什么是pytest

- pytest 能够支持简单的单元测试和复杂的功能测试；
- pytest 可以结合 Requests 实现接口测试； 结合 Selenium、Appium 实现自动化功能测试；
- 使用 pytest 结合 Allure 集成到 Jenkins 中可以实现持续集成。
- pytest 支持 315 种以上的插件；

为什么选择pytest

- 报告
- 多线程
- 顺序控制
- 丰富的第三方插件
- 兼容unitest

Pytest 测试用例编写规则是什么？

类型	规则
文件	test_开头 或者 _test 结尾
类	Test 开头
方法/函数	test_开头
注意：测试类中不可以添加 __init__ 构造函数	

Pytest 测试框架结构（setup/teardown）

类型	规则
setup_module/teardown_module	全局模块级
setup_class/teardown_class	类级，只在类中前后运行一次（常用）
setup_function/teardown_function	函数级，在类外
setup_method/teardown_methond	方法级，类中的每个方法执行前后
setup/teardown	在类中，运行在调用方法的前后（重点、常用）

Mark：标记测试用例

- 场景:只执行符合要求的某一部分用例 可以把一个web项目划分多个模块，然后指定模块名称执行。
- 解决: 在测试用例方法上加 @pytest.mark.标签名
- 执行: -m 执行自定义标记的相关用例
  - pytest -s test_mark_zi_09.py -m=webtest
  - pytest -s test_mark_zi_09.py -m apptest
  - pytest -s test_mark_zi_09.py -m "not ios"

Mark：跳过(Skip)及预期失败(xFail)

- 这是pytest 的内置标签，可以处理一些特殊的测试用例，不能成功的测试用例
- skip - 始终跳过该测试用例
- skipif - 遇到特定情况跳过该测试用例
- xfail - 遇到特定情况,产生一个“期望失败”输出

Skip 使用场景

- 调试时不想运行这个用例
- 标记无法在某些平台上运行的测试功能
- 在某些版本中执行，其他版本中跳过
- 比如：当前的外部资源不可用时跳过
  - 如果测试数据是从数据库中取到的，
  - 连接数据库的功能如果返回结果未成功就跳过，因为执行也都报错
- 解决1：添加装饰器
  - @pytest.mark.skip
  - @pytest.mark.skipif
- 解决2：代码中添加跳过代码
  - pytest.skip(reason) 跳过后面的码不执行

xfail 使用场景

- 与skip 类似 ，预期结果为fail ，标记用例为fail
- 用法：添加装饰器@pytest.mark.xfail
  代码中用pytest.xskip(reason)

pytest断言

assert 表达式

assert 表达式 +消息

assert原生断言

assert expression1 ["," expression2]

缺点：条件判断失败的具体位置不知道，需要自己去比较

uniitest断言

assertIn（expect，result）断言包含（被包含的写前面）

assertEqual（expect，result）断言相等

assertTure（条件）断言是否为真。返回Ture或False

pytest断言

与unittest不同，pytest使用的是python自带的assert关键字来进行断言

Pytest对assert语句进行了重写，在测试用例中会给出具体的错误位置，还可以在assert后增加异常信息

常用断言

1、在自动化测试用例中，最常用的断言是相等断言（==）

断言字符串

断言函数或者接口返回值

大于、小于、不等于、in/not in

assert xx：判断xx为真

assert not xx：判断xx不为真

assert a in b：判断b包含a

assert a == b：判断a等于b

assert a !=b：判断a不等于b

参数化

参数化设计方法就是将模型中的定量信息变量化，使之成为任意调整的参数。

对于变量化参数赋予不同数值，就可得到不同大小和形状的零件模型

- 测试场景
  - 测试登录成功，登录失败(账号错误，密码错误)
  - 创建多种账号: 中文文账号，英文文账号
- 普通测试用例方法
  - Copy多份代码 or 读入入参数?
  - 一次性执行多个输入入参数
- pytest参数化实现方法
  - @pytest.mark.parametrize进行参数化和数 据驱动更灵活

Mark：参数化测试函数使用

- 单参数
- 多参数
- 用例重命名
- 笛卡尔积

单参数情况

```
search_list = ['appium','selenium','pytest'] @pytest.mark.parametrize('name',search_list)def test_search(name):    assert name in search_list
```

参数化：多参数情况（参数放在列表/元祖中存储）

```
@pytest.mark.parametrize("test_input,expected",[    ("3+5",8),("2+5",7),("7+5",12)])def test_mark_more(test_input,expected):    assert eval(test_input) == expected
```

参数化：用例重命名-添加ids参数

```
@pytest.mark.parametrize("test_input,expected",[    ("3+5",8),("2+5",7),("7+5",12)],ids=['add_3+5=8','add_2+5=7','add_3+5=12'])def test_mark_more(test_input,expected):    assert eval(test_input) == expected
```

参数化：笛卡尔积

- 比如
  - `a= [1,2,3]
  - b=[a,b,c] `
- 有几种组合形势 ？
  - (1,a),(1,b),(1,c)
  - (2,a),(2,b),(2,c)
  - (3,a),(3,b),(3,c)

Python 代码执行 pytest

- 使用 main 函数

```
if __name__ == '__main__':
    # 1、运行当前目录下所有符合规则的用例，包括子目录（test_*.py 和 *_test.py）
    pytest.main()
    # 2、运行test_mark1.py::test_dkej模块中的某一条用例
    pytest.main(['test_mark1.py::test_dkej','-vs'])
    # 3、运行某个 标签
    pytest.main(['test_mark1.py','-vs','-m','dkej'])

运行方式
  `python test_*.py `
```

- 使用python -m pytest调用 pytest（jenkins持续集成用到）

常用的异常处理方法

- try…except

```
try:
    可能产生异常的代码块
except [ (Error1, Error2, ... ) [as e] ]:
    处理异常的代码块1
except [ (Error3, Error4, ... ) [as e] ]:
    处理异常的代码块2
except  [Exception]:
    处理其它异常
```

- pytest.raises()

```
def test_raise():
    with pytest.raises(ValueError, match='must be 0 or None'):
        raise ValueError("value must be 0 or None")

def test_raise1():
    with pytest.raises(ValueError) as exc_info:
        raise ValueError("value must be 42")
    assert exc_info.type is ValueError
    assert exc_info.value.args[0] == "value must be 42"
```

Fixture 特点及优势

- 1、命令灵活：对于 setup,teardown,可以不起这两个名字
- 2、数据共享：在 conftest.py 配置里写方法可以实现数据共享，不需要 import 导入。可以跨文件共享
- 3、scope 的层次及神奇的 yield 组合相当于各种 setup 和 teardown
- 4、实现参数化

Fixture 在自动化中的应用- 基本用法

- 场景：

测试用例执行时，有的用例需要登陆才能执行，有些用例不需要登陆。

setup 和 teardown 无法满足。fixture 可以。默认 scope（范围）function

- 步骤：
  - 1.导入 pytest
  - 2.在登陆的函数上面加@pytest.fixture()
  - 3.在要使用的测试方法中传入（登陆函数名称），就先登陆
  - 4.不传入的就不登陆直接执行测试方法。

Fixture 在自动化中的应用 - 作用域(scope = model/)

取值	范围	说明
function	函数级	每一个函数或方法都会调用
class	类级别	每个测试类只运行一次
module	模块级	每一个.py文件调用一次
package	包级	每一个python包只调用一次(暂不支持)
session	会话级	每次会话只需要运行一次，会话内所有方法及类，模块都共享这个方法

Fixture 在自动化中的应用 - yield 关键字

- 场景：

你已经可以将测试方法【前要执行的或依赖的】解决了，

测试方法后销毁清除数据的要如何进行呢？

- 解决：

通过在fixture 函数中加入 yield 关键字，yield 是调用第一次返回结果，

第二次执行它下面的语句返回。

- 步骤：

在@pytest.fixture(scope=module)。

在登陆的方法中加 yield，之后加销毁清除的步骤

Fixture 在自动化中的应用 - 数据共享

- 场景：

你与其他测试工程师合作一起开发时，公共的模块要在不同文件中，要在大家都访问到的地方。

- 解决：

使用 conftest.py 这个文件进行数据共享，并且他可以放在不同位置起着不同的范围共享作用。

- 前提：
  - conftest 文件名是不能换的
  - 放在项目下是全局的数据共享的地方
- 执行：
  - 系统执行到参数 login 时先从本模块中查找是否有这个名字的变量什么的，
  - 之后在 conftest.py 中找是否有。
- 步骤：

将登陆模块带@pytest.fixture 写在 conftest.py

Fixture 在自动化中的应用 - 自动应用

场景：

不想原测试方法有任何改动，或全部都自动实现自动应用，

没特例，也都不需要返回值时可以选择自动应用

解决：

使用 fixture 中参数 autouse=True 实现

步骤：

在方法上面加 @pytest.fixture(autouse=True)

Fixture 在自动化中的应用 -参数化

场景：

测试离不开数据，为了数据灵活，一般数据都是通过参数传的

解决：

fixture 通过固定参数 request 传递

步骤：

在 fixture 中增加@pytest.fixture(params=[1, 2, 3, 'linda'])

在方法参数写 request，方法体里面使用 request.param 接收参数

pytest.ini 是什么

- pytest.ini 是pytest 的配置文件

- 可以修改 pytest 的默认行为
- 不能使用任何中文符号，包括汉字、空格、引号、冒号等等

功能

- 修改用例的命名规则
- 配置日志格式，比代码配置更方便
- 添加标签，防止运行过程报警告错误
- 指定执行目录
- 排除搜索目录

pytest 配置- 改变运行规则

```
;执行check_开头和 test_开头的所有的文件，后面一定要加*
python_files = check_* test_*
;执行所有的以Test和Check开头的类
python_classes = Test*  Check*
;执行所有以test_和check_开头的方法
python_functions= test_* check_*
```

pytest 配置- 添加默认参数

```
addopts = -v -s --alluredir=./results
```

pytest 插件分类

- 外部插件：pip install 安装的插件
- 本地插件：pytest自动模块发现机制（conftest.py存放的）
- 内置插件：代码内部的_pytest目录加载

pytest用例执行顺序（unittest是按照ascii码的顺序）

用例之间的顺序是文件之间按照ASCLL码排序，文件内的用例按照从上往下执行。

setup_module->setup_claas->setup_function->testcase->teardown_function->teardown_claas->teardown_module

可以通过第三方插件pytest-ordering实现自定义用例执行顺序

2.pytest-ordering的使用

通过装饰器的方法来控制case的执行顺序

1.方式一：

- 第一个执行： @ pytest.mark.run('first')
- 第二个执行： @ pytest.mark.run('second')
- 倒数第二个执行： @ pytest.mark.run('second_to_last')
- 最后一个执行： @ pytest.mark.run('last') 2.方式二：
- 第一个执行： @ pytest.mark.first
- 第二个执行： @ pytest.mark.second
- 倒数第二个执行： @ pytest.mark.second_to_last
- 第四个执行： @pytest.mark.last 3.方式三： 第一个执行： @ pytest.mark.run(order=1) 第二个执行： @ pytest.mark.run(order=2) 倒数第二个执行： @ pytest.mark.run(order=-2) 最后一个执行： @ pytest.mark.run(order=-1) 执行优先级： 0>较小的正数>较大的正数>无标记>较小的负数>较大的负数 导入 pytest @ pytest.mark.run （ order = -2 ） def test_three （ ） ： assert True @ pytest.mark.run （ order = -1 ） def test_four （ ） ： assert True @ pytest.mark.run （ order = 2 ） def test_two （ ） ： assert True @ pytest.mark.run （ order = 1 ） def test_one （ ） ： assert True 4.方式四： 导入 pytest @ pytest.mark.run （ after = 'test_second' ） def test_third （ ） ： assert True def test_second （ ） ： assert True @ pytest.mark.run （ before = 'test_second' ） def test_first （ ） ： assert True $ py.test test_foo.py -vv ============================测试会话开始========== ========== test_foo.py:11 ： test_first通过 test_foo.py:7 ：通过了test_second test_foo.py:4 ：test_third通过 ========== 4 = 0.02秒========== ==========

Pytest插件：xdist:执行分布式并发

pytest hook 介绍

- 是个函数，在系统消息触时被系统调用

- 自动触发机制
- Hook 函数的名称是确定的
- pytest有非常多的勾子函数
- 使用时直接编写函数体

Pytest全局用例共用之conftest.py详解 本文转自： 一、’conftest特点： 1、可以跨.py文件调用，有多个.py文件调用时，可让conftest.py只调用了一次fixture，或调用多次fixture 2、conftest.py与运行的用例要在同一个pakage下，并且有__init__.py文件 3、不需要import导入 conftest.py，pytest用例会自动识别该文件，放到项目的根目录下就可以全局目录调用了，如果放到某个package下， 那就在改package内有效，可有多个conftest.py 4、conftest.py配置脚本名称是固定的，不能改名称 5、conftest.py

文件不能被其他文件导入 6、所有同目录测试文件运行前都会执行conftest.py文件 二、’conftest用法：conftest文件实际应用需要结合fixture来使用，fixture中参数scope也适用conftest中fixture的特性，这里再说明一下 1、fixture源码详解 fixture（scope='function'，params=None，autouse=False，ids=None，name=None）： fixture里面有个scope参数可以控制fixture的作用范围，scope：有四个级别参数"function"（默认），"class"，"module"，"session params：一个可选的参数列表，它将导致多个参数调用fixture功能和所有测试使用它。 autouse：如果True，则为所有测试激活fixture func可以看到它。如果为False则显示需要参考来激活fixture ids：每个字符串id的列表，每个字符串对应于params这样他们就是测试ID的一部分。如果没有提供ID它们将从params自动生成 name：fixture的名称。这默认为装饰函数的名称。如果fixture在定义它的统一模块中使用，夹具的功能名称将被请求夹具的功能arg遮蔽， 解决这个问题的一种方法时将装饰函数命令"fixture_<fixturename>"然后使用"@pytest.fixture（name='<fixturename>'）"。 2、fixture的作用范围 fixture里面有个scope参数可以控制fixture的作用范围：session>module>class>function -function：每一个函数或方法都会调用 -class：每一个类调用一次，一个类中可以有多个方法 -module：每一个.py文件调用一次，该文件内又有多个function和class -session：是多个文件调用一次，可以跨.py文件调用，每个.py文件就是module function默认模式 @pytest.fixture(scope='function')或 @pytest.fixture() 3、conftest结合fixture的使用 conftest中fixture的scope参数为session，所有测试.py文件执行前执行一次 conftest中fixture的scope参数为module，每一个测试.py文件执行前都会执行一次conftest文件中的fixture conftest中fixture的scope参数为class，每一个测试文件中的测试类执行前都会执行一次conftest文件中的fixture conftest中fixture的scope参数为function，所有文件的测试用例执行前都会执行一次conftest文件中的fixture 三、conftest应用场景 1、每个接口需共用到的token 2、每个接口需共用到的测试用例数据 3、每个接口需共用到的配置信息