unittest框架

1.定义

unittest是Python单元测试框架。官方库

unittest中有4个重要的概念：test fixture（前置和后置，特殊函数）, test case(测试用例), test suite(套件), test runner（执行器）

2.书写规范

```
1.  文件名字命名：文件名不能写中文，不能写unittest，不能直接写testcase
2.  步骤
        1.导包import unittest
        2.继承unittest.TestCase
        class Add(unittest.TestCase)
            def test_1(self):
                pass
        3.方法名称必须以test字母开头
```

命令行

```
python -m unittest base_unittest.Add.test_1
```

TestSuite (测试套件)

说明：
多条测试用例集合在一起，就是一个TestSuite (测试套件)

第一种方法：创建suite对象，把测试用例依次加入suite套件

1. 实例化： suite = unittest.TestSuite() (suite：为TestSuite实例化的名称)
2. 添加用例：suite.addTest(ClassName("MethodName")) (ClassName：为类名；MethodName：为方法名)

```
suite.addTest(Add('test_1'))
suite.addTests([Add('test_1')])
```

第二种方法：从文件中加载测试用例，然后形成测试套件

```
# 1.从文件中加载测试用例，形成suite套件
suite = unittest.defaultTestLoader.discover('script','unittest*.py')
```

第三种方法：从测试类中加载用例,形成suite套件

```
# 从测试类中加载用例,形成suite套件
 suite = unittest.defaultTestLoader.loadTestsFromTestCase(Add)
# suite = unittest.defaultTestLoader.loadTestsFromModule(unittest_study)
```

Runner(执行器)

1.普通文本执行器

```
runner = unittest.TextTestRunner()
runner.run(suite)
```

2.能生成测试报告的执行器

```
from tool.HTMLTestRunner import HTMLTestRunner
with open('./report/add.html','wb') as f:
    HTMLTestRunner(f,title='加法运算',description='加法用例的设计').run(suite)
```

> 参数说明
>
> stream 文件流
>
> title 报告的名字
>
> description 报告的详细描述
>
> verbosity
>
> =1的时候 默认值为1，不限制完整结果，即单个用例成功输出'.',失败输出'F',错误输出'E'
>
> =2的时候，需要打印详细的返回信息

Fixture

说明：Fixture是一个概述，对一个测试用例环境的初始化和销毁就是一个Fixture

Fixture控制级别：

1. 方法级别

   1.1 运行于测试方法的始末，即：运行一次测试方法就会运行一次setUp和tearDown

   1.2 使用：

   1.2.1 初始化(前置处理):

```
def setUp(self): --> 首先自动执行
    pass
```

   1.2.2 销毁(后置处理):

```
def tearDown(self): --> 最后自动执行
    pass
```

2. 类级别

2.1 作用

运行于测试类的始末，即：每个测试类只会运行一次setUpClass和tearDownClass

2.2 使用

2.2.1 初始化(前置处理):

```
@classmethod
def setUpClass(cls): --> 首先自动执行
    pass
```

2.2.2 销毁(后置处理):

```
@classmethod
def tearDownClass(cls): --> 最后自动执行
    pass
```

断言

概念：让程序代替人为判断测试程序执行结果是否符合预期结果的过程

【掌 握】
assertEqual(expected, actual, msg=None) 验证expected==actual，不等则fail
assertIn(member, container, msg=None) 验证是否member in container

参数化

参数化：unittest本身没有参数化这个功能，需要第三方辅助，可以用ddt来实现

参数化数据

```
data = [{"x": 1, "y": 3, "expect": 4},
        {"x": 2, "y": 3, "expect": 5},
        {"x": 10, "y": 20, "expect": 30}]
```

```
data =[(1,3,4),(2,3,5),(10,20,30)]
```

1.下载ddt

```
pip install ddt
```

2.用法

2.1 现在测试类上面加一个装饰器

```
@ddt.ddt
class Add(unittest.TestCase):
    pass
```

2.2然后在需要做参数化的测试用例上面加一个装饰器

```
 # 循环遍历data，然后把每一个元素，赋值给test_4里面的参数
@ddt.data(*data)
def test_4(self, test_data):
    print(test_data)
```

> 列表套元组,列表嵌套列表或者列表嵌套字典，然后给ddt传入的必须是每一条可迭代对象，
> 所以传入之前如果是列表嵌套，那么需要解包，
> ddt.data里面只要是可迭代对象就

跳过

定义：对于一些未完成的或者不满足测试条件的测试函数和测试类，可以跳过执行。

使用方式
直接将测试函数标记成跳过

```
@unittest.skip('代码未完成')
```

根据条件判断测试函数是否跳过

```
@unittest.skipIf(condition, reason)
```

pytest和unittest框架的区别：

1.用例设计对比

1.1 unittest

a. 测试类必须继承unittest.TestCase

b. 测试函数必须以"test_"开头

c. 测试类必须有unittest.main()方法

1.2 pytest

a. 测试文件的文件名必须以"test"*开头，或者以*"test"结尾

b. 测试类命名必须以"Test"开头

c. 测试函数名必须以"test"开头

d. 测试类里面不能使用`__init__`方法

总结：pytest是基于unittest衍生出来的新的测试框架，使用起来相对于unittest来说更简单、效率来说更高，pytest兼容unittest测试用例，但是反过来unittest不兼容pytest，所以说pytest的容错性更好一些！在使用交互逻辑上面pytest比unittest更全一些！

2.断言对比

2.1unittest

assertEqual(a, b) # 判断a和b是否相等

...

2.2pytest

pytest只需要用assert 来断言就行，assert 后面加需要断言的条件就可以了，例如：assert a = = b # 判断a是否等于b

总结：从断言上面来看，pytest的断言比unittest要简单些，unittest断言需要记很多断言格式，pytest只有assert一个表达式，用起来比较方便

3.前置和后置函数

unittest的前置和后置没有pytest的前置后置灵活，unittest的包括类和方法级别的，pytest包括函数级别的，类级别的，方法级别的，模块级别的，除此之外还可以通过@pytest.fixture()去进行单独定义。

4.参数化

4.1 unittest：需要借助第三方包，ddt来进行实现

4.2 pytest：通过装饰器@pytest.mark.parametrize来实现

5.报告

5.1 unittest ： 通过HTMLTestRunner生成报告

5.2 pytest：

a. 通过pytest-html生成html格式报告

b. 通过allure生成方案（很详细）

附件：断言资料

UnitTest断言方法

序号	断言方法	断言描述
1	assertEqual(arg1, arg2, msg=None)	验证arg1=arg2，不等则fail 【常用】
2	assertNotEqual(arg1, arg2, msg=None)	验证arg1 != arg2, 相等则fail
3	assertTrue(expr, msg=None)	验证expr是true，如果为false，则fail 【常用】
4	assertFalse(expr,msg=None)	验证expr是false，如果为true，则fail 【常用】
5	assertIs(arg1, arg2, msg=None)	验证arg1、arg2是同一个对象，不是则fail
6	assertIsNot(arg1, arg2, msg=None)	验证arg1、arg2不是同一个对象，是则fail
7	assertIsNone(expr, msg=None)	验证expr是None，不是则fail
8	assertIsNotNone(expr, msg=None)	验证expr不是None，是则fail
9	assertIn(arg1, arg2, msg=None)	验证arg1是arg2的子串，不是则fail
10	assertNotIn(arg1, arg2, msg=None)	验证arg1不是arg2的子串，是则fail
11	assertIsInstance(obj, cls, msg=None)	验证obj是cls的实例，不是则fail
12	assertNotIsInstance(obj, cls, msg=None)	验证obj不是cls的实例，是则fail
13	assertAlmostEqual (first, second, places = 7, msg = None, delta = None)	验证first约等于second。 palces: 指定精确到小数点后多少位，默认为7
14	assertNotAlmostEqual (first, second, places, msg, delta)	验证first不约等于second。 palces: 指定精确到小数点后多少位，默认为7 注： 在上述的两个函数中， 如果delta指定了值，则first和second之间的差值必须≤delta
15	assertGreater (first, second, msg = None)	验证first > second，否则fail
16	assertGreaterEqual (first, second, msg = None)	验证first ≥ second，否则fail
17	assertLess (first, second, msg = None)	验证first < second，否则fail

18	**assertLessEqual (first, second, msg = None)**	**验证first ≤ second，否则fail**
19	assertRegexpMatches (text, regexp, msg = None)	验证正则表达式regexp搜索匹配的文本text。regexp：通常使用re.search()
20	assertNotRegexpMatches (text, regexp, msg = None)	验证正则表达式regexp搜索不匹配的文本text。regexp：通常使用re.search() 说明：两个参数进行比较（＞、≥、＜、≤、约等、不约等）
21	assertListEqual(list1, list2, msg = None)	验证列表list1、list2相等，不等则fail，同时报错信息返回具体的不同的地方
22	assertTupleEqual (tuple1, tuple2, msg = None)	验证元组tuple1、tuple2相等，不等则fail，同时报错信息返回具体的不同的地方
23	assertSetEqual (set1, set2, msg = None)	验证集合set1、set2相等，不等则fail，同时报错信息返回具体的不同的地方
24	assertDictEqual (expected, actual, msg = None	验证字典expected、actual相等，不等则fail，同时报错信息返回具体的不同的地方