链滴

# 使用 pytest 的 fixture 让测试更加灵活

作者：zyjImmortal
原文链接：https://ld246.com/article/1581258333075
来源网站：链滴

# 什么是fixture

fixture是测试函数运行前后，由pytest执行的外壳函数，它的作用是将一些非核心测试逻辑从测试函分离出来，以便于其他测试函数使用，同时保持这些边缘逻辑的一致性。

fixture中的代码可以定制，满足多变对的测试需求，包括定义传入测试中的数据集、配置测试前系统初始状态、为批量测试提供数据源等。

原文链接：使用 pytest 的 fixture 让测试更加灵活

```python
@pytest.fixture()
def provide_data():
    return 44


def test_data(provide_data):
    assert provide_data == 44
```

pytest.fixture()装饰器用于声明函数是一个fixture，如果测试函数的参数列表包含fixture的名字，那pytest会检测到，并在测试函数运行之前执行该fixture，fixture会将数据返回给测试函数。

```
(pytest_pratice) ➜ chapter3 git:(master) ✗ pytest test_fixture.py
============================= test session starts ==============================
platform darwin -- Python 3.8.0, pytest-5.3.5, py-1.8.1, pluggy-0.13.1
rootdir: /Users/zhouyajun/PycharmProjects/geek/pytest_pratice/chapter3
collected 1 item

test_fixture.py .

============================== 1 passed in 0.02s ===============================
(pytest_pratice) ➜ chapter3 git:(master) ✗
```

# fixture执行和销毁的逻辑

一般来说fixture会在测试函数执行前运行，但是如果fixture函数中包含了yield关键字，那么pytest会知道yield处停止，转而运行测试函数，等测试函数执行完毕后再回到fixture，继续执行yield后面的码。是不是似曾相识，每个测试用例都会有的setup和teardown。

原文链接：使用 pytest 的 fixture 让测试更加灵活

```python
# autouse=True 表示当前文件中的所有测试都将使用这个fixture,
# yield之前的代码在测试运行前执行，之后的代码在运行结束后执行
@pytest.fixture(autouse=True)
def tasks_db(tmpdir):
    # 初始化数据库，相当于setup
    tasks.start_tasks_db(str(tmpdir), 'tiny')
    yield
    # 关闭数据库连接，释放资源，相当于teardown
    tasks.stop_tasks_db()
```

pytest --setup-show test_add.py

--setup-show 会在控制台输出测试过程中执行的是什么以及执行的先后顺序，执行上面命令查看输内容

```
(pytest_pratice) ➜ func git:(master) ✗ pytest --setup-show test_add.py
============================= test session starts ==============================
platform darwin -- Python 3.8.0, pytest-5.3.5, py-1.8.1, pluggy-0.13.1
rootdir: /Users/zhouyajun/PycharmProjects/geek/pytest_pratice/chapter2/tasks_proj/tests, inifile: pytest.ini
collected 1 item

test_add.py
SETUP    S tmp_path_factory
        SETUP    F tmp_path (fixtures used: tmp_path_factory)
        SETUP    F tmpdir (fixtures used: tmp_path)
        SETUP    F tasks_db (fixtures used: tmpdir)
        func/test_add.py::test_add_returns_valid_id (fixtures used: request, tasks_db, tmp_path, tmp_path_factory, tmpdir).
        TEARDOWN F tasks_db
        TEARDOWN F tmpdir
        TEARDOWN F tmp_path
TEARDOWN S tmp_path_factory

============================== 1 passed in 0.02s ===============================
```

从截图中可以看到，测试函数被夹在中间，pytest将每一个fixture的执行分成SETUP和TEARDOWN部分，测试函数执行前有多个setup，是因为自定义的fixture中用到了内置的一些fixture，关于内置fixture后面再写文章界介绍。

fixture前面的F和S代表的是fixture的作用范围，F代表的函数级别的作用范围，S代表的是会话级别的为范围。

# 使用fixture传数据

fixture非常适合存放测试数据，并且它也可以返回任何类型的数据。

```python
@pytest.fixture()
def a_tuple():
    return (1, 'foo', None, {'bar': 23})


def test_a_tuple(a_tuple):
    assert a_tuple[3]['bar'] == 32
```

通过构造一个fixture并使其返回一个元祖，在测试函数中传入fixture，并是断言失败，下图是输出结果

```
(pytest_pratice) ➜ chapter3 git:(master) ✗ pytest test_fixture.py::test_a_tuple
============================= test session starts ==============================
platform darwin -- Python 3.8.0, pytest-5.3.5, py-1.8.1, pluggy-0.13.1
rootdir: /Users/zhouyajun/PycharmProjects/geek/pytest_pratice/chapter3
collected 1 item

test_fixture.py F

=================================== FAILURES ===================================
_____________________________ test_a_tuple _____________________________

a_tuple = (1, 'foo', None, {'bar': 23})

    def test_a_tuple(a_tuple):
>       assert a_tuple[3]['bar'] == 32
E       assert 23 == 32

test_fixture.py:19: AssertionError
============================== 1 failed in 0.04s ===============================
```

pytest给出了清晰的堆栈内容，还给出了引起assert异常的函数参数值，那么如果fixture中发生了异，是否会被堆栈跟踪，输出的信息是否有区别呢?

原文链接：使用 pytest 的 fixture 让测试更加灵活


在fixture中加一个注定失败的断言，运行测试用例，pytest正确定位了fixture中发生的异常，而且并有报告为FAIL，而是ERROR，这个区分很清晰，如果是测试结果被标记为了FAIL，那么用户就知道失败发生在测试用例中ongoing，而不是依赖的fixture中。

后续。。。。

原文链接：使用 pytest 的 fixture 让测试更加灵活