0227课堂笔记

1.补充几个接口定义

在mtxshop_api.py中，修改或增加下述代码：

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-02-20 17:14
# @Copyright：北京码同学
import requests

session = requests.session()
token = ''
def buyer_login():
    url = 'http://www.mtxshop.com:7002/passport/login'

    headers = {
        'Authorization': ''
    }
    # 查询参数通常使用params来表示
    params = {
        'username': 'shamo',
        'password': 'e10adc3949ba59abbe56e057f20f883e',
        'captcha': '1512',
        'uuid': 'jsjdhdhdhdhdhdhh'
    }
    resp = session.request(url=url,method='post', headers=headers,
params=params)

    status_code = resp.status_code  # 获取响应状态码
    # print(status_code)
    # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对响应信
息是json格式
    # print(resp_info)
    print(resp_json)
    global token
    token = resp_json['access_token']
def add_cart():
    url = 'http://www.mtxshop.com:7002/trade/carts'
    global token
    headers = {
        'Authorization': token
    }
    # 查询参数通常使用params来表示
    params = {
        'sku_id': 5173,
        'num': 1
    }
    resp = session.request(url=url,method='post', headers=headers,
params=params)
```

```
    # status_code = resp.status_code  # 获取响应状态码
    # # print(status_code)
    # # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    # resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对响应
信息是json格式
    # # print(resp_info)
    # print(resp_json)
    return resp
def buy_now():
    # 添加购物车接口
    url = 'http://www.mtxshop.com:7002/trade/carts/buy'
    global token
    headers = {
        'Authorization': token
    }
    # 查询参数通常使用params来表示
    params = {
        'sku_id': 5173,
        'num': 1
    }
    resp = session.request(url=url,method='post', headers=headers,
params=params)

    # status_code = resp.status_code  # 获取响应状态码
    # # print(status_code)
    # # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    # resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对响应
信息是json格式
    # # print(resp_info)
    # print(resp_json)
    return resp
def set_address():
    # 设置收货地址接口
    url = 'http://www.mtxshop.com:7002/trade/checkout-params/address-id/3816'
    global token
    headers = {
        'Authorization': token
    }

    resp = session.request(url=url,method='post', headers=headers)

    # status_code = resp.status_code  # 获取响应状态码
    # # print(status_code)
    # # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    # resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对响应
信息是json格式
    # # print(resp_info)
    # print(resp_json)
    return resp
def set_payment_type():
    # 设置支付类型接口
    url = 'http://www.mtxshop.com:7002/trade/checkout-params/payment-type'
    global token
    headers = {
        'Authorization': token
    }
    # 查询参数通常使用params来表示
    params = {
```

```
        'payment_type': 'COD' #表示货到付款
    }
    resp = session.request(url=url,method='post', headers=headers,params=params)

    # status_code = resp.status_code  # 获取响应状态码
    # # print(status_code)
    # # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    # resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对响应信息是json格式
    # # print(resp_info)
    # print(resp_json)
    return resp

def create_trade():
    # 创建交易接口
    url = 'http://www.mtxshop.com:7002/trade/create'
    global token
    headers = {
        'Authorization': token
    }
    # 查询参数通常使用params来表示
    params = {
        'client': 'PC', #表示货到付款
        'way':'BUY_NOW'
    }
    resp = session.request(url=url,method='post', headers=headers,params=params)

    # status_code = resp.status_code  # 获取响应状态码
    # # print(status_code)
    # # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    # resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对响应信息是json格式
    # # print(resp_info)
    # print(resp_json)
    return resp
if __name__ == '__main__':
    buyer_login()
    add_cart()
```

2.测试框架pytest

第一步只是实现了各个接口的基本定义，并没有完成调用和测试，那么我们可以使用pytest这个测试框架针对接口实现测试

1. pytest测试框架安装

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

2. pytest测试脚本编写
    - 测试文件命令

      默认情况下，测试脚本文件必须以test_开头或者以下划线test结尾
    - 测试用例

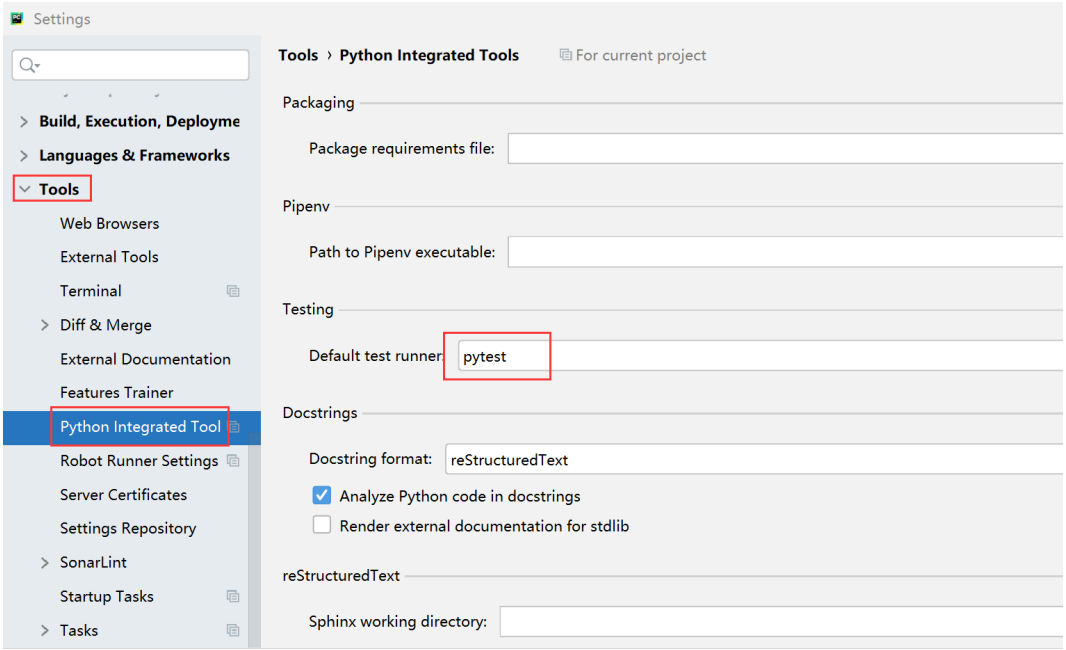
```python
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-02-27 9:53
# @Copyright：北京码同学
from requests_study.mtxshop_api import *

def test_add_cart():
    # 测试添加购物车接口
    # 先调用登录接口，否则会没有权限
    buyer_login()
    # 调用接口，得到响应对象
    resp = add_cart()
    # 拿到resp对象以后，根据期望结果的需求进行判断
    # 预期响应状态码为200
    status_code = resp.status_code
    assert status_code == 200
```

- 执行

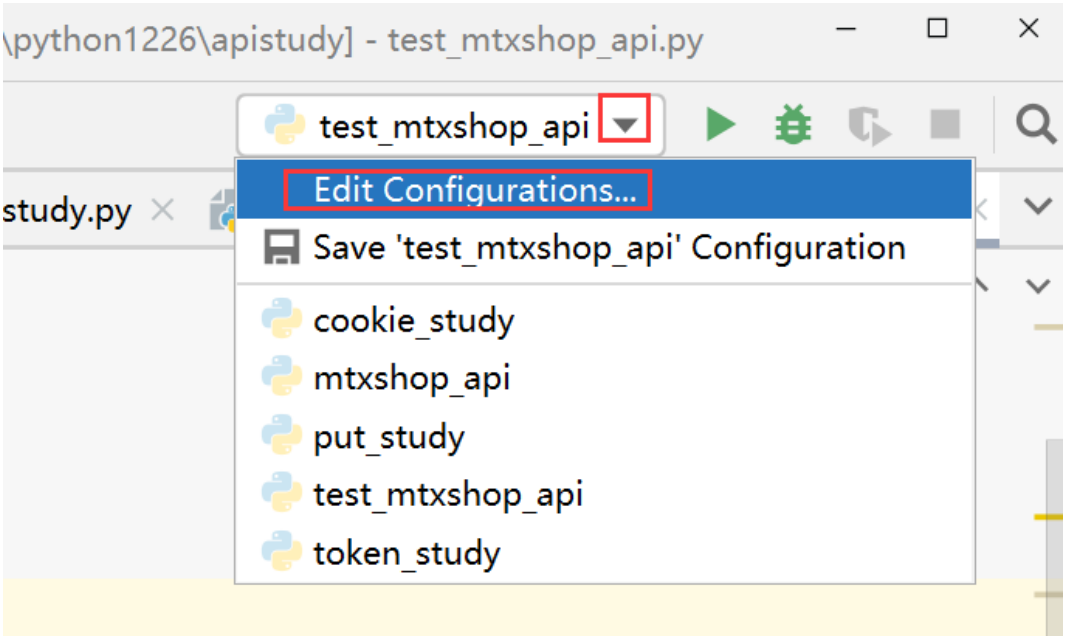
第一种方式在命令行执行，在pycharm中点下发的terminal，进入命令行模式，进入到要执行的脚本所在的目录中，执行如下指令

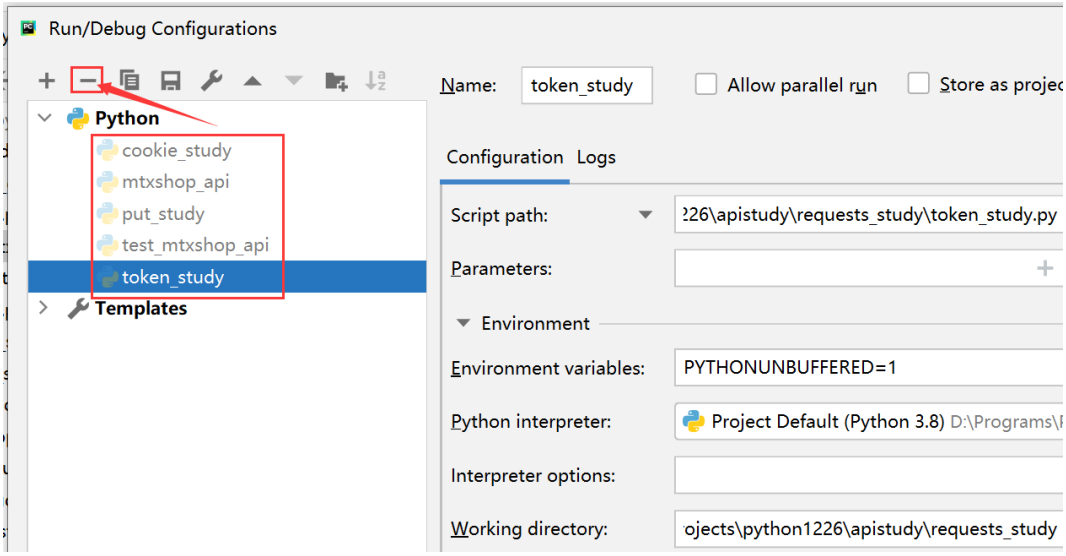
```
pytest -sv test_mtxshop_api.py
```

第二种方式，在测试文件上右键执行，需要按照如下设置后才行，File-->Settings

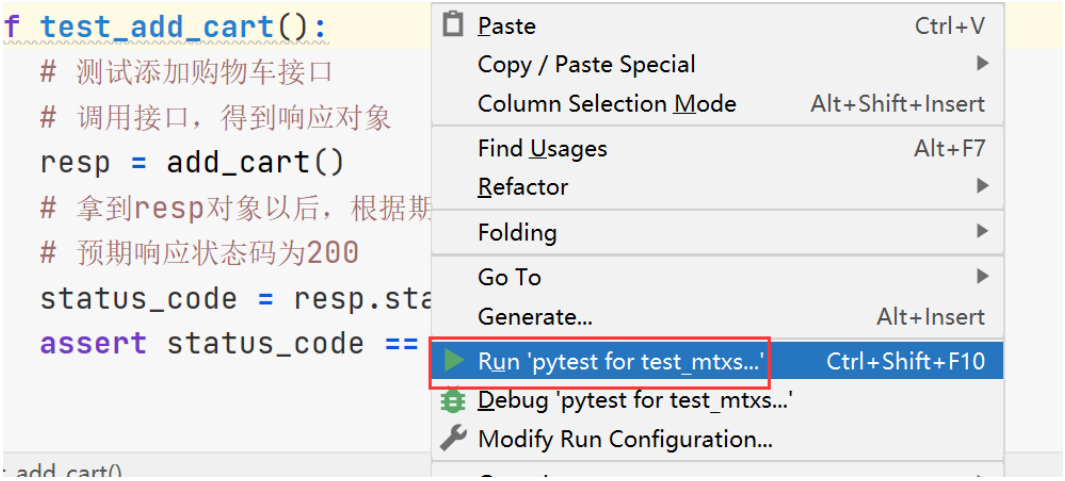


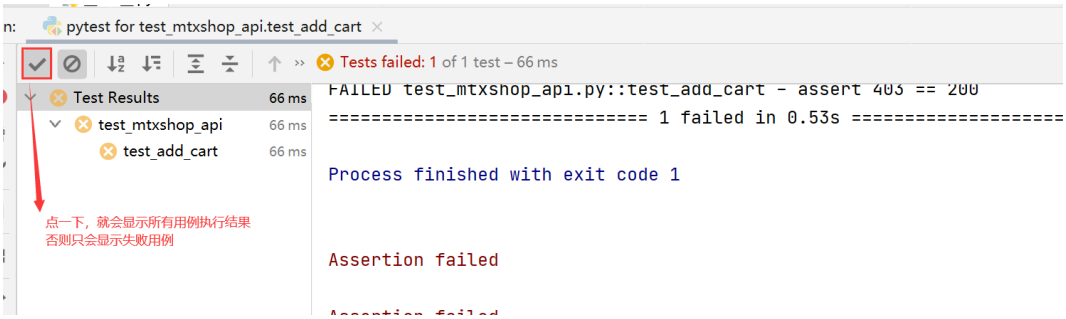
清除之前的执行记录





右键测试文件，会看到如下图，直接点击即可执行





3. 接口定义封装优化

在之前的封装中，每个接口的参数都是写死的，无法针对各个参数的测试用例传递数据，因此我们需要修改原来的接口定义中的参数传递

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-02-20 17:14
# @Copyright：北京码同学
import requests

session = requests.session()
token = ''
def buyer_login():
    url = 'http://www.mtxshop.com:7002/passport/login'

    headers = {
        'Authorization': ''
    }
    # 查询参数通常使用params来表示
    params = {
        'username': 'shamo',
        'password': 'e10adc3949ba59abbe56e057f20f883e',
        'captcha': '1512',
        'uuid': 'jsjdhdhdhdhdhdhh'
    }
    resp = session.request(url=url,method='post', headers=headers,
params=params)

    status_code = resp.status_code  # 获取响应状态码
    # print(status_code)
    # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对响
应信息是json格式
    # print(resp_info)
    print(resp_json)
    global token
    token = resp_json['access_token']
def add_cart(sku_id=5173,num=1):
    url = 'http://www.mtxshop.com:7002/trade/carts'
    global token
    headers = {
        'Authorization': token
    }
    # 查询参数通常使用params来表示
    params = {
        'sku_id': sku_id,
```

```
        'num': num
    }
    resp = session.request(url=url,method='post', headers=headers,
params=params)

    # status_code = resp.status_code  # 获取响应状态码
    # # print(status_code)
    # # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    # resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对
响应信息是json格式
    # # print(resp_info)
    # print(resp_json)
    return resp
def buy_now(sku_id=5173,num=1):
    # 添加购物车接口
    url = 'http://www.mtxshop.com:7002/trade/carts/buy'
    global token
    headers = {
        'Authorization': token
    }
    # 查询参数通常使用params来表示
    params = {
        'sku_id': sku_id,
        'num': num
    }
    resp = session.request(url=url,method='post', headers=headers,
params=params)

    # status_code = resp.status_code  # 获取响应状态码
    # # print(status_code)
    # # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    # resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对
响应信息是json格式
    # # print(resp_info)
    # print(resp_json)
    return resp
def set_address(address_id=3816):
    # 设置收货地址接口
    url = f'http://www.mtxshop.com:7002/trade/checkout-params/address-
id/{address_id}'
    global token
    headers = {
        'Authorization': token
    }

    resp = session.request(url=url,method='post', headers=headers)

    # status_code = resp.status_code  # 获取响应状态码
    # # print(status_code)
    # # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    # resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对
响应信息是json格式
    # # print(resp_info)
    # print(resp_json)
    return resp
def set_payment_type(payment_type='COD'):
    # 设置支付类型接口
    url = 'http://www.mtxshop.com:7002/trade/checkout-params/payment-type'
```

```
    global token
    headers = {
        'Authorization': token
    }
    # 查询参数通常使用params来表示
    params = {
        'payment_type': payment_type #表示货到付款
    }
    resp = session.request(url=url,method='post',
headers=headers,params=params)

    # status_code = resp.status_code  # 获取响应状态码
    # # print(status_code)
    # # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    # resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对
响应信息是json格式
    # # print(resp_info)
    # print(resp_json)
    return resp

def create_trade(client='PC',way='BUY_NOW'):
    # 创建交易接口
    url = 'http://www.mtxshop.com:7002/trade/create'
    global token
    headers = {
        'Authorization': token
    }
    # 查询参数通常使用params来表示
    params = {
        'client': client,
        'way':way
    }
    resp = session.request(url=url,method='post',
headers=headers,params=params)

    # status_code = resp.status_code  # 获取响应状态码
    # # print(status_code)
    # # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    # resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种方法只能针对
响应信息是json格式
    # # print(resp_info)
    # print(resp_json)
    return resp
if __name__ == '__main__':
    buyer_login()
    add_cart()
```

4. 添加购物车测试脚本编写

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-02-27 9:53
# @Copyright：北京码同学
from requests_study.mtxshop_api import *

def test_add_cart():
```

```
    # 测试添加购物车接口
    # 先调用登录接口，否则会没有权限
    buyer_login()
    # 调用接口，得到响应对象
    resp = add_cart()
    # 拿到resp对象以后，根据期望结果的需求进行判断
    # 预期响应状态码为200
    status_code = resp.status_code
    assert status_code == 200

# 在之前的封装中，每个接口的参数都是写死的，无法针对各个参数的测试用例传递数据
# 因此我们需要修改原来的接口定义中的参数传递

# 测试添加购物车sku_id不存在
def test_add_cart1():
    # 测试添加购物车接口
    # 先调用登录接口，否则会没有权限
    buyer_login()
    # 调用接口，得到响应对象
    resp = add_cart(sku_id=6354535)
    # 拿到resp对象以后，根据期望结果的需求进行判断
    # 预期响应状态码为500
    status_code = resp.status_code
    print(resp.json())
    assert status_code == 500
    # 如果还想针对响应信息中的某些字段做校验，就需要进一步操作
    # {'code': '451', 'message': '商品已失效，请刷新购物车'}
    # 判断响应中的业务码是451
    resp_json = resp.json() #获取响应结果,数据类型是字典
    code = resp_json['code']
    assert code == '451'
    # 判断message的值是 商品已失效，请刷新购物车
    message = resp_json['message']
    assert message == '商品已失效，请刷新购物车'
# 测试添加购物车num为0
def test_add_cart2():
    # 测试添加购物车接口
    # 先调用登录接口，否则会没有权限
    buyer_login()
    # 调用接口，得到响应对象
    resp = add_cart(num=0)
    # 拿到resp对象以后，根据期望结果的需求进行判断
    # 预期响应状态码为400
    status_code = resp.status_code
    print(resp.json())
    assert status_code == 400
    # 如果还想针对响应信息中的某些字段做校验，就需要进一步操作
    # {'code': '004', 'message': '加入购物车数量必须大于0'}
    # 判断响应中的业务码是004
    resp_json = resp.json() #获取响应结果,数据类型是字典
    code = resp_json['code']
    assert code == '004'
    # 判断message的值是 商品已失效，请刷新购物车
    message = resp_json['message']
    assert message == '加入购物车数量必须大于0'
```

5. pytest前置后置处理

   通过观察我们发现，每个接口在调用时都先调用了登录接口，这说明登录接口是他们共同的前置操作

   - 模块级别

     指的是在同一个pytest测试脚本文件中所有测试用例执行前后去做的动作

     修改测试脚本文件，增加以下代码，并注释各个用例之间的登录接口

```
# 模块级别的前置
def setup_module():
    buyer_login()
    print('这是在当前模块所有测试用例执行之前，只执行一次的前置动作')

# 比如当前模块下所有用例执行完成后，要清除相关的测试数据，那么可以使用后置
def teardown_module():
    print('当前模块下，所有用例执行完后，我来做后置动作')
```

   - 函数级别

     在面试时可能会碰到这样一个问题：接口自动化时，如果token失效了怎么办？

     方案：在每条测试用例执行前都重新获取token

     对应我们现在所写的脚本，就是增加一个函数级别的前置动作，在前置动作中调用登录接口，给token赋值

     函数级别的前置和后置，指的是在每条测试用例执行前后都要被执行的

```
# 函数级别的前置和后置
def setup_function():
    buyer_login()
    print('每条测试用例执行前，我都会执行')
def teardown_function():
    buyer_login()
    print('每条测试用例执行后，我都会执行')
```

6. pytest测试脚本的另一种编写方式

   之前我们都是使用纯函数的方式去编写测试脚本，pytest还支持使用类的方式去编写测试脚本

   基本默认规则是，测试类名称必须以Test开头

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-02-27 11:18
# @Copyright：北京码同学
from requests_study.mtxshop_api import *
class TestBuyNow():
    # 在类下面去编写测试脚本
    # 以一个方法作为一个测试用例，方法的默认规则是test_开头或者_test结尾

    def test_buy_now(self):
        buyer_login()
        resp = buy_now()
        # 做断言
        status_code = resp.status_code
```

```
        assert status_code == 200
    def test_buy_now1(self):
        buyer_login()
        resp = buy_now(sku_id=6354535)
        # 做断言
        status_code = resp.status_code
        print(resp.json())
        assert status_code == 500
        # {'code': '004', 'message': '不合法'}
        resp_json = resp.json()
        code = resp_json['code']
        assert code == '004'
        # 判断message的值是 商品已失效，请刷新购物车
        message = resp_json['message']
        assert message == '不合法'
```

我们发现类下方每条测试用例都先调用了登录，可以使用前置来优化

- 类级别

  指的是在当前类下测试用例执行前后，只执行一次的前置后置动作

- 方法级别

  指的是在当前类下每条测试用例执行前后，都会被执行的前后置动作

代码修改：

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-02-27 11:18
# @Copyright：北京码同学
from requests_study.mtxshop_api import *
class TestBuyNow():
    # 在类下面去编写测试脚本
    # 以一个方法作为一个测试用例，方法的默认规则是test_开头或者_test结尾
    def setup_class(self):
        buyer_login()
        print('当前类下用例执行之前，只执行一次')
    def teardown_class(self):
        print('当前类下用例执行之后，只执行一次')
    def setup_method(self):
        print('当前类下每条用例执行之前，执行一次')
    def teardown_method(self):
        print('当前类下每条用例执行之后，执行一次')
    def test_buy_now(self):
        # buyer_login()
        resp = buy_now()
        # 做断言
        status_code = resp.status_code
        assert status_code == 200
    def test_buy_now1(self):
        # buyer_login()
        resp = buy_now(sku_id=6354535)
        # 做断言
        status_code = resp.status_code
        print(resp.json())
        assert status_code == 500
```

```
        # {'code': '004', 'message': '不合法'}
        resp_json = resp.json()
        code = resp_json['code']
        assert code == '004'
        # 判断message的值是 商品已失效，请刷新购物车
        message = resp_json['message']
        assert message == '不合法'
```

7. 小总结

   pytest两种编写测试脚本的方式

   1. 纯函数式
   2. 面向对象式(类)

   建议在同一个项目中，采用一种即可，尽量不要混着用，我比较推荐面向对象式的编写

   前置后置学习了四种

   模块级别和函数级别，比较适用于纯函数式脚本编写模式

   类级别和方法级别，比较适合面向对象式的脚本编写模式

8. pytest参数化

   在我们之前所写的添加购物车和立即购买的测试用例中，我们发现调用的接口以及断言几乎都差不多，每条用例测试数据不一样，假如一个接口测试数据有100组，难道我们要写100个几乎相同的测试用例吗?

   那肯定不是。

   所以我们可以利用pytest的参数化方式，将测试数据进行统一管理，然后通过pytest的参数化装饰器传递给测试用例即可

   - 针对立即购买接口进行参数化

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-02-27 11:41
# @Copyright：北京码同学
import pytest

from requests_study.mtxshop_api import *
class TestBuyNowDataDriver:

    def setup_class(self):
        buyer_login()
    # 测试数据管理对象
    test_data = [
        ['必填参数均正确没有非必填字段',5173,   1,       200],
        ['产品已下架',               5875,   1,      500],
        ['产品不存在',              456665,  1,      500],
        ['购买数量超过库存',          5173, 99999999, 500],
        ['购买数量为0',              5173,  0,        400],
        ['购买数量为负数',            5173,   -1,     400],
        ['购买数量为空',              5173,  None,    400],
        ['产品id为空',               None,      1,    400]
    ]
```

```
 @pytest.mark.parametrize('casename,sku_id,num,expect_status_code',
test_data)
    def test_buy_now(self,casename,sku_id,num,expect_status_code):
        # 调用接口时传入测试数据中相关的变量
        resp = buy_now(sku_id=sku_id,num=num)
        # 做断言
        status_code = resp.status_code
        assert status_code == expect_status_code
```

- 笛卡尔积参数化方式

  在mtxshop_api.py中增加清空购物车的接口

```
def delete_cart():
    # 设置收货地址接口
    url = 'http://www.mtxshop.com:7002/trade/carts'
    global token
    headers = {
        'Authorization': token
    }

    resp = session.request(url=url,method='delete',
headers=headers)

    # status_code = resp.status_code  # 获取响应状态码
    # # print(status_code)
    # # resp_info = resp.text  # 获取响应信息，返回结果是字符串格式
    # resp_json = resp.json()  # 获取响应信息，返回结果是个字典，注意这种
方法只能针对响应信息是json格式
    # # print(resp_info)
    # print(resp_json)
    return resp
```

  使用笛卡尔积参数化方式，针对创建交易接口进行测试

```
class TestCreateTrade:
    def setup_class(self):
        buyer_login()

    client_data = ['PC','WAP','NATIVE','REACT','MINI'] # 5
    way_data = ['BUY_NOW','CART'] # 2
    expect_status_code = [200] # 1
    # 上面三组数据都作为参数，同时组合形成测试数据，组合结果总共5*2*1=10
    # 这种方式叫做 pytest笛卡尔积参数化
    # 笛卡尔积使用场景，在一个接口参数中存在多个参数具备有效值，并且他们需要组合
才能覆盖正向场景，就可以使用笛卡尔积
    @pytest.mark.parametrize('client',client_data)
    @pytest.mark.parametrize('way',way_data)
    @pytest.mark.parametrize('expect_status',expect_status_code)
    def test_create_trade(self,client,way,expect_status):
        # 对于创建交易这个来说，在调用之前需要先调用立即购买或者添加购物车接口
        # 因为创建交易会根据参数的不同去redis缓存中取订单相关的信息，然后创建
        # 当way参数是BUY_NOW会去缓存中读立即购买的数据
        # 当way参数是CART时会去缓存中读添加购物车的数据
        # 如何让缓存中产生创建交易接口需要的数据，就是调用对应的接口
        if way == 'BUY_NOW':
```

```
            buy_now()
        elif way == 'CART':
            delete_cart() #先清空购物车，防止多人使用或者购物车里有异常数据
            add_cart()
        resp = create_trade(client=client,way=way)
        status_code = resp.status_code
        print(resp.text)
        assert status_code == expect_status
```

- 立即购买接口参数化增加更多断言数据

```
class TestBuyNowDataDriver:

    def setup_class(self):
        buyer_login()
    # 测试数据管理对象
    test_data = [
        ['必填参数均正确没有非必填字段',5173,   1,       200,'',''],
        ['产品已下架',                5875,   1,      500,'004','产品已
下架'],
        ['产品不存在',               456665,  1,      500,'004','不合
法'],
        ['购买数量超过库存',           5173, 99999999, 500,'451','商品
库存已不足，不能购买。'],
        ['购买数量为0',               5173,  0,        400,'004','购买
数量必须大于0'],
        ['购买数量为负数',             5173,   -1,     400,'004','购买
数量必须大于0'],
        ['购买数量为空',               5173,  None,    400,'004','购买
数量不能为空'],
        ['产品id为空',                None,      1,    400,'004','产
品id不能为空']
    ]

 @pytest.mark.parametrize('casename,sku_id,num,expect_status_code,e
xpect_busi_code,expect_message',test_data)
    def
test_buy_now(self,casename,sku_id,num,expect_status_code,expect_bus
i_code,expect_message):
        # 调用接口时传入测试数据中相关的变量
        resp = buy_now(sku_id=sku_id,num=num)
        # 做断言
        status_code = resp.status_code
        assert status_code == expect_status_code
        print(resp.text) # 由于响应信息可能为空，所以我们使用resp.text打印
结果，因为空的字符串是无法使用resp.json()去获取的
        # 注意这个接口如果业务正常成功，那么响应信息是空的
        if status_code!=200:
            # 状态码不是200时，才进行响应信息的校验
            try:
                resp_json = resp.json()
            except:
                pass
            code = resp_json['code']
            assert code == expect_busi_code
            # 判断message的值是 商品已失效，请刷新购物车
            message = resp_json['message']
```

```
            assert message == expect_message
```

9. pytest多断言

   通常我们直接使用assert作为断言方式，但是注意assert断言一旦失败，代码不会继续向下执行

   当我们一个测试用例，需要有多组同级别的断言时，比如我们要断言响应信息里的多个字段值，

   一旦一个失败，剩下的就不会被判断，这不是我们希望的，我们希望能够将同级别的断言不管成功还是失败都执行到

   此时，需要借助pytest的一个插件pytest-assume，可以实现上述多断言都执行的情况

   - 安装插件

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

   - 使用多断言

     注意:

     敲代码时assume不会自动引用出来，需要手动敲

     pytest.assume必须使用pytest方式执行时才有效，否则会报错

     修改立即购买用例为多断言方式

     ```
     class TestBuyNowDataDriver:

         def setup_class(self):
             buyer_login()
         # 测试数据管理对象
         test_data = [
             ['必填参数均正确没有非必填字段',5173,   1,       200,'',''],
             ['产品已下架',               5875,   1,
      500,'004','产品已下架'],
             ['产品不存在',              456665,  1,
      500,'004','不合法'],
             ['购买数量超过库存',           5173, 99999999,
     500,'451','商品库存已不足，不能购买。'],
             ['购买数量为0',               5173,  0,
      400,'004','购买数量必须大于0'],
             ['购买数量为负数',             5173,   -1,
     400,'004','购买数量必须大于0'],
             ['购买数量为空',              5173,  None,
      400,'004','购买数量不能为空'],
             ['产品id为空',                None,      1,
      400,'004','产品id不能为空']
         ]

      @pytest.mark.parametrize('casename,sku_id,num,expect_status_co
     de,expect_busi_code,expect_message',test_data)
     ```

```
    def
test_buy_now(self,casename,sku_id,num,expect_status_code,expect
_busi_code,expect_message):
        # 调用接口时传入测试数据中相关的变量
        resp = buy_now(sku_id=sku_id,num=num)
        # 做断言
        status_code = resp.status_code
        assert status_code == expect_status_code
        print(resp.text) # 由于响应信息可能为空，所以我们使用
resp.text打印结果，因为空的字符串是无法使用resp.json()去获取的
        # 注意这个接口如果业务正常成功，那么响应信息是空的
        if status_code!=200:
            # 状态码不是200时，才进行响应信息的校验
            try:
                resp_json = resp.json()
            except:
                pass
            code = resp_json['code']
            # assert code == expect_busi_code
            pytest.assume(code == expect_busi_code)
            # 判断message的值是 商品已失效，请刷新购物车
            message = resp_json['message']
            # assert message == expect_message
            pytest.assume(message == expect_message)
```

10. conftest.py

    该文件是一个特殊文件，是pytest在执行时会自动扫描该文件中的钩子函数并且按照一定的规则执行

    第一个解决参数化时用例名称是中文时的乱码问题

    创建一个conftest.py文件，增加如下代码

```
def pytest_collection_modifyitems(
        session: "Session", config: "Config", items: List["Item"]
) -> None:
    # item表示每个测试用例，解决用例名称中文显示问题
    for item in items:
        item.name = item.name.encode("utf-8").decode("unicode-
escape")
        item._nodeid = item._nodeid.encode("utf-
8").decode("unicode-escape")
```

11. pytest fixture函数

    在conftest.py中

```
# 使用@pytest.fixture装饰的方法是一个fixture方法
# 装饰器有两个参数，一个是scope,一个是autouse
# scope表示该方法的作用域，autouse该方法是否自动调用
# scope 作用域总共有session、package、class、function、module
# session表示的是本次pytest执行中该方法只会被执行一次
# package和session几乎相同
# class 在同一个测试类中不管被调用多少次，都只执行一次
# function 每一个测试用例执行时都会被调用
# module 在同一个模块下，不管这个方法被调用多少次，都只执行一次
@pytest.fixture(scope='session',autouse=True)
```

```
def get_token():
    print('fixture里调用了登录')
    buyer_login()
@pytest.fixture(scope='package',autouse=True)
def pa1():
    print('fixture里package')
@pytest.fixture(scope='class',autouse=True)
def pa2():
    print('fixture里class')
@pytest.fixture(scope='function',autouse=True)
def pa3():
    print('fixture里function')
@pytest.fixture(scope='module',autouse=True)
def pa4():
    print('fixture里module')

# 该fixture 的autouse没有写，默认是False，所以他不会被自动调用
# 因此想使用时，需要手动调用
# 调用时可以采用@pytest.mark.usefixtures('ceshilei')方式
# 也可以采用在测试用例函数中传参，参数名称就是该fixture函数名称
@pytest.fixture(scope='class')
def ceshilei():
    print('fixture里ceshilei')
```

上面所写的所有的fixture方法其实都是前置动作，就是在测试用例或者测试类执行之前去执行的

如果我想使用fixture实现后置动作如何来做呢

```
@pytest.fixture(scope='class')
def qianzhi_and_houzhi():
    print('这是前置动作')
    yield # 从这一行往后是后置动作，在这之前是前置动作
    print('这是后置动作')
```

希望fixture方法能够返回相关的数据

```
@pytest.fixture(scope='class')
def return_data():
    print('这是计算数据的过程')
    yield '这是fixture的返回值'
    print('这是return_data的后置动作')
```

13. pytest总结

    pytest编写测试用例的规则以及方式

    pytest前置后置动作

    pytest参数化

    conftest.py

    pytest 的fixture，可以充当前置后置动作

3.redis在测试中的应用

在之前接口的断言中我们完成了响应状态码以及响应信息的断言处理，但是我们没有完成数据的断言，指的是接口所影响到数据是否正确，这个数据可能是存在redis、数据库、mq等等存储介质中

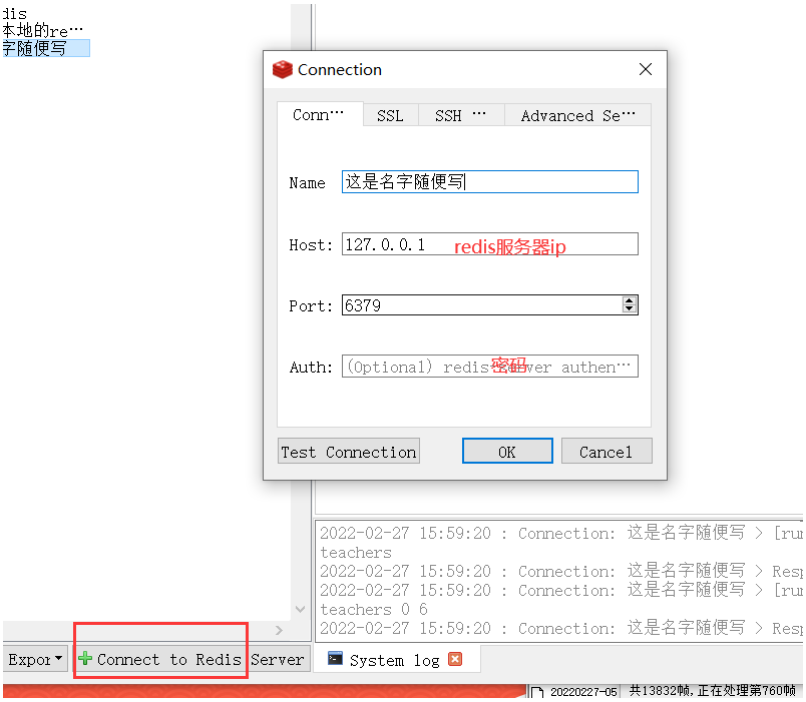
针对立即购买接口实现redis数据断言

1. windows redis启动问题

   如果启动不了，可以在命令行进入到redis所在的目录，执行如下命令

   ```
   redis-server redis.windows.conf
   ```

2. 使用客户端工具连接redis



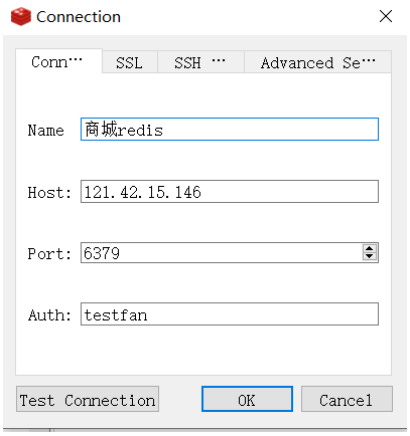
3. 使用python脚本来实现redis数据操作

   - 安装redis第三方库

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

4. 使用redis客户端工具连接商城redis

   ip: 121.42.15.146 密码：testfan



5. 安装javaobj-py3

```
# windows:
pip install javaobj-py3 -i https://pypi.douban.com/simple/
# mac
python3 -m pip install javaobj-py3 -i https://pypi.douban.com/simple/
```

6. 封装redis操作类

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-02-27 16:35
# @Copyright：北京码同学
import javaobj
import redis


class RedisUtil:
    # decode_responses设置为true以字符串形式得到数据，设置为False以二进制形式得到
    def __init__(self,host,pwd,port=6379,decode_responses=False):
        self.pool = redis.ConnectionPool(host=host,password=pwd,port=port, decode_responses=decode_responses,encoding_errors='ignore')
        self.r = redis.Redis(connection_pool=self.pool)  # 表示从上面的连接池拿到一个连接对象
    def get(self,key):
        type = self.r.type(key).decode('utf8')
        if type == 'string':
            return self.r.get(key)
        elif type == 'hash':
            return  self.r.hgetall(key)
        elif type == 'zset':
            return self.r.zrange(key,0,-1)
        elif type == 'set':
            return self.r.smembers(key)
        elif type == 'list':
            return self.r.lrange(key,0,-1)
        else:
            raise Exception(f'不支持的数据类型{type}或者{key}不存在')
if __name__ == '__main__':
    redis_util = RedisUtil(host='121.42.15.146',pwd='testfan')
    res = redis_util.get('{BUY_NOW_ORIGIN_DATA_PREFIX}_59') # 这个key后面的数字是根据用户发生变化，是用户id
    print(res)
    # 商城项目后台开发语言是java，存redis数据时，存的是java对象
    # 立即购买的对象是一个List集合，里边放的是CartSkuOriginVo对象，这些都是java里的
    # 无法直接解析，因此要使用一个第三方库，javaobj-py3，这个库可以帮我们将java类型的序列化数据转换成python对象
    res_list = javaobj.loads(res) # 这是得到结果转换成python的列表
    print(res_list)
    # 业务上这个列表中，只会存储一个数据
    obj = res_list[0] #通过索引得到列表里的数据
    # 立即购买存储的实际上是产品订单的一些信息，这个对象必然具备非常多的属性
    print(dir(obj)) # 可以使用这个方法来看obj对象都有哪些属性，但最佳的方式还是直接问比较好
    # print(obj.__getattribute__('price'))
```

```
    print(obj.__getattribute__('num'))
    print(obj.__getattribute__('skuId'))
```

7. 在测试用例中引用redis

我们这个项目使用redis的接口是比较多的，所以我们要在测试之前拿到一个redis操作对象，这个时候可以利用pytest什么东东来解决?

使用前置动作来完成，因为有很多测试文件都要用，所以我们可以使用基于session级别的fixture来完成redis操作对象的创建

在conftest.py中增加下列方法

```
@pytest.fixture(scope='session',autouse=True)
def redis_util():
    redis_util = RedisUtil(host='121.42.15.146',pwd='testfan')
    yield redis_util
```

由于立即购买的缓存key值中有动态的用户id，所以我们提取一下

先修改buyer_login接口增加return resp代码

再修改conftest.py

```
@pytest.fixture(scope='session',autouse=True)
def get_token():
    print('fixture里调用了登录')
    resp = buyer_login()
    uid = resp.json()['uid']
    yield uid
```

修改测试用例

```
class TestBuyNowDataDriver:

    # def setup_class(self):
    #     buyer_login()
    # 测试数据管理对象
    test_data = [
        ['必填参数均正确没有非必填字段',5173,   1,       200,'',''],
        ['产品已下架',                5875,   1,      500,'004','产品已下
架'],
        ['产品不存在',              456665,  1,      500,'004','不合法'],
        ['购买数量超过库存',          5173, 99999999, 500,'451','商品库存已
不足，不能购买。'],
        ['购买数量为0',               5173,  0,        400,'004','购买数量
必须大于0'],
        ['购买数量为负数',            5173,   -1,     400,'004','购买数量
必须大于0'],
        ['购买数量为空',              5173,  None,    400,'004','购买数量
不能为空'],
        ['产品id为空',                None,      1,    400,'004','产品id不
能为空']
    ]
    @pytest.mark.usefixtures('ceshilei') # 这也是一种调用fixture方法的方式，
参数是fixture方法的名字
```

```
 @pytest.mark.parametrize('casename,sku_id,num,expect_status_code,expec
t_busi_code,expect_message',test_data)
    def
test_buy_now(self,casename,sku_id,num,expect_status_code,expect_busi_co
de,expect_message,redis_util,get_token):
        # 调用接口时传入测试数据中相关的变量
        resp = buy_now(sku_id=sku_id,num=num)
        # 做断言
        status_code = resp.status_code
        assert status_code == expect_status_code
        print(resp.text) # 由于响应信息可能为空，所以我们使用resp.text打印结
果，因为空的字符串是无法使用resp.json()去获取的
        # 注意这个接口如果业务正常成功，那么响应信息是空的
        if status_code!=200:
            # 状态码不是200时，才进行响应信息的校验
            try:
                resp_json = resp.json()
            except:
                pass
            code = resp_json['code']
            # assert code == expect_busi_code
            pytest.assume(code == expect_busi_code)
            # 判断message的值是 商品已失效，请刷新购物车
            message = resp_json['message']
            # assert message == expect_message
            pytest.assume(message == expect_message)
        if status_code == 200:
            # 从redis中获取数据并做校验
            uid = get_token #get_token是个fixture，调用时他就相当于他的返回
值，我们返回了uid
            # 格式化字符串时，如果字符串本身就有大括号，那么需要使用双大括号
            res =
redis_util.get(f'{{BUY_NOW_ORIGIN_DATA_PREFIX}}_{uid}')
            res_list = javaobj.loads(res)  # 这是得到结果转换成python的列表
            # 业务上这个列表中，只会存储一个数据
            obj = res_list[0]  # 通过索引得到列表里的数据
            redis_num = obj.__getattribute__('num') # 实际结果
            redis_skuId = obj.__getattribute__('skuId') # 实际结果
            pytest.assume(redis_num == num)
            pytest.assume(redis_skuId == sku_id)
```