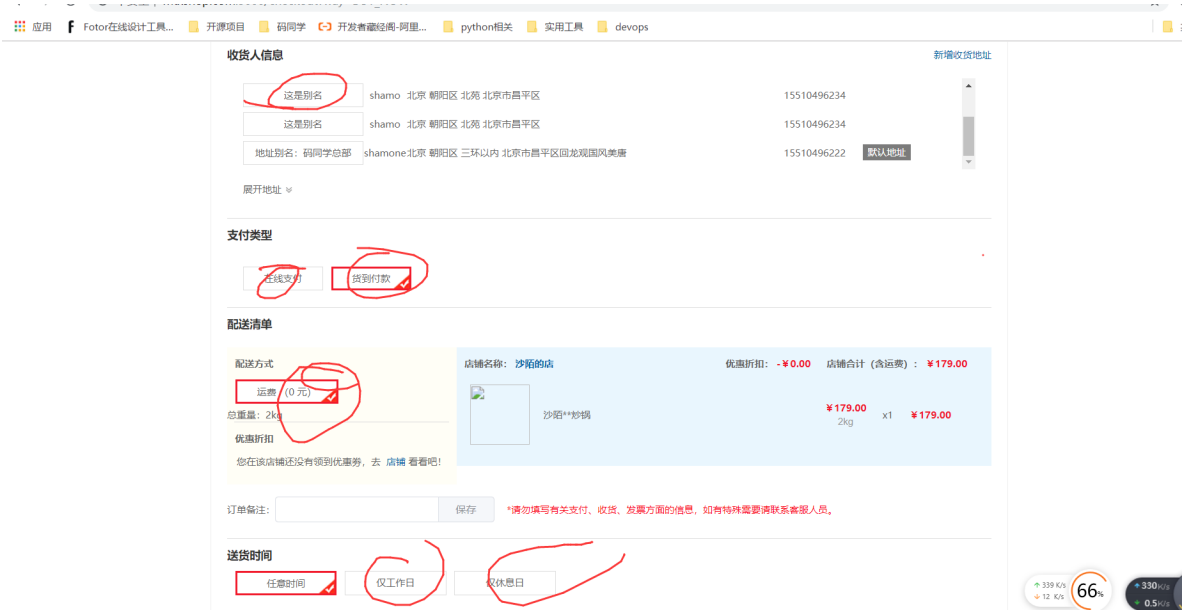
0306课堂笔记

1.设置收货地址接口的缓存处理

下单前页面需要设置订单的相关信息，这些信息都会被存入redis缓存中



一个测试类

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-03-06 9:54
# @Copyright：北京码同学
import javaobj
import pytest

from requests_study.mtxshop_api import set_address, set_payment_type


class TestSetAddress:


    def test_set_address(self,redis_util,get_token):
        # 调用接口拿到返回
        resp = set_address()
        # 做断言
        status_code = resp.status_code
        assert status_code == 200

        resp = set_payment_type()
        status_code = resp.status_code
        assert status_code == 200
        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
        res = redis_util.get(f'{{CHECKOUT_PARAM_ID_PREFIX}}_{uid}')
        # res是个字典
        for key, value in res.items():
```

```
            key = javaobj.loads(key)
            # print(key)
            try:
                value = javaobj.loads(value)
                if key == 'addressId':
                    # print(value)
                    pytest.assume(value==3896)
                if key == 'paymentType':
                    value = value.__getattribute__('constant')
                    pytest.assume(value=='COD')
            except:
                pass
```

2.数据库断言

- 安装第三方库

  因为数据库是mysql的，所以先安装pymysql

  ```
  #windows
  pip install PyMySQL
  # mac
  python3 -m pip install PyMySQL
  ```

- pymysql基本操作

  ```
  # !/usr/bin python3
  # encoding: utf-8 -*-
  # @author: 沙陌 微信：Matongxue_2
  # @Time: 2022-03-06 10:26
  # @Copyright：北京码同学
  import pymysql

  connect = pymysql.Connect(
      host='121.42.15.146',
      port=3306,
      user='root',
      password='Testfan#123',
      charset='utf8mb4',
      cursorclass=pymysql.cursors.DictCursor
  )

  # SELECT * FROM mtxshop_trade.es_order WHERE member_id=59 ORDER BY order_id
  DESC LIMIT 1 ;
  cursor = connect.cursor() #获取一个游标对象
  # 通过游标对象去操作
  cursor.execute("SELECT * FROM mtxshop_trade.es_order WHERE member_id=59
  ORDER BY order_id DESC LIMIT 1 ;")
  data = cursor.fetchall() #拿到执行之后的结果
  cursor.close() # 关闭游标对象
  connect.commit() # 提交数据
  connect.close() # 关闭连接对象

  print(data)
  # data是一个列表，里边套着无数个字典，一个字典就是一行数据
  # 拿到第一行数据
  print(data[0])
  ```

```
# 拿到第一行数据里的trade_sn
print(data[0]['trade_sn'])
```

- 数据库操作封装

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-03-06 10:38
# @Copyright：北京码同学
import time

import pymysql


class DB_Util:

    def __init__(self,host,user,password,port=3306):
        self.connect = pymysql.Connect(
            host=host,
            user=user,
            password=password,
            port=port,
            charset='utf8mb4',
            cursorclass=pymysql.cursors.DictCursor
        )


    def select(self,sql):
        cursor = self.connect.cursor() # 相当于拿到了数据库数据集
        cursor.execute(sql)
        data = cursor.fetchall()
        self.connect.commit() # 如果不提交数据，那么如果数据有更新，那么下次查询还是旧数据
        cursor.close()
        return data

    def update(self,sql):
        cursor = self.connect.cursor()
        cursor.execute(sql)
        self.connect.commit()
        cursor.close()

    def close(self):
        if self.connect!=None:
            self.connect.close()
if __name__ == '__main__':
    db_util = DB_Util(host='121.42.15.146',user='root',password='Testfan#123')
    for i in range(10):
        time.sleep(10)
        data = db_util.select('SELECT * FROM mtxshop_trade.es_order WHERE member_id=59 ORDER BY order_id DESC LIMIT 1 ;')
        print(data[0])
    db_util.close()
```

- 在conftest.py中增加一个fixture

  这个fixture主要适用于数据库连接对象的创建以及关闭，整个测试过程中，只需要连接一次数据库即可

```
@pytest.fixture(scope='session',autouse=True)
def db_util():
    db_util = DB_Util(host='121.42.15.146', user='root',
password='Testfan#123')
    yield db_util
    db_util.close()
```

- 在测试用例中做断言

```
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,db_util):
        # 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

        # 判断数据数据库数据是否正确
        # 数据库里是实际值，那么期望数据是谁
        # 期望数据可以是接口发起时的接口数据，比如产品id，下单数量
        # 期望数据还可以是接口响应数据里的内容
        data = db_util.select('SELECT * FROM mtxshop_trade.es_order WHERE
member_id=59 ORDER BY order_id DESC LIMIT 1 ;')
        db_res = data[0] # 第一行数据，这是个字典
        db_trade_sn = db_res['trade_sn']
        items_json = db_res['items_json']
        # print(type(items_json))  # 由于items_json拿到之后是个字符串，不好从中提
取内容
        # 所以我们可以将字符串转换列表套字典的形式
```

```
        # items_json = eval(items_json) # 这种不行，因为字符串有null，python里是None
        # print(type(items_json))
        items_json = json.loads(items_json)  # 使用json库需要先导入import json
        # print(type(items_json))
        # print(items_json)
        # print(items_json[0])
        # print(items_json[0]['sku_id'])
        # print(items_json[0]['num'])
        db_sku_id = items_json[0]['sku_id']
        db_num = items_json[0]['num']

        # 获取响应信息中的期望值
        resp_json = resp.json()
        resp_trade_sn = resp_json['trade_sn']
        pytest.assume(db_trade_sn == resp_trade_sn) #断言响应数据和数据库中trade_sn是否一致
        pytest.assume(db_sku_id == 5173)
        pytest.assume(db_num == 1)
```

3.jsonpath在测试中的应用

由于我们提取接口响应数据时，层级结构比较深的情况，提取是不太方便的，我们可以jsonpath表达式帮我们简化处理

- jsonpath基本规则

  在线调试地址： http://www.e123456.com/aaaphp/online/jsonpath/

  规则： https://goessner.net/articles/JsonPath/index.html#e2

- 脚本中使用

```
#windows
pip install jsonpath
# mac
python3 -m pip install jsonpath
```

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-03-06 13:32
# @Copyright：北京码同学
import jsonpath
```

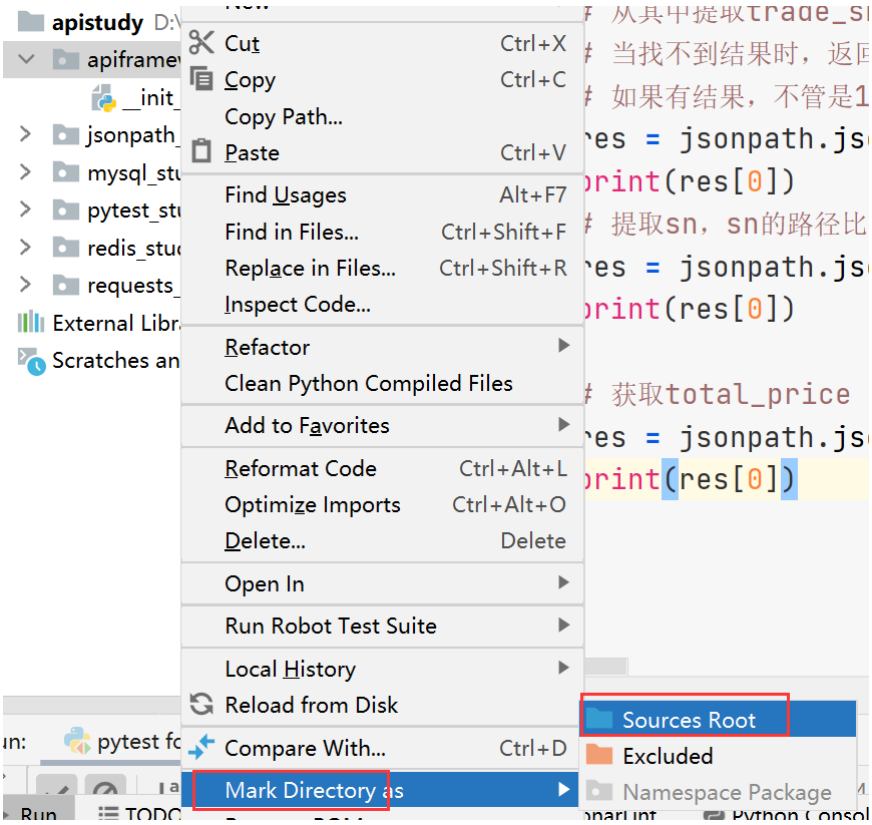
```
s =
{"trade_sn":"20220306000052","member_id":59,"member_name":"shamo","payment_type":"COD","price_detail":
{"total_price":179.0,"original_price":179.0,"goods_price":179.0,"freight_price":0.0,"discount_price":0.0,"cash_back":0.0,"coupon_price":0.0,"full_minus":0.0,"is_free_freight":0,"exchange_point":0},"consignee":
{"consignee_id":3716,"name":"金枝","province":"北京","city":"朝阳区","county":"三环以内","town":None,"address":"北京","mobile":"18888888888","telephone":None,"province_id":1,"county_id":2799,"city_id":72,"town_id":0},"coupon_list":None,"order_list":
[{"seller_id":20,"shipping_type_id":None,"shipping_type_name":None,"seller_name":"沙陌的店","weight":2.0,"price":
{"total_price":179.0,"original_price":179.0,"goods_price":179.0,"freight_price":0.0,"discount_price":0.0,"cash_back":0.0,"coupon_price":0.0,"full_minus":0.0,"is_free_freight":0,"exchange_point":0},"sku_list":
[{"seller_id":20,"seller_name":"沙陌的店","goods_id":4942,"sku_id":5173,"sku_sn":"shamo001","cat_id":83,"num":1,"purchase_num":0,"goods_weight":2.0,"original_price":179.0,"purchase_price":179.0,"subtotal":179.0,"name":"沙陌**炒锅","goods_image":"http://www.mtxshop.com:7000/statics/attachment/goods/2021/12/13/21/50442449.png_300x300.png","template_script":None,"checked":1,"is_free_freight":1,"single_list":[],"group_list":[],"promotion_tags":
[],"not_join_promotion":0,"template_id":0,"spec_list":None,"point":None,"snapshot_id":None,"service_status":"NOT_APPLY","last_modify":1640517899,"enable_quantity":9998714,"rule":None,"invalid":0,"error_message":"","is_ship":1,"goods_type":"NORMAL"}],"gift_list":[],"gift_coupon_list":
[],"gift_point":None,"invalid":None,"trade_sn":"20220306000052","sn":"20220306000052","consignee":{"consignee_id":3716,"name":"金枝","province":"北京","city":"朝阳区","county":"三环以内","town":None,"address":"北京","mobile":"18888888888","telephone":None,"province_id":1,"county_id":2799,"city_id":72,"town_id":0},"shipping_id":0,"payment_type":"COD","ship_time":None,"receive_time":"任意时间","member_id":59,"member_name":"shamo","remark":"","create_time":1646538406,"shipping_type":None,"order_status":"NEW","pay_status":"PAY_NO","ship_status":"SHIP_NO","ship_name":None,"order_price":179.0,"shipping_price":None,"comment_status":"UNFINISHED","disabled":None,"payment_method_id":None,"payment_plugin_id":None,"payment_method_name":None,"payment_account":None,"goods_num":1,"warehouse_id":None,"cancel_reason":None,"ship_province_id":None,"ship_city_id":None,"ship_region_id":None,"ship_town_id":None,"ship_province":None,"ship_city":None,"ship_region":None,"ship_town":None,"signing_time":None,"the_sign":None,"admin_remark":None,"address_id":None,"need_pay_money":179.0,"ship_no":None,"logi_id":None,"logi_name":None,"need_receipt":0,"receipt_title":None,"receipt_content":None,"service_status":"NOT_APPLY","client_type":"PC","receipt_history":None,"order_type":"NORMAL","order_data":None,"goods_coupon_prices":None}],"gift_list":None}

# 从其中提取trade_sn
# 当找不到结果时，返回的是False
# 如果有结果，不管是1个还是多个，那么返回的数据类型是列表
res = jsonpath.jsonpath(s,'$.trade_sn')
print(res[0])
# 提取sn，sn的路径比较深
res = jsonpath.jsonpath(s,'$..sn')
print(res[0])

# 获取total_price
res = jsonpath.jsonpath(s,'$.price_detail.total_price')
print(res[0])
```

4.搭建接口框架结构

创建一个apiframework的包，将其设置为项目根目录



依次创建下面的结构

- common：这是一个package，主要用来存放接口测试过程中需要用到的底层公共方法，比如数据库封装、日志封装、redis封装、文件读写、requests封装、随机数据生成、加解密封装等等
- api：这是一个package，封装各个接口服务的基类，以及单接口的基本定义
- testcases：这是一个package，主要是放基于pytest的测试用例脚本
- config：这是一个目录，主要是配置文件，比如域名配置、redis配置、数据配置，公共基础数据配置
- data：这是一个目录，主要放测试数据文件
- logs：这是一个目录，用来存放收集到的日志文件
- report：这是一个目录，存储测试结果数据以及测试报告的
- conftest.py: 这个文件是用来编写pytest的一些自带的钩子函数的，并且统一在这里完成自定义的fixture编写
- pytest.ini：这个文件是pytest执行时的参数配置文件，在这里可以指定pytest执行的命令行参数，并且还可以修改pytest识别用例的默认规则

```
[pytest]
;addopts 指的是pytest在执行时的参数
addopts = -sv
;testpaths 你要执行的测试脚本的目录
testpaths = ./testcases
;python_files 识别测试文件的规则
python_files = test_*.py
;测试类的识别规则
python_classes = Test*
;测试用例的识别规则
python_functions = test_*
```

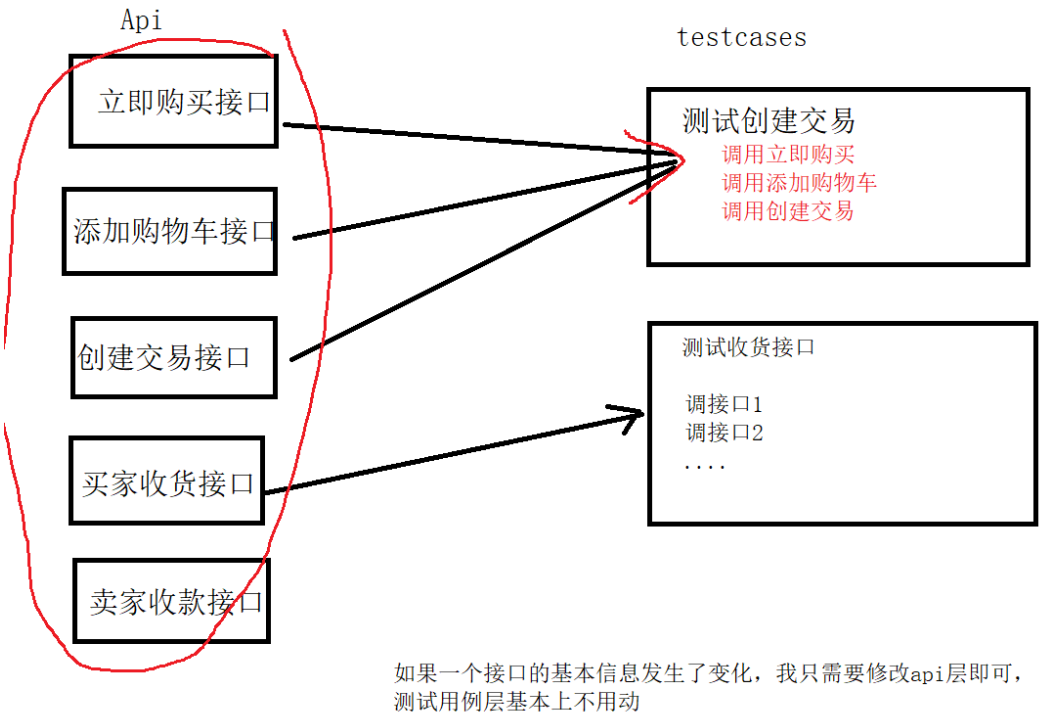
- run.py: 这是框架统一的执行入口

```
import pytest

if __name__ == '__main__':
    # 执行时，会自动识别pytest.ini中的规则，完成执行
    pytest.main()
```

5.apiobject模式

这是一种封装接口框架的模式，把每一个接口当成单独的对象来进行管理，在测试用例中通过调用接口对象来实现测试用例的组装



把每个接口的基本信息定义成一个类

- 在api目录下创建package

  通过一定的维度去管理自己的接口定义，咱们是一个微服务的项目，有买家服务、卖家服务、管理员服务、基础服务，所以在api目录下创建4个package，分别是buyer、seller、manager、basic

- 单接口定义
- 抽取基类
- 抽取requests库的封装

- 封装更多的买家接口
- 实现添加购物车的测试用例

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-03-06 15:57
# @Copyright：北京码同学
import pytest

from api.buyer.cart import AddCartApi


class TestAddCart:

    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,expect_bus
i_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，传入测试数据
        add_cart = AddCartApi(sku_id=sku_id,num=num)
        resp = add_cart.send()
        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)
```

6.框架集成日志

为了能够更加清晰的看到接口执行过程中的各种信息，我们需要收集到接口的各个信息

日志类以及setting.py文件

setting.py是采用获取该文件所在的目录来得到当前项目路径的，可以帮我们在处理文件操作时进行拼接文件路径，达到一个动态的文件路径效果

接口自动化分析问题时，核心的信息是接口的各种信息(url/headers/参数/响应)，通过这些信息去分析我们的接口为什么失败

日志收集在RequestsClient这个类中

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-03-06 15:21
# @Copyright：北京码同学
import requests

from common.logger import GetLogger


class RequestsClient:
    session = requests.session() # 统一的一个session对象，可以帮我们自动关联cookie
    def __init__(self):
        self.logger = GetLogger.get_logger()
        self.session = RequestsClient.session
        self.url = None
        self.method = None
        self.headers = None
        self.params = None
        self.data = None
        self.json = None
        self.files = None
        self.resp = None
    def send(self,**kwargs):
        # 抽取时，每个接口的request请求不一定会发送多少个参数
        # 可以使用不定长关键字参数传递方式
        # 判断一下，调用send时，如果有些参数没有传，那么就用对象自身的
        if 'url' not in kwargs:
            kwargs['url'] = self.url
        if 'method' not in kwargs:
            kwargs['method'] = self.method
        if 'headers' not in kwargs:
            kwargs['headers'] = self.headers
        if 'params' not in kwargs:
            kwargs['params'] = self.params
        if 'data' not in kwargs:
            kwargs['data'] = self.data
        if 'json' not in kwargs:
            kwargs['json'] = self.json
        if 'files' not in kwargs:
            kwargs['files'] = self.files
```

```
        # 遍历kwargs,收集接口请求的所有信息
        for key,value in kwargs.items():
            self.logger.debug(f'接口的{key}是:{value}')
        try:
            self.resp = self.session.request(**kwargs)
            self.logger.debug(f'接口的响应状态码是:{self.resp.status_code}')
            self.logger.debug(f'接口的响应是:{self.resp.text}')
        except:
            self.logger.exception('接口发起异常')
        return self.resp
```

7.集成allure测试报告

- 安装allure-pytest插件

```
# windows
pip install allure-pytest
# mac
python3 -m pip install allure-pytest
```

- 在pytest.ini中增加参数

  修改pytest.ini的addopts为

```
addopts = -sv  --alluredir ./report/data --clean-alluredir
```

  --alluredir ./report/data 指的是指定测试结果数据存储的目录

  --clean-alluredir 指的是每次执行都清除掉原来的结果数据

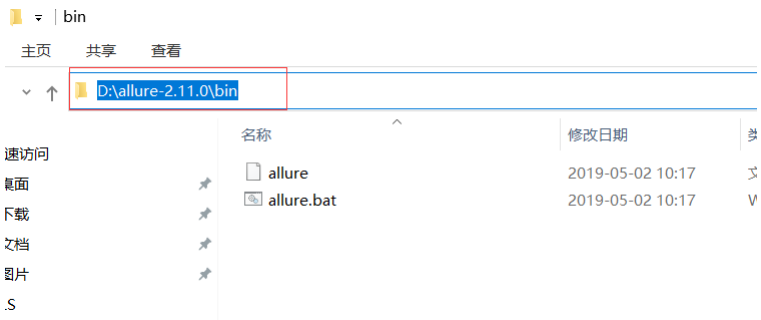
- 配allure命令行工具

  前面两步只是收集到了测试结果数据，并没有将其变成可视化的一个报告，因此我们需要借助allure的命令行工具来读取测试结果数据，生成html报告

  1. 安装jdk，参考群文件java环境安装
  2. 配置allure命令行

     下载群文件的allure-2.11.0.zip，解压即可

     然后配置环境变量，将下述路径追加到环境变量path中去，然后重启pycharm



     命令行里输入allure --version能够看到版本就说明配好了

     如果你是mac，可以直接使用brew install allure进行安装

  3. 修改run.py如下：

```
# !/usr/bin python3
# encoding: utf-8 -*-
# @author: 沙陌 微信：Matongxue_2
# @Time: 2022-03-06 14:13
# @Copyright：北京码同学
import os

import pytest

if __name__ == '__main__':
    # 执行时，会自动识别pytest.ini中的规则，完成执行
    pytest.main() # pytest -sv  --alluredir ./report/data --clean-
alluredir testcases
    os.system('allure generate ./report/data -o ./report/html --clean')
```

./report/data 指的是你测试结果数据存放的目录

./report/html 生成的html报告所在的目录

- 运行run.py

  在项目的左侧结构中，打开report下的html下的index.html，在pycharm选择浏览器打开