这是用在函数上的fixture  
执行了test2  
·这是用在函数上的fixture  
执行了test3  
.  
3 passed in 0.03 seconds  
[Finished in 0.7s]
```

运行结果

② 同样的，当我加上 `scope="class"` 时，当前模块下的所有类，都会调用一次fixture。

```
import pytest

@pytest.fixture(scope="class", autouse=True)
def fixture_for_class():
    print("这是用在测试类上的fixture")

class TestDemoClass1():

    def test_1(self):
        print("执行了test1")

    def test_2(self):
        print("执行了test2")

class TestDemoClass2():

    def test_3(self):
        print("执行了test3")

if __name__ == '__main__':
    pytest.main(["-s", "-q", "./test_fixture_demo.py"])
```

知乎 @把苹果咬哭

我加上scope="class"时，当前模块下的所有类，都会调用一次fixture

看下运行结果：

```
这是用在测试类上的fixture  
执行了test1  
· 执行了test2  
· 这是用在测试类上的fixture  
执行了test3  
·  
3 passed in 0.03 seconds  
[Finished in 0.8s] 知乎 @把苹果咬哭
```

运行结果

③ 文件中同时包含了function, class, 就不可以使用 `autouse` 了, 否则function也会执行到

---



```
import pytest

@pytest.fixture(scope="class", autouse=True)
def fixture_for_class():
    print("这是用在测试类上的fixture")

class TestDemoClass1():

    def test_1(self):
        print("执行了test1")

    def test_2(self):
        print("执行了test2")

class TestDemoClass2():

    def test_3(self):
        print("执行了test3")

def test_4():
    print("执行了test4")

if __name__ == '__main__':
    pytest.main(["-s", "-q", "./test_fixture_demo.py"])
```

文件中同时包含了function, class, 就不可以使用autouse了, 否则function也会执行到

看运行结果:

```
这是用在测试类上的fixture  
执行了test1  
. 执行了test2  
. 这是用在测试类上的fixture  
执行了test3  
. 这是用在测试类上的fixture  
执行了test4  
.  
4 passed in 0.03 seconds  
[Finished in 0.8s]
```

知乎 @把苹果咬哭

运行结果

#### ④ 如何运行类的fixture

你要使用的类。

```
import pytest

@pytest.fixture(scope="class")
def fixture_for_class():
    print("这是用在测试类上的fixture")

@pytest.mark.usefixtures("fixture_for_class")
class TestDemoClass1():

    def test_1(self):
        print("执行了test1")

    def test_2(self):
        print("执行了test2")

@pytest.mark.usefixtures("fixture_for_class")
class TestDemoClass2():

    def test_3(self):
        print("执行了test3")

    def test_4():
        print("执行了test4")

if __name__ == '__main__':
    pytest.main(["-s", "-q", ".", "/test fixture demo.py"])
```

知乎 @把苹果咬哭

@pytest.mark.usefixtures(这里是你要用的fixture),标记在你要使用的类

看下运行结果:

```
这是用在测试类上的fixture  
执行了test1  
.执行了test2  
.这是用在测试类上的fixture  
执行了test3  
.执行了test4  
.  
4 passed in 0.03 seconds  
[Finished in 0.8s]
```

知乎 @把苹果咬哭

运行结果，test\_4没有执行那个class的fixture的

**这里要注意的是：**scope= “class” ， 别忘记添加， 否则类下的每个function都会执行。

#### ⑤ 当scope='session'时， 要注意的点！

如果你的 scope='session' ,那么不要像function, class, module那样， 和case放在一起， 我们要放在另一个文件， conftest.py下， 才可以。

这里是同一个项目下的， 2个case文件：

```
test_fixture_demo.py • test_fixture_de
import pytest

class TestDemoClass1():

    def test_1(self):
        print("执行了test1")

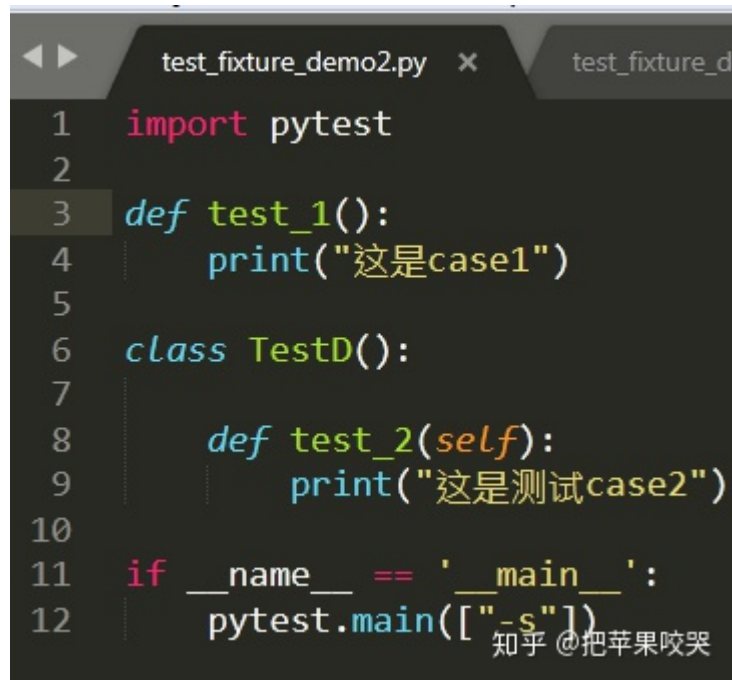
    def test_2(self):
        print("执行了test2")

class TestDemoClass2():

    def test_3(self):
        print("执行了test3")

def test_4():
    print("执行了test4")
```

同一个项目下的 test\_fixture\_demo.py


 A screenshot of a code editor window titled 'test\_fixture\_demo2.py'. The code is written in Python and includes imports, a function, a class with a method, and a main guard.
 

```

1  import pytest
2
3  def test_1():
4      print("这是case1")
5
6  class TestD():
7
8      def test_2(self):
9          print("这是测试case2")
10
11 if __name__ == '__main__':
12     pytest.main(["-s"])
  
```

 A watermark '知乎 @把苹果咬哭' is visible in the bottom right corner of the code area.

同一个项目下的 test\_fixture\_demo2.py

还有个同项目下的 conftest.py:


 A screenshot of a code editor window titled 'conftest.py'. The code defines a session-level fixture using the pytest fixture decorator.
 

```

1  import pytest
2
3  @pytest.fixture(scope="session", autouse=True)
4  def always_session():
5      print("这是session的fixture")
6
  
```

 A watermark '知乎 @把苹果咬哭' is visible in the bottom right corner of the code area.

还有个同项目下的 conftest.py

运行测试case，这个2个case文件下的所有case都会被找到，但是只会执行一次conftest.py中的fixture。

```
test_fixture_demo.py 这是session的fixture
执行了test1
. 执行了test2
. 执行了test3
. 执行了test4
.
test_fixture_demo2.py 这是case1
. 这是测试case2
.
```

知乎 @把苹果咬哭

```
===== 6 passed in 0.07 seconds =====
```

运行结果：这个2个case文件下的所有case都会被找到，但是只会执行一次conftest.py中的 fixture

#### ⑥ 当case里需要传入多个 fixture 或者 yield 怎么办呢？先后执行的顺序是什么？

在后续使用的过程中，遇到了case里我想传入多个yield，于是写了demo验证了下。下面上结果，就不放截图啦(形式传参 “before” 表示执行在case前，“yield” 表示执行在case后)

- `def test_1(before, yield)` 与 `def test_1(yield, before)`

结论：不管你顺序如何，依旧会先执行case前的before，case结束后执行yield

- `def test_2(before1, before2, yield)`

结论：这样有多个before，会依次按传参顺序先后执行。

- `def test_3(before, yield1, yield2)`

结论：这样有多个yield的，会依次从后往前执行，这里先执行yield2,再执行yield1。

关于pytest的fixture使用，就先写这么多吧，应该可以满足日常使用的场景了，欢迎补充。

把苹果咬哭 编辑于 2022-11-11 10:06 · IP 属地上海