day10 课堂笔记

课程之前

复习和反馈

作业

今日内容

- unittest 框架的组成
  - TestLoader
  - Fixture
- 断言
- 跳过(某些用例由于某些原因不想执行)
- 参数化
- 测试报告

unittest 组成

TestLoader (测试加载)

```
TestLoader (测试加载), 作用和 TestSuite 的作用是一样的, 对 TestSuite 功能的补充,
用来组装测试用例的

比如: 如果 TestCase 的代码文件有很多, (10 20, 30 )
- 使用步骤
1. 导包
2. 实例化测试加载对象并添加用例    ---> 得到的是 suite 对象
3. 实例化 运行对象
4. 运行对象执行套件对象
```

代码实现

```
在一个项目中 TestCase(测试用例) 的代码,一般放在一个单独的目录 (case)
```

```
"""TestLoader 的使用"""
# 1, 导包
import unittest

# 2, 实例化加载对象并添加用例
# unittest.TestLoader().discover('用例所在的路径', '用例的代码文件名')
```

```python
# 用例所在的路径,建议使用相对路径, 用例的代码文件名可以使用 *(任意多个任意字符) 通配符
# suite = unittest.TestLoader().discover('./case', 'hm*.py')
# suite = unittest.TestLoader().discover('./case', '*test*.py')
# suite = unittest.TestLoader().discover('./case', '*test*')
suite = unittest.TestLoader().discover('./case', '*case1.py')

# 3, 实例化运行对象
# runner = unittest.TextTestRunner()
# # 4, 执行
# runner.run(suite)

# 可以将 3 4 步 变为一步
unittest.TextTestRunner().run(suite)

```

```python
# 1. 导包
# 2. 使用默认的加载对象并加载用例
# 3. 实例化运行对象并运行

"""TestLoader 的使用"""
# 1, 导包
import unittest

# 2, 使用默认的加载对象并加载用例
suite = unittest.defaultTestLoader.discover('case', 'hm_*.py')

# 可以将 3 4 步 变为一步
unittest.TextTestRunner().run(suite)


```

Fixture(测试夹具)

```
Fixture(测试夹具) 是一种代码结构

在某些特定的情况下 会自动执行
```

方法级别[掌握]

```
在每个测试方法(用例代码) 执行前后都会自动调用的结构
```

```
# 方法执行之前
def setUp(self):
    每个测试方法执行之前都会执行
    pass

# 方法执行之后
def tearDown(self):
    每个测试方法执行之后都会执行
    pass
```

类级别[掌握]

```
在每个测试类中所有方法执行前后 都会自动调用的结构(在整个类中 执行之前执行之后个一次)
```

```
# 类级别的Fixture 方法, 是一个 类方法
# 类中所有方法之前
@classmethod
def setUpClass(cls):
    pass

# 类中所有方法之后
@classmethod
def tearDownClass(cls):
    pass
```

模块级别[了解]

```
模块: 代码文件
在每个代码文件执行前后执行的代码结构
```

```
# 模块级别的需要写在类的外边直接定义函数即可
# 代码文件之前
def setUpModule():
    pass

# 代码文件之后
def tearDownModule():
    pass
```

```
方法级别和类级别的 前后的方法,不需要同时出现,根据用例代码的需要自行的选择使用
```

案例

```
1. 打开浏览器(整个测试过程中就打开一次浏览器)  类级别
2. 输入网址(每个测试方法都需要一次)  方法级别
```

```
3. 输入用户名密码验证码点击登录(不同的测试数据) 测试方法
4. 关闭当前页面(每个测试方法都需要一次)  方法级别
5. 关闭浏览器(整个测试过程中就关闭一次浏览器) 类级别
------
1. 打开浏览器(整个测试过程中就打开一次浏览器)  类级别
2. 输入网址(每个测试方法都需要一次)  方法级别
3. 输入用户名密码验证码点击登录(不同的测试数据) 测试方法
4. 关闭当前页面(每个测试方法都需要一次)  方法级别
2. 输入网址(每个测试方法都需要一次)  方法级别
3. 输入用户名密码验证码点击登录(不同的测试数据) 测试方法
4. 关闭当前页面(每个测试方法都需要一次)  方法级别
2. 输入网址(每个测试方法都需要一次)  方法级别
3. 输入用户名密码验证码点击登录(不同的测试数据) 测试方法
4. 关闭当前页面(每个测试方法都需要一次)  方法级别
5. 关闭浏览器(整个测试过程中就关闭一次浏览器) 类级别
```

```python
import unittest


class TestLogin(unittest.TestCase):
    def setUp(self):
        """每个测试方法执行之前都会先调用的方法"""
        print('输入网址......')

    def tearDown(self) -> None:
        """每个测试方法执行之后都会调用的方法"""
        print('关闭当前页面......')

    @classmethod
    def setUpClass(cls) -> None:
        print('------1. 打开浏览器')

    @classmethod
    def tearDownClass(cls) -> None:
        print('------5. 关闭浏览器')

    def test_1(self):
        print('输入正确用户名密码验证码,点击登录 1')

    def test_2(self):
        print('输入错误用户名密码验证码,点击登录 2')
```

断言

```
让程序代替人工自动的判断预期结果和实际结果是否相符.

断言的结果有两种:
> True, 用例通过
> False, 代码抛出异常, 用例不通过

在 unittest 中使用断言, 都需要通过 self.断言方法  来试验
```

assertEqual

```
self.assertEqual(预期结果, 实际结果)   # 判断预期结果和实际结果是否相等
1. 如果相等, 用例通过
2. 如果不相等,用例不通过, 抛出异常
```

assertIn

```
self.assertIn(预期结果, 实际结果)   # 判断预期结果是否包含在实际结果中
1. 包含 ,用例通过
2. 不包含, 用例不通过, 抛出异常

assertIn('admin', 'admin')   # 包含
assertIn('admin', 'adminnnnnnnn')   # 包含
assertIn('admin', 'aaaaaadmin')   # 包含
assertIn('admin', 'aaaaaadminnnnnnnn')   # 包含
assertIn('admin', 'adddddddmin')   # 不是包含
```

```python
import unittest

from tools import login


class TestLogin(unittest.TestCase):
    def test_username_password_ok(self):
        """正确的用户名和密码: admin, 123456, 登录成功"""
        self.assertEqual('登录成功', login('admin', '123456'))

    def test_username_error(self):
        """错误的用户名: root, 123456, 登录失败"""
        self.assertEqual('登录失败', login('root', '123456'))

    def test_password_error(self):
        """错误的密码: admin, 123123, 登录失败"""
        self.assertEqual('登录失败', login('admin', '123123'))

    def test_username_password_error(self):
        """错误的用户名和错误的密码: aaa, 123123, 登录失败"""
        # self.assertEqual('登录失败', login('aaa', '123123'))
        self.assertIn('失败', login('aaa', '123123'))
```

参数化

```
参数化 在测试方法中，使用 变量 来代替具体的测试数据，然后使用传参的方法将测试数据传递给方法的变量
好处：相似的代码不需要多次书写.

工作中场景：
1. 测试数据一般放在 json 文件中
2. 使用代码读取 json 文件,提取我们想要的数据 ---> [(), ()] or [[], []]
```

安装插件

```
unittest 框架本身是不支持 参数化，想要使用参数化,需要安装插件来完成

- 联网安装(在 cmd 窗口安装 或者 )
 pip install parameterized

------
pip 是 Python 中包(插件) 的管理工具，使用这个工具下载安装插件
```

验证

```
pip list  # 查看到  parameterized

新建一个 python 代码文件，导包验证
from pa... import pa...
```

```
from parameterized import pa
                          parameterized   parameterized.parame…
```

参数化代码

```
1. 导包 unittest/ pa
2. 定义测试类
3. 书写测试方法(用到的测试数据使用变量代替)
4. 组织测试数据并传参
```

```python
# 1. 导包 unittest/ pa
import unittest
from parameterized import parameterized
from tools import login

# 组织测试数据  [(), (), ()] or [[], [], []]
data = [                                    列表中有几组数据， 就是几个用例
    ('admin', '123456', '登录成功'),
    ('root', '123456', '登录失败'),
    ('admin', '123123', '登录失败')
]


# 2. 定义测试类
class TestLogin(unittest.TestCase):
    # 3. 书写测试方法(用到的测试数据使用变量代替)
    @parameterized.expand(data)
    def test_login(self, username, password, expect):
        self.assertEqual(expect, login(username, password))

# 4. 组织测试数据并传参(装饰器 @)
```

数据和参数要保持一致

参数化 2

```json
[
  {
    "desc": "正确的用户名和密码",
    "username": "admin",
    "password": "123456",
    "expect": "登录成功"
  },
  {
    "desc": "错误的的用户名",
    "username": "root",
    "password": "123456",
    "expect": "登录失败"
  },
  {
    "desc": "错误的的密码",
    "username": "admin",
    "password": "123123",
    "expect": "登录失败"
  }
]
```

```python
# 1. 导包 unittest/ pa
import json
import unittest
from parameterized import parameterized
from tools import login
```

```python

# 组织测试数据  [(), (), ()] or [[], [], []]
def build_data():
    with open('data.json', encoding='utf-8') as f:
        result = json.load(f)   # [{}, {}, {}]
        data = []
        for i in result:  # i {}
            data.append((i.get('username'), i.get('password'), i.get('expect')))

    return data


# 2. 定义测试类
class TestLogin(unittest.TestCase):
    # 3. 书写测试方法(用到的测试数据使用变量代替)
    @parameterized.expand(build_data())
    def test_login(self, username, password, expect):
        self.assertEqual(expect, login(username, password))

# 4. 组织测试数据并传参(装饰器 @)


```

跳过

```
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，不想执行,可以使用跳过
使用方法，装饰器完成
代码书写在 TestCase 文件
```

```python
# 直接将测试函数标记成跳过
@unittest.skip('跳过额原因')
# 根据条件判断测试函数是否跳过 ，判断条件成立，跳过
@unittest.skipIf(判断条件, '跳过原因')
```

```python
import unittest

# version = 30
version = 29


class TestDemo(unittest.TestCase):
    @unittest.skip('没有什么原因,就是不想执行')
    def test_1(self):
        print('测试方法 1')

    @unittest.skipIf(version >= 30, '版本大于等于 30，不用测试')
```

```
    def test_2(self):
        print('测试方法 2')

    def test_3(self):
        print('测试方法 3')
```

测试报告

自带的测试报告

```
只有单独运行 TestCase 的代码,才会生成测试报告
```

生成第三方的测试报告

```
1. 获取第三方的 测试运行类模块 ， 将其放在代码的目录中
2. 导包 unittest
3. 使用  套件对象，加载对象 去添加用例方法
4. 实例化 第三方的运行对象 并运行 套件对象
```

```
# 1. 获取第三方的 测试运行类模块 ， 将其放在代码的目录中
# 2. 导包 unittest
import unittest
from HTMLTestRunner import HTMLTestRunner

# 3. 使用  套件对象，加载对象 去添加用例方法
suite = unittest.defaultTestLoader.discover('.', 'hm_05_pa1.py')
# 4. 实例化 第三方的运行对象 并运行 套件对象
# HTMLTestRunner()
# stream=sys.stdout，必填,测试报告的文件对象(open )，注意点,要使用 wb 打开
# verbosity=1，可选，报告的详细程度,默认 1 简略，2 详细
# title=None，可选，测试报告的标题
# description=None 可选，描述信息，Python 的版本，pycharm 版本

# file = 'report.html'   # 报告的后缀是.html
file = 'report1.html'   # 报告的后缀是.html
with open(file, 'wb') as f:
    # runner = HTMLTestRunner(f)  # 运行对象
    runner = HTMLTestRunner(f, 2, '测试报告', 'python 3.6.8 ')  # 运行对象

    # 运行对象执行套件，要写在 with 的缩进中
    runner.run(suite)

```

```
1. 组织用例文件(TestCase 里边)，书写参数化，书写断言，书写 Fixture，书写 跳过，如果单个测试测试文件，直接运行，得到测试报告，如果有多个测试文件，需要组装运行生成测试报告

2. 使用 套件对象组装，或者使用 加载对象组装

3. 运行对象 运行
3.1 运行对象 = 第三方的运行类(文件对象(打开文件需要使用 wb 方式))
3.2 运行对象.run(套件对象)
```

```python
import unittest
from HTMLTestRunnerCN import HTMLTestReportCN

# 组装用例方法
suite = unittest.defaultTestLoader.discover('.', '*pa1.py')

# 实例化运行对象
with open('report_cn.html', 'wb') as f:
    runner = HTMLTestReportCN(f)
    runner.run(suite)

```