0213课堂笔记

作者：沙陌

微信：Matongxue_2

1.作业

在学员管理系统中增加展示全班学员成绩平均分的业务

先在stu_operate.py的类中增加如下方法：

```
    @classmethod
    def show_avg_score(cls):
        # stus_dict的结构 {1:stu1,2:stu2,3:stu3,4:stu4}
        # 上述字典中，所有的value都是学员对象，每个学员都有各自的成绩

        # 平均分的计算规则是总分/学员个数
        sum = 0
        for stu in cls.stus_dict.values():
            score = stu.getScore()#得到学员成绩
            print(f'score:{score}')
            sum = sum+int(score)
        count = len(cls.stus_dict)
        return sum/count
```

再在stu_manager中增加对象操作调用

```
        elif op == 6:
            avg_score = StuOperate.show_avg_score()
            print(f'全班平均分是{avg_score}')
```

2.测试框架

unittest是python中非常流行的一个单元测试框架，那么在自动化测试中也有广泛的应用，接下来我们针对学员管理系统的核心业务方法进行测试

测试脚本编写

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-02-13 9:56
# @Copyright：北京码同学

# 在这里编写单元测试用例
import unittest

# 对于unittest的测试类来说，必须继承 unittest.TestCase
from lesson0123.stu_operate import StuOperate
from lesson0123.student import Student
```

```
class TestStudent(unittest.TestCase):

    # 这里的一个方法就是一条用例，方法名称必须以test开头才会被unittest识别
    # 测试新增学员这个核心业务
    def test_add_stu(self):
        # 所谓测试，就是调用新增学员的方法，根据你的测试设计传递参数
        stu = Student('7','沙陌',78326262)
        res = StuOperate.add_stu(stu)
        # 判断预期结果是否正确，这个过程我们叫做断言
        assert res == '添加成功'
    # 下面这个方法不是测试用例，不符合规则
    def aa(self):
        pass
```

注意可能会报错存储学员的文件找不到，那么修改file_load.py中的文件路径为绝对路径即可

3.unittest前置后置

```
class TestChangeStudent(unittest.TestCase):

    # 测试用例前置动作，数据准备
    @classmethod
    def setUpClass(cls) -> None:
        # 这里为当前测试类下的用例准备测试数据
        stu = Student('7', '沙陌', 78326262)
        res = StuOperate.add_stu(stu)
        print('类级别的前置动作')
    @classmethod
    def tearDownClass(cls) -> None:
        # 这里是在当前类下所有测试用例执行完成之后，要做的后置动作
        # 执行完成后将测试数据清掉
        StuOperate.delete_stu('7')
        print('类级别的后置动作')
    # 我希望在每条测试用例执行之前都帮我去准备数据
    def setUp(self) -> None:
        # 这里是为每条测试用例做的前置动作
        stu = Student('7', '沙陌', 78326262)
        res = StuOperate.add_stu(stu)
        print('方法级别的前置动作')
    def tearDown(self) -> None:
        # 这里是为每条测试用例做的后置动作
        StuOperate.delete_stu('7')
        print('方法级别的后置动作')
    def test_change_stu(self):

        res = StuOperate.change_stu('7',name='沙陌1')
        assert res == '修改成功'
        # 验证数据真的已经被修改了
        # {'1':stu1,'2':stu2,'3':stu3,'4':stu4}
        file_context = FileOperate().read() #这是个字典
        actual_name = file_context.get('7').getName() #获取学员姓名
        assert actual_name == '沙陌1'
    def test_change_stu1(self):

        res = StuOperate.change_stu('7',phone='63662553')
        assert res == '修改成功'
        # 验证数据真的已经被修改了
```

```
        file_context = FileOperate().read() #这是个字典
        actual_name = file_context.get('7').getPhone() #获取学员手机号
        assert actual_name == '63662553'
```

4.unittest 数据驱动

数据驱动指的是我将测试数据放在一个对象中，然后只写一条测试用例，这条测试用例会根据测试数据的多少来执行，比如你有三组测试数据，那么这用例就会三次

- 安装ddt库

```
# windows下
pip install ddt -i https://pypi.douban.com/simple/
# mac下
python3 -m pip install ddt -i https://pypi.douban.com/simple/
```

- 脚本实现

```
# 在类上加上数据驱动的装饰器
@ddt.ddt
class TestChangeStudentDDT(unittest.TestCase):


    @classmethod
    def setUpClass(cls) -> None:
        stu = Student('7', '沙陌', 78326262)
        res = StuOperate.add_stu(stu)
        # 清除id为67的学员
        StuOperate.delete_stu('67')
        print('类级别的前置动作')
    @classmethod
    def tearDownClass(cls) -> None:
        # 这里是为每条测试用例做的后置动作
        StuOperate.delete_stu('7')

    # 正常修改所有属性
    # 学员id不存在
    # 准备测试数据
    # 三列数据，第一列是学员id，第二列是要修改的属性，第三列是期望值
    test_data = [
        ['7', {'name': '张三', 'phone': '1773663', 'qq': '364646'}, '修改成
功'],
        ['67', {}, '学员id:67不存在']
    ]

    # 在测试用例上方增加ddt数据读取的装饰器，参数写你的数据对象，加上*
    @ddt.data(*test_data)
    def test_change(self,data):
        print(data)
        # 因为测试数据data是一个列表，所以我们需要得到每个测试数据
        id = data[0]
        kwargs = data[1] # {'name': '张三', 'phone': '1773663', 'qq':
'364646'}
        expect_value = data[2]
        # 将上述数据，使用到测试调用中
        # **kwargs 相当于将字典拆成了，name=张三,phone=1773663,qq=364646
```

```
        res = StuOperate.change_stu(id,**kwargs)
        assert res == expect_value


        # pass
if __name__ == '__main__':
    unittest.main()
```

5. 测试套件及测试报告

HTMLTestRunner.py在群文件下载

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-02-13 11:42
# @Copyright：北京码同学
import unittest

from lesson0213.HTMLTestRunner import HTMLTestRunner
from lesson0213.test_student import TestStudent

if __name__ == '__main__':
    # 在这里完成测试用例的组织
    # suite = unittest.TestSuite()
    # # 只执行TestStudent的test_add_stu测试用例
    # suite.addTest(TestStudent('test_add_stu'))


    # 通常我们测试脚本会有很多个，存放在某个目录下，那么我们可以批量加载
    suite = unittest.defaultTestLoader.discover('lesson0213','test*.py')

    # 加一个执行，右键执行才会出现
    # runner = unittest.TextTestRunner()
    # runner.run(suite)

    # 对于测试来说，最终我们需要一个比较直观的测试报告
    # 借助一个第三方的工具，来实现执行器，并生成测试报告
    # 重新创建一个执行器
    with open('testreport.html',mode='wb') as f:
        HTMLTestRunner(f,title='测试报告',description='这是个测试报告').run(suite)
```

6.商城项目业务

- 立即购买接口

  立即购买并不会完成订单创建，他只是将你要购买的产品信息，存入到redis缓存中

  立即购买接口在成功时只有响应状态码，没有响应信息

  sku_id 参数并不是我们理解的商品id，他是根据商品生成的一个产品id，可以通过查库得到

  找到数据的mtxshop_goods库

```
SELECT * FROM `es_goods_sku` WHERE goods_id=4942;
```

- 添加购物车接口

  添加购物车接口，也不会完成订单创建，他也是将信息存到redis缓存中，但是这个接口在成功时有响应状态码，也有响应信息

- 设置收货地址接口

  该接口设置下单时的收货地址，数据也是会存在redis缓存中，如果设置成功接口响应只有状态码，没有响应信息

- 设置支付方式

  由于咱们网站不支持在线支付，所以想完整的走后续流程，那么支付方式必须设置成货到付款

  参数值写COD表示货到付款

  数据也是会存在redis缓存中，如果设置成功接口响应只有状态码，没有响应信息

- 创建交易接口

  创建接口的参数里并没有订单相关信息，但是他会完成订单创建，数据主要是根据 **way**这个参数的值来的

  way如果是BUY_NOW,那么他会自动去redis缓存中读取当前用户通过 立即购买接口 形成的数据

  way如果是CART,那么他会自动去redis缓存中读取当前用户通过 添加购物车接口 形成的数据