全程66期接口自动化第六次

回顾：

1 熟悉用例表 （独立接口+上下游接口 ---上游 relation 字段 下游 ${}）

2架构 指导实现剧情

2.1 封装了 PymysqlAPI 工具类

|  |
| --- |
| **工具名字 类名 MysqlUtil**  **功能1 get\_fetchone 获取单条数据**  **功能2 get\_fetchall 获取多条数据**  **功能3 sql\_execute 更新类sql** |

2.2 caseutil 用例工具类-----考虑 工具类 复用

|  |
| --- |
| **1.1 加载所有用例 ---完成**  **1.2 加载需要执行用例-----进度**  **1.3 加载用例配置**  **1.4 回写存放用例结果** |

### 内容1： 用例工具类

|  |
| --- |
| #其他工具类  import datetime  from utils.mysqlutil import MysqlUtil  gongju1=MysqlUtil() # gongju1 封装pymysql的API  #我是用例工具类  # 考虑 通用尽可能复用  class RdTestCase:  # 1.1加载所有用例--->复用性更高～  def load\_all\_case(self,jwweb):  # 需求： 为了复用 应该在使用工具传递测试项目  # 解决： 形式参数  sql=f"select \* from jwtest\_case\_list where web='{jwweb}'"  return gongju1.get\_fetchall(sql)  # 1.2加载需要执行用例 标准 isdel字段 是否为1  # 方式1 sql实现 select XXX where web='{jwweb}' and isdel=‘1’ “  # 方式2 Python实现 已经能获取所有数据了～（上课 加强Python+语法：列表推导式）  # 根据字典的isdel 键进行筛选 存放到指定的 列表 即可  def is\_run\_data(self,jwweb):  # 方式1：可读性 +后期调试  # #0 准备空列表 等待存放需要被执行的用例～  # jw\_run\_list=[]  # # 1 获取所有用例self.load\_all\_case()  # all\_case=self.load\_all\_case(jwweb)  # # 2 for 取出所有用例  # for case in all\_case:  # # 3 判断每条用例isdel字段  # if case['isdel']==1:  # # 4 isdel 为1用例存放到列表中即可～  # jw\_run\_list.append(case)  # return jw\_run\_list # 存放需要执行的用例～  # python 语法===>列表推导式  # [3结果变量 1 for 2 if]  # 方式2 精简 可读性 差  return [case for case in self.load\_all\_case(jwweb) if case['isdel']==1]  # 1.3加载用例配置  # sql方式实现 给定 XXXX项目 的 XXXX值即可  def loadConfKey(self,jwweb,jwkey):  jwsql=f"select \* from jwtest\_config where web='{jwweb}' and key1='{jwkey}'"  return gongju1.get\_fetchone(jwsql)  # 1.4回写存放用例结果 实际结果 测试结果 用例编号  def updateResults(self,response,is\_pass,case\_id):  # 1times--当前时间  current\_time=datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')  # 2 insert 语句---<练习 补全这个sql语句～  jwsql=f'insert into jwtest\_result\_record(case\_id,times,response,result) ' \  f'values ("{case\_id}","{current\_time}","{str(response)}","{str(is\_pass)}")'  #3 执行完 用例结果回写完毕了  return gongju1.sql\_execute(jwsql)  if \_\_name\_\_ == '\_\_main\_\_':  gongju2=RdTestCase()  print("自测第一个",gongju2.load\_all\_case('okr-api')) #  print("自测第二个",gongju2.is\_run\_data('okr-api'))  print("自测第三个",gongju2.loadConfKey('atstudy\_okr','url\_api')['value'])  # 自测第四个方法  jia\_shiji={'success':True,'message':'登录成功'}  jia\_pass=True  jia\_caseid='666'  print("自测第四个方法",gongju2.updateResults(jia\_shiji,jia\_pass,jia\_caseid)) |

小结:--恭喜工具2

|  |
| --- |
| **工具名字 RdTestCase**  **功能1： .load\_all\_case 加载所有用例**  **功能2 : .is\_run\_data 加载 需要执行的用例**  **功能3 loadConfKey 获取接口服务器配置**  **功能4 updateResults 支持用例结果回写** |

### 内容2： 项目配置项 配置文件:

需求： 希望 工具类 代码 尽量不变（复用），但是 项目容易变化东西~影响工具类~

解决： 容易变化东西 ----》配置 单独存放 + 和Python代码分离~

价值： 如果配置 变更 只要修改配置文件即可 增强 工具类复用性~

配置 文件

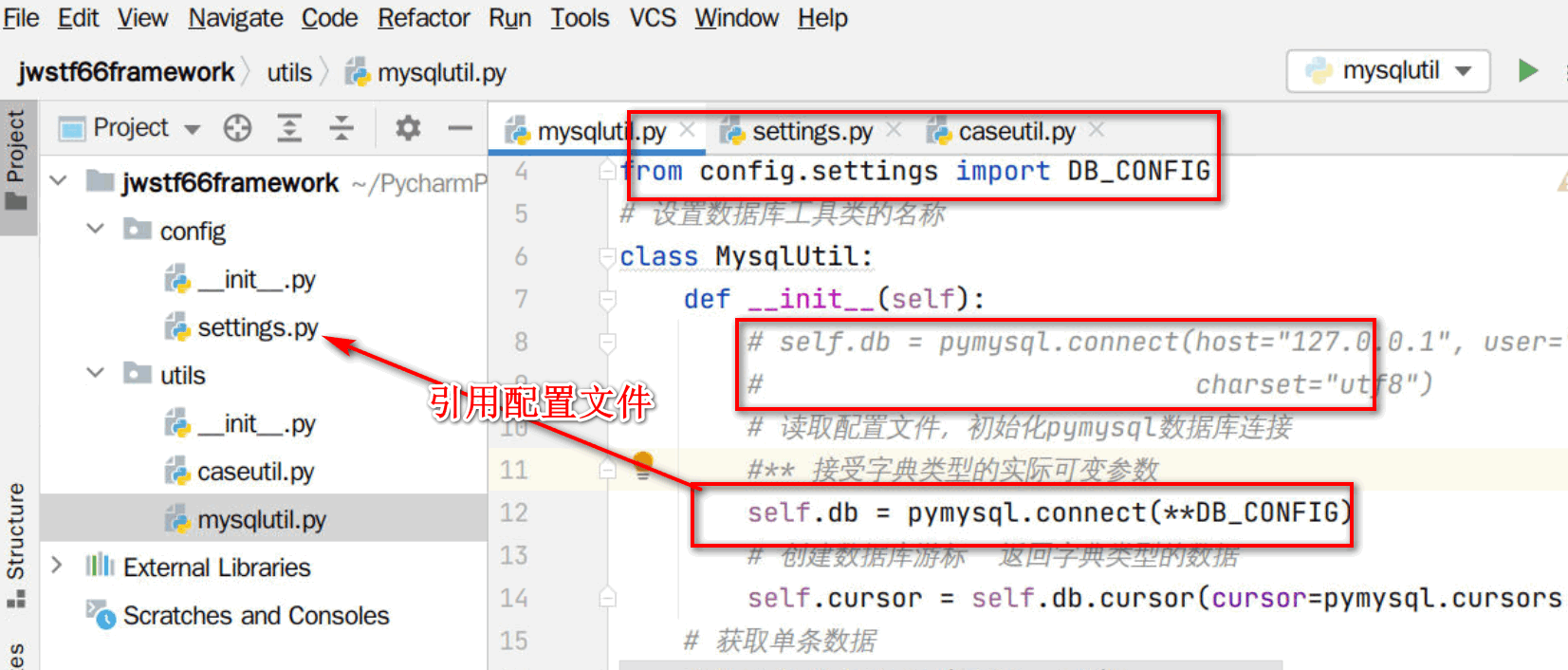
1非python

ini ----Selenium项目自动化框架 yml ----appium 小框架 优势 分离彻底 缺点 需要额外工具代码 加载

2 py 配置文件 ---接口自动化框架 优势： 导包调用 简单 缺点 不够彻底

产生配置文件

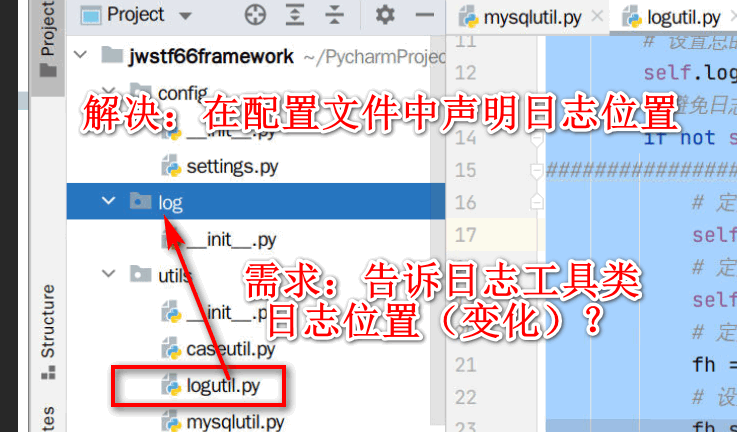
|  |
| --- |
| # 数据库信息配置  DB\_CONFIG = {"host": "127.0.0.1",  "user": "root",  "password": "123456",  "database": "jwtest1",  "port": 3306,  "charset": "utf8"} |



### 内容3： 整合日志系统（工具类）：

|  |
| --- |
| from config.settings import get\_log\_path  import logging  import time  import os  class LogUtil:  def \_\_init\_\_(self):  # 初始化日志对象，设置日志名称  self.logger = logging.getLogger("jwlogger")  # 设置总的日志级别开关  self.logger.setLevel(logging.DEBUG)  # 避免日志重复  if not self.logger.handlers:  ########################1.设置在文件中输出###################  # 定义日志名称  self.log\_name = '{}.log'.format(time.strftime("%Y\_%m\_%d", time.localtime()))  # 定义日志路径及文件名称  self.log\_path\_file = os.path.join(get\_log\_path(), self.log\_name)  # 定义文件处理handler  fh = logging.FileHandler(self.log\_path\_file, encoding='utf-8', mode='a')  # 设置文件处理handler的日志级别  fh.setLevel(logging.DEBUG)  # 日志格式变量  formatter = logging.Formatter("%(asctime)s - %(filename)s[line:%(lineno)d] - %(levelname)s: %(message)s")  # 设置打印格式  fh.setFormatter(formatter)  # 添加handler  self.logger.addHandler(fh)  # 关闭handler  fh.close()  ##################################2.设置在控制台输出########  # 定义控制台输出流handler  fh\_stream = logging.StreamHandler()  # 控制台输出日志级别  fh\_stream.setLevel(logging.DEBUG)  # 设置打印格式  fh\_stream.setFormatter(formatter)  # 添加handler  self.logger.addHandler(fh\_stream)  def log(self):  # 返回定义好的logger对象，对外直接使用log函数即可  return self.logger  # 通过logger 对外输出日志信息～  logger = LogUtil().log()  # 测试代码  if \_\_name\_\_ == '\_\_main\_\_':  logger.info('jwtest11')  logger.info('jwtest22')  logger.info('jwtest33') |

需求：



配置文件

|  |
| --- |
| # 配置2：  project\_path=os.path.dirname(os.path.dirname(\_\_file\_\_))  # 声明配置文件位置  log\_path=project\_path+os.sep+"log"  def get\_log\_path():  return log\_path |

### 内容4： requests 工具类---思路 ：

发get 发post？ ----》用例决定

|  |
| --- |
| 工具思路:  1def api\_run(来自于数据库用例 url , method , data请求体, headers , cookies)：  请求：  1.1 if如果 method==get:  res= 调用 requests get API  1.2 if 如果 method==post:  1.2.1 headers判断是{"Content-Type":"application/json"}  res=调用 post(url,headers,json=请求体)  1.2.2 headers判断不是json  res= 调用 post(url,headers,data=请求体)  响应：  return res.status\_code 授权码  return res.cookies.dict() cookies  return res.headers 响应头  return res.json() 响应体  都要？  解决： 需要内容二次封装 封装字典中合适  dict1=dict() 空字典  dict1["code"]=res.status\_code  dict1["cookies"]=res.cookies.dict()  dict1["headers"]= res.headers  dict1["body"]= res.json()  return dict1 # 接口访问的所有实际结果 +方便提取 |

示例代码：

|  |
| --- |
| # 导入Requests库  import json  import requests  # 导入logger对象  from utils.logutil import logger  # 定义http请求类  class RequestSend:  # 封装requests请求函数  # 必传 选传  def api\_run(self, url, method, data=None, headers=None, cookies=None):  # 定义变量，获取响应结果  res = None  # 判断请求方法  if method == "get":  # 如果是get方法，则执行下面命令，发送http请求，方法为get  res = requests.get(url, data=data, headers=headers, cookies=cookies)  # 如果是post方法，则执行下面命令  elif method == "post":  # 判断headers内容，如果是json格式  if headers == {"Content-Type": "application/json"}:  # 发送http请求，方法为post，参数使用json=data  res = requests.post(url, json=data, headers=headers, cookies=cookies)  # 判断headers内容，如果是application/x-www-form-urlencoded格式  elif headers == {"Content-Type": "application/x-www-form-urlencoded"}:  # 发送http请求，方法为post，参数使用data=data  res = requests.post(url, data=data, headers=headers, cookies=cookies)  # 获取请求响应的状态码  code = res.status\_code  # 获取请求响应的cookies  cookies = res.cookies.get\_dict() # cookies 比较特别需要转换才能拿到数据  #获取请求响应头的headers  headers=res.headers  # 3拼装结果数据  # 异常处理  try:  # 获取响应结果json格式  body = res.json()  # 捕获异常  except:  body = {"hi":"无数据"}  # 自定义参数code写入字典  # 定义字典  dict1 = dict()  dict1['code'] = code  # 自定义头信息封装起来  dict1['headers']=headers  # 自定义参数body写入字典  dict1['body'] = body  # 自定义参数cookies写入字典  dict1['cookies'] = cookies  # 返回自定义字典  return dict1  # 对外调用方法，\*\*kwargs 传入的参数是dict类型  def send(self, url, method, \*\*kwargs):  # 调用自定义方法  return self.api\_run(url=url, method=method, \*\*kwargs)  # 测试代码  if \_\_name\_\_ == '\_\_main\_\_':  # 登录接口的四要素  url = "http://127.0.0.1:6088/api/User"  data = {"username":"admin","password":"123"}  method = "post"  headers = {"Content-Type": "application/json"}  gongju4=RequestSend()  print(gongju4.send(url=url, method=method, headers=headers, data=data)) |

小结 第四把工具 名字 RequestSend 功能 send

发出接口请求 获取 封装实际结果

### 内容5：核心 组织DDT用例（面向工具类）：

思路：

1 细化需求----》用例（Pytest）

2 用例---》空函数+摸鱼（执行器+其他工具类）

3依次实现函数（支持ddt）

**今天目标 ： 执行独立接口用例+支持日志系统+支持 测试报告**

|  |
| --- |
| import pytest  # 工具类：  from utils.caseutil import RdTestCase  from utils.logutil import logger  from utils.requestutils import RequestSend  gongju2=RdTestCase() # 用例工具  case\_list=gongju2.is\_run\_data('okr-api') # [{用例1}，{用例2}，{用例3}]  gongju4=RequestSend() # 发出接口工具  # 思想 跑用例： gongju2得到用例数据 交给 gongju4 使用  class TestApi:  @pytest.mark.parametrize('case',case\_list)  def test\_run(self,case): # 用例条数 决定了 test\_run 个数～  # 需求1：发接口时候 拼接接口服务器地址～  url=gongju2.loadConfKey('atstudy\_okr','url\_api')['value']+case['url']  method = case['method']  # 坑：来自于数据库 字符串---》字典才能给 工具4使用解决 eval（）  headers = eval(case['headers'])  #print("当前用例请求头类型",type(headers))  cookies = eval(case['cookies'])  data = eval(case['request\_body'])  relation =str( case['relation'])  case\_name=case['title']  try:  logger.info(f"主人，正在执行{case\_name}用例")  res\_data=gongju4.send(url,method,data=data,headers=headers,cookies=cookies)  logger.info(f"实际结果：{res\_data}")  #发送失败 断言失败 发送成功 不能说明断言成功 进一步比对预期和实际 ==》测试结果  except:  logger.error("用例执行失败，请进一步排查")  assert False  return res\_data  if \_\_name\_\_ == '\_\_main\_\_':  pytest.main(["-s","-v","test\_run.py"]) |

预告：  **上下游接口（难点~）**+ 用例断言+用例结果回写（第四天）