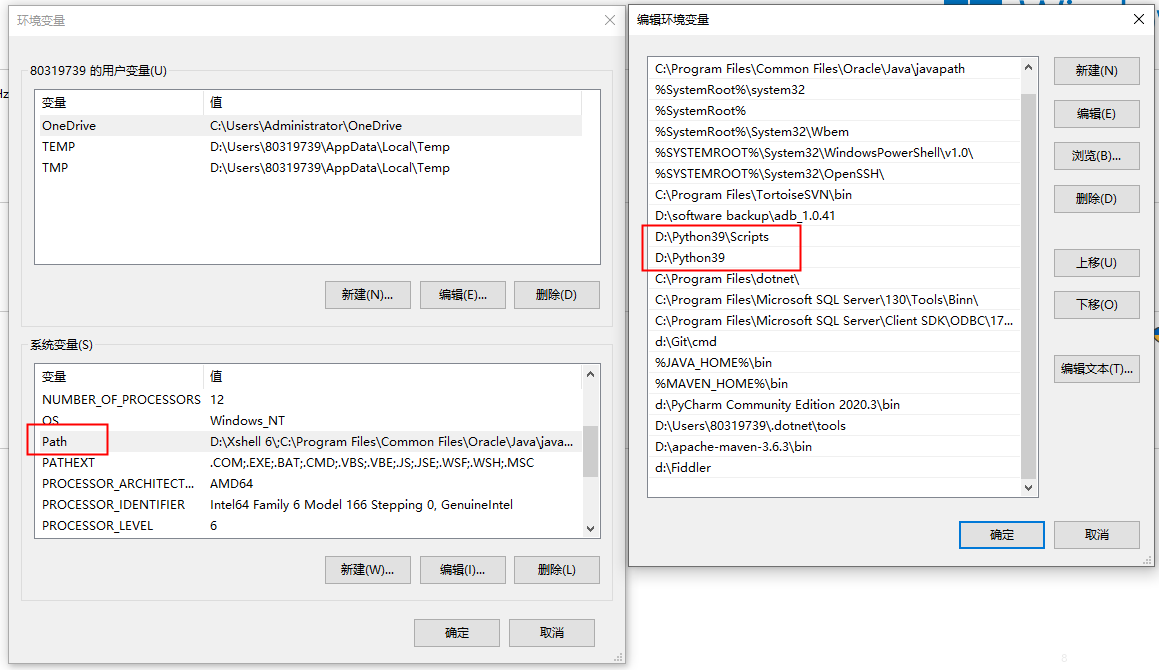
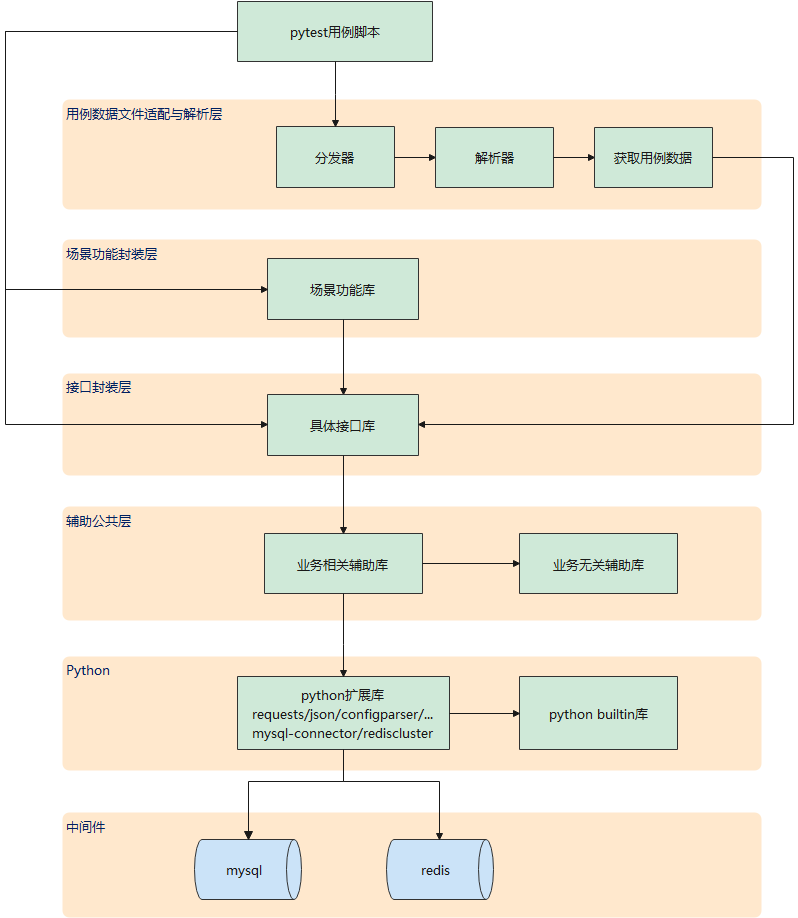
# 安全支付-接口自动化框架

—— 80319739（严非）

1. **安装**
2. 获取方式
   1. 切换到本地想保存自动化工程的目录
   2. git clone <http://code.oppoer.me/atms-scriptproject/finance-test/secure-payment/SecurePaymentsAutomation>
3. 基础环境安装
   1. 安装Python
      1. <https://www.python.org/downloads/> 下最新版
      2. 添加系统环境变量
   2. 安装依赖包
      1. 切换到工程根目录，命令行执行pip install -r requirement.txt
      2. 若安装失败，可以尝试手工重试 pip install xxxx，仍然失败，可以联系我
   3. 配置python运行环境（指定解释器）
      1. Pycharm（原生支持python），大多数人的选择
      2. Eclipse + Pydev插件
   4. 检验Python安装是否成功
      1. 命令行输入python，能进入控制台不报错，则表示系统能识别python

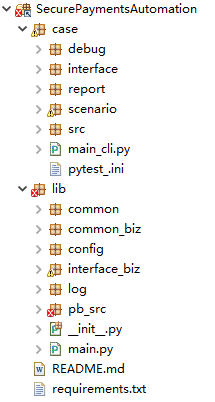


1. **框架**



1. **项目文件结构**
2. case
   1. interface：单接口用例
   2. scenario：场景用例
   3. debug：个人调试用例
   4. src：用例数据文件

分别按协议类型http/dubbo划分子用例集

1. lib
   1. common
      1. algorithm：各种加密算法
      2. case\_processor：用例数据文件操作相关
      3. concurrent：线程
      4. db\_operation：各类数据库操作
      5. exception：异常处理
      6. file\_operation：文件操作
      7. logger：日志处理
      8. session：http/dubbo交互
      9. utils：所有与业务无关的通用函数
   2. common\_biz
      1. 业务通用方法
      2. 签名算法
      3. 查找业务线和渠道私钥
      4. 分库分表查询
      5. 账号登录
   3. interface\_biz
      1. http/dubbo/scarlett各接口通用相关封装
   4. config：各类配置文件
      1. 账号
      2. 指定被测环境
      3. sql语句
      4. 私钥
      5. 各个测试环境对应的域名、数据库连接信息、dubbo信息等
   5. pb\_src：protobuf接口定义文件
2. 详情请参见



1. **case编写**
2. 编写测试数据文件
   1. 表格形式存储在case\src下，并按协议和区域归类，e.g.

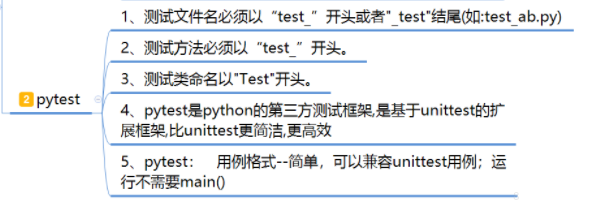
.\SecurePaymentsAutomation\case\src\http\inland.xlsx

协议是http，区域是国内

* 1. 每张工作表对应一个接口所有的正向、负向测试数据



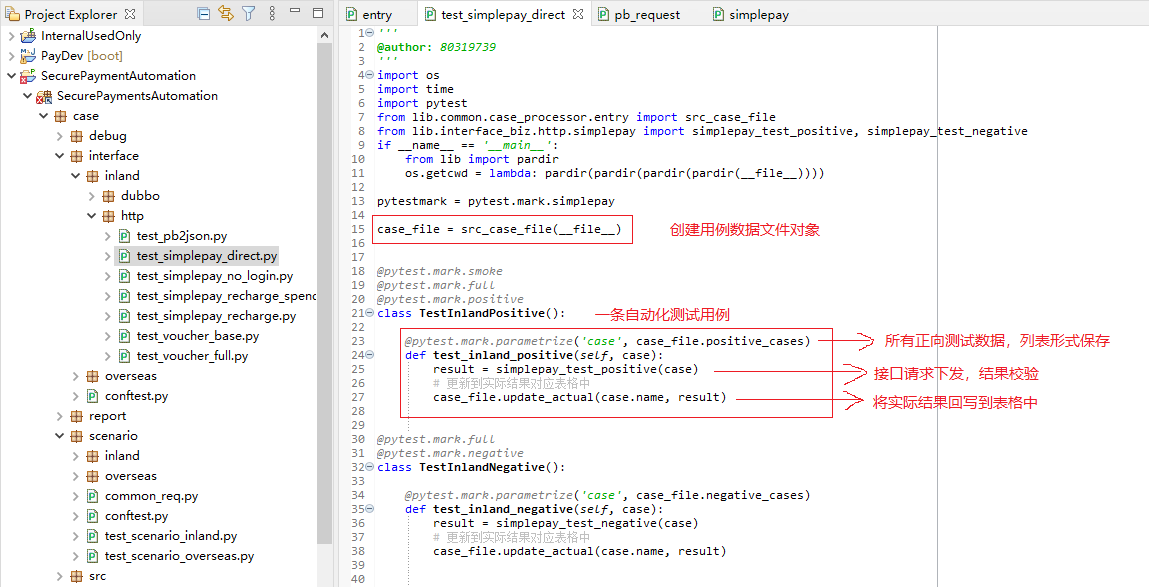
1. 编写测试用例
   1. 基于pytest框架
      1. 扩展unittest的自动化测试框架



* + 1. 判断用例成功失败——断言

pytest直接使用原生态的python assert表达式

* + 1. 参数化运行（数据驱动）
       1. 多行测试数据复用同一个测试用例，精简用例脚本代码量
       2. 数据存储在表格里，每次循环获取一行
    2. 自定义用例执行
       1. 按照用例标签，分类执行
       2. 按照用例名称
    3. 失败重跑机制
       1. 安装失败重跑插件：pytest-rerunfailures
    4. 测试报告
       1. 安装插件：pytest-html，报告为html格式，可读性一般
       2. 安装pytest-allure，报告可视化程度高，当前自动化测试平台采用
  1. 举个栗子
     1. 以直扣订单为例
     2. 请求下发和结果校验过程，若有需求，可以让lib开发者协助



1. 各用例之间必须解耦
   1. 单个用例执行之后不允许有残留数据——不留痕
   2. 建立良好的编写setup和teardown习惯（即前置条件和后置操作）
   3. 单个用例setup阶段创建或修改的内容，teardown阶段务必还原
2. 善用fixture，精简用例
   1. fixture（装置）是pytest在程序运行初始阶段就预编译加载的函数

########################################################################################

*@pytest.fixture(scope, params, autouse, ids, name)*

scope: 标志该装置的作用域范围，有4个层级：

session：全局，跨文件共享

module：模块（文件）

class：测试类

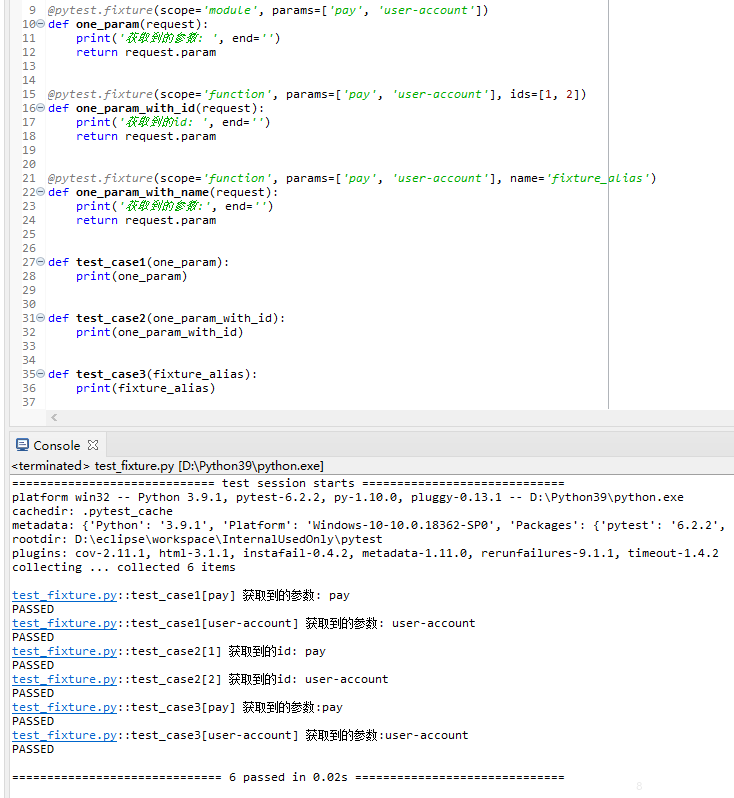
function：单用例

params：一个参数列表，会形成多个子用例，通过request.param获取值

autouse：是否自动运行该fixture

ids：输出结果中替代params的取值

name：重命名fixture



########################################################################################

1. 编写conftest.py（可选但推荐）
   1. 固定名称，不能随意更改
   2. 一般放置在项目根目录下，pytest会自动识别该文件；也可以放到某个package下，那就仅在该package内有效，所以可以有多个conftest.py
   3. 由b)，因此不需要显式import导入 conftest.py模块
   4. 所有同目录及子目录测试文件运行前都会优先执行conftest.py文件
   5. 用于存放各种级别的fixture，可被多个.py文件共享，一次执行处处可用
2. 运用参数化思想编写测试用例，精简用例脚本的代码量

########################################################################################

*@pytest.mark.parametrize('参数名1,参数名2,...', list)*

1、第一个参数是字符串，多个参数中间用逗号隔开

2、第二个参数是list，每个元素都是一个元组，元组里的每个元素跟参数顺序一一对应

[3、传一个参数：@pytest.mark.parametrize('参数名',](mailto:3、传一个参数@pytest.mark.parametrize('参数名',) list) 进行参数化

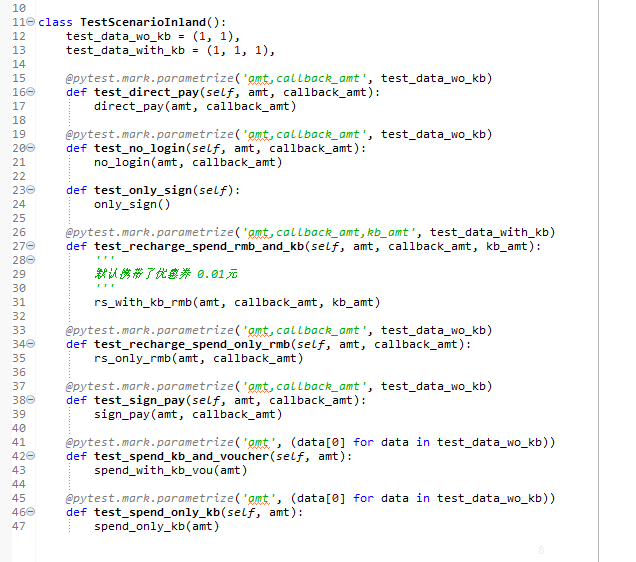
4、传两个参数：@pytest.mark.parametrize('参数1,参数2'，[(参数1\_data1, 参数2\_data2),(参数1\_data2, 参数2\_data2)])，e.g.

@pytest.mark.parametrize('method,url,param,expected', [

('get', 'www.baidu.com', '{"page":1}', '{"code":0,"msg":"成功"})',

('post', 'www.baidu.com', '{"page":2}','{"code":0,"msg":"成功"}')

])



########################################################################################

1. **lib编写要求**
2. interface\_biz调用common\_biz和common
3. common\_biz调用common

P.S. 不允许反向调用

1. **代码规范**
2. 命名规范

包名：全小写，蛇形格式

文件名（模块名）：全小写，蛇形格式

类名：驼峰格式

方法名：全小写，蛇形格式

函数名：全小写，蛇形格式

全局变量：全大写，蛇形格式

局部变量：全小写，蛇形格式

1. import规范

一个模块独占一行，from import 可以多模块同行导入

先后顺序：

标准库导入

相关第三方库导入

本地应用/库特定导入

推荐使用绝对路径导入（非强制），当绝对路径较长时，相对路径可以降低复杂度

1. lib编码规范
   1. 面向对象
   2. 高内聚、低耦合，提升扩展性、可读性、易用性
   3. 上层单向依赖下层，防止循环导入
   4. 相同变量若需多处访问，则提为全局变量
   5. 相同过程若需多处调用，则提为通用方法，善用AOP思想（装饰器或元类解决）
2. 其它代码风格要求，请参阅PEP8规范
3. **git代码提交/合并规范**
4. master提交

只允许提交lib代码，权限仅开放给肖遥、严非

1. case分支提交

测试人员从master（基线分支）checkout到自身分支，各自编写用例后，上传到远端对应 分支（不合入master）

注：第一次上传需要跟踪远程分支：git push --set-upstream origin 分支名

1. 分支合并

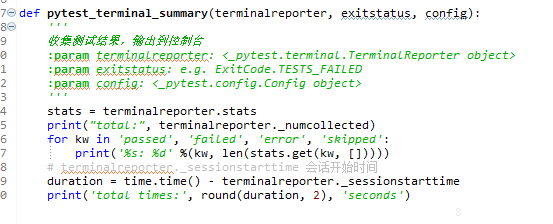
master单向并入case-xxx分支

1. **进阶**
2. 钩子函数

在conftest.py编写，pytest自动识别并调用，实现对应功能。e.g.

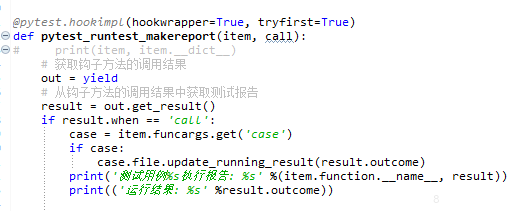
* 1. *pytest\_terminal\_summary*

获取并修改统计结果展示



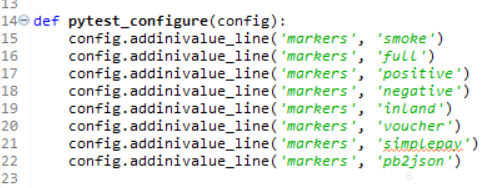
* 1. *pytest\_runtest\_makereport*

获取单个用例运行情况（不同执行阶段setup/call/teardown）



* 1. *pytest\_configure*

执行pytest初始配置



* 1. *pytest\_addoption(parser)*

动态增加命令行参数

parser.addoption('-s', help='不让pytest抓取日志，而是打印到控制台')

* 1. *pytest\_collection\_modifyitems(items)*

修改pytest待执行的用例执行顺序，自定义跳过某些用例