### DestroyerRobot下ApiTesting全链路接口自动化测试框架

--**2021年3月2日**

**谢猛--vivid**

**简介：**

此框架是基于Python+Unittest+BeautifulReport+requests+Config+yaml+json

+ddt实现全链路接口自动化数据驱动测试。

主要流程：解析接口数据包->生成接口基础配置（config项目主配置文件+yaml子项目主配文件）->生成测试用例（yaml+json）->生成测试脚本（.py）->运行测试场景（unittest）->生成测试报告（BeautifulReport）。

测试流程：初始化请求->除了接口基础信息->读取前置接口用例->发送前置接口->处理当前接口数据->发送当前接口->检查接口返回。

接口自动化测试几大块：测试用例设计、测试脚本编写、测试结果校验、测试报告生成、测试配置管理。

依赖关系及处理难点：接口之间依赖关系、测试数据与脚本分离、测试数据参数化处理。

而这些本框架通通已经解决，无需编写冗余代码，基本以copy和修改参数为准，需要获取接口数据信息即可。

关于接口依赖：只要填写签章接口对应路径即可，如果存在数据依赖关系，此时你可以填写前置接口对应的参数值，本框架将自动为你调用和替换关联数据。

关于测试数据：本框架采用yaml配置模板信息，excel文件保存请求参数和执行结果数据，将执行json信息写入到excel文件中，解决各类数据错综复杂的问题。

关于参数化：本框架提供了大量的util类来满足使用者对数据操作及采集。

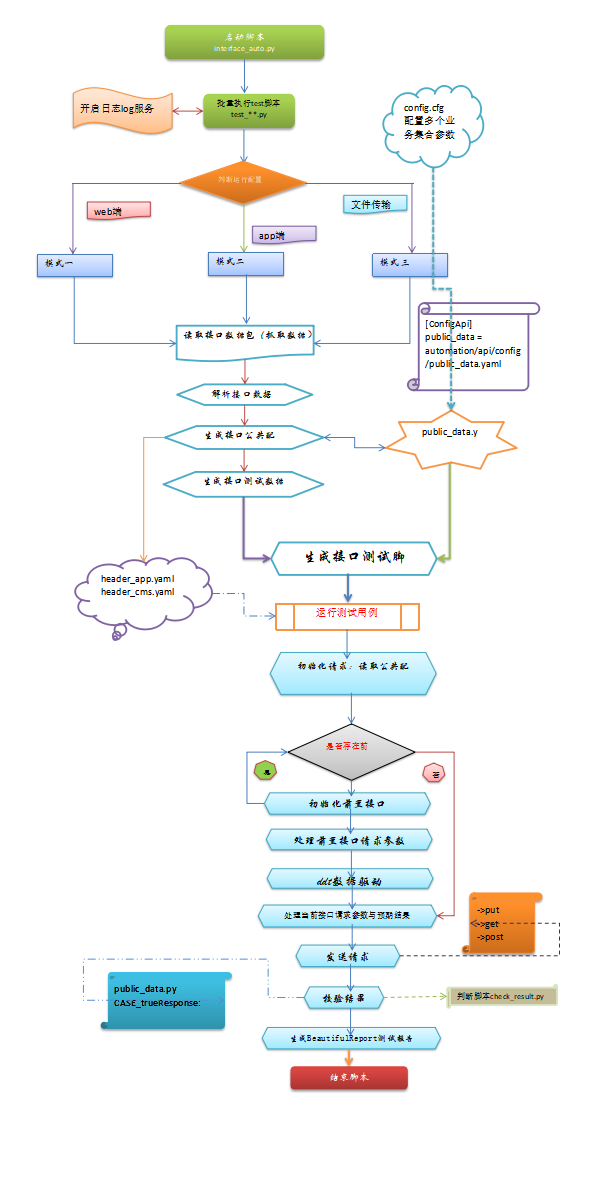
**目前主要支持三种运行模型：**

>0、web端接口请求

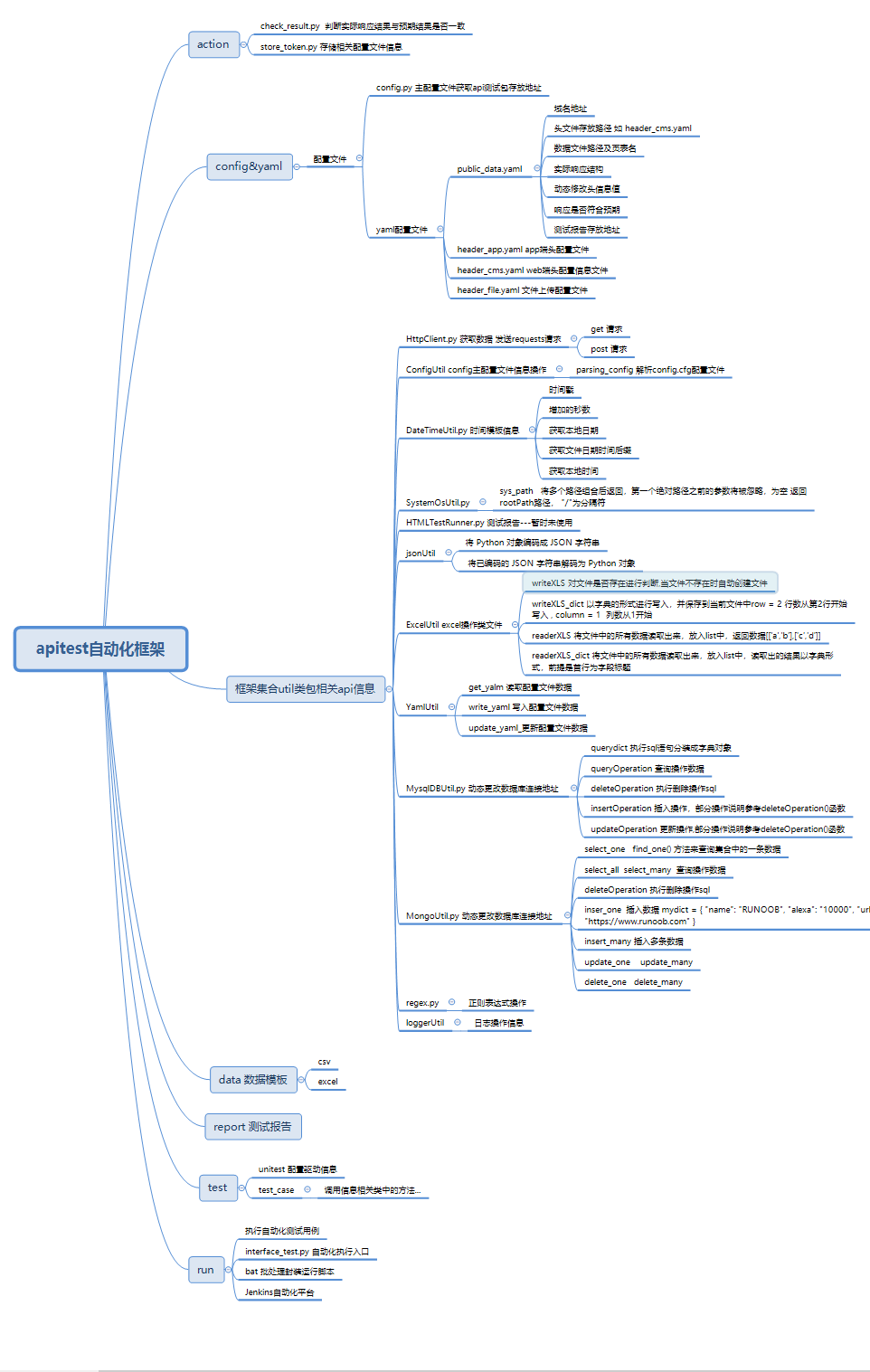
>1、文件传输

>2、app端接口请求

**框架流程图：**



**项目结构：**



**执行入口：run interface\_test.py**

|  |
| --- |
| **#!/usr/bin/env python**  **# -\*- coding: utf-8 -\*-**  **# @Time : 2020/11/9 15:45**  **# @Author : vivid**  **# @Email : 331597811@163.com**  **# @File : interface\_auto.test.py**  **import unittest**  **from BeautifulReport import BeautifulReport**  **from DestroyerRobot.automation.util.LoggerUtil import Log**  **from DestroyerRobot.automation.util.ConfigUtil import Config**  **from DestroyerRobot.automation.util.SystemOsUtil import SystemOs**  **from DestroyerRobot.automation.util.YamlUtil import yamlUtil**  **from DestroyerRobot.automation.util.DateTimeUtil import TestDateTime**  **class testFileReport:**  **def apis\_yaml(self):**  **conf = Config("ConfigApi")**  **api\_path = conf.get\_configPath("public\_data")**  **api\_yaml = SystemOs().sys\_path(api\_path)**  **report = yamlUtil(api\_yaml).get\_yalm()**  **report\_path = SystemOs().sys\_path(report['report'])**  **test\_path = SystemOs().sys\_path(report['test\_path'])**  **# 0 报告地址 ， 1 测试数据集地址**  **return report\_path,test\_path** |

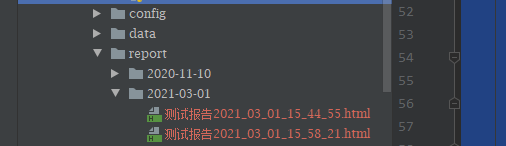
**操作脚本：test test\_cms.py**

|  |
| --- |
| **import unittest**  **import time**  **from ddt import ddt,data,unpack**  **from DestroyerRobot.automation.util.ConfigUtil import Config**  **from DestroyerRobot.automation.util.SystemOsUtil import SystemOs**  **from DestroyerRobot.automation.util.YamlUtil import yamlUtil**  **from DestroyerRobot.automation.util.ExcelUtil import xlsxoper**  **from DestroyerRobot.automation.util.LoggerUtil import Log**  **from DestroyerRobot.automation.util.HttpClient import HttpClient**  **from DestroyerRobot.automation.api.action.check\_result import CheckResult**  **from DestroyerRobot.automation.api.action.store\_token import stort\_token**  **@ddt**  **class TestCase(unittest.TestCase):**  **"""**  **全局变量中将excel中首行标题提取:testdata2 , yaml数据返回结果:datainfo**  **新脚本中配置新的excel需要改写内容如下 ：对应新的配置文件信息**  **global:全局变量**  **file\_path = SystemOs().sys\_path(datainfo['file\_path2'])**  **testdata = xlsxoper(file\_path).readerXLS\_dict(datainfo['SheetName2'])[0] # 数据用于ddt使用**  **testdata2 = xlsxoper(file\_path).readerXLS\_dict(datainfo['SheetName2'])[1] # 首行标题**  **def:**  **执行脚本对应的excel写入sheet**  **xls = xlsxoper(file\_path).writeXLS\_dict(datas['caseNo'],column,str(responseData),datainfo['SheetName2'])**  **"""**  **global log , file\_path,datainfo,testdata2**  **log = Log().logger()**  **conf = Config("ConfigApi")**  **keys = conf.parsing\_config("public\_data")**  **public\_data = SystemOs().sys\_path(keys) #通过主配置文件config获取主yaml配置文件路径-**  **datainfo = yamlUtil(public\_data).get\_yalm()**  **file\_path = SystemOs().sys\_path(datainfo['file\_path2'])**  **testdata = xlsxoper(file\_path).readerXLS\_dict(datainfo['SheetName2'])[0] # 数据用于ddt使用**  **testdata2 = xlsxoper(file\_path).readerXLS\_dict(datainfo['SheetName2'])[1] # 首行标题**  **#testdata,testdata2 = xlsxoper(file\_path).readerXLS\_dict(datainfo['SheetName2']) #等于 testdata,testdata2**  **#\*testdata 只获取数据**  **@data(\*testdata)**  **def test\_01\_demo(self, datas):**  **"""**  **接口测试**  **:param datas: 从yaml文件中获取excel路径，读取数据集，通过ddt形式**  **"""**  **isSkip = datas['isSkip'] #获取excle中isSkip**  **if isSkip == 'y':**  **log.info("当isSkip==y跳过此条用例")**  **else:**  **#print("datas:",datas)**  **webapp = datas['setApp']**  **setApp=datainfo[webapp] #获取excel中设备类型setApp是网页还是手机**  **log.info('setApp: %s ',setApp)**  **setpath = SystemOs().sys\_path(setApp) #获取yaml子配置文件路径信息如：C:/Users/vivid/PycharmProjects/untitled/DestroyerRobot/yaml\_file/header\_cms.yaml**  **header = yamlUtil(setpath).get\_yalm() #获取header信息**  **url = datainfo[datas['domain']]+datas['requestUrl']**  **#"请求方式= %s",datas['method'],"请求参数= %s",datas['param'] "请求格式= %s",datas['paramType'],"请求头部信息= %s",header**  **log.info("请求地址= %s ",url)**  **# log.info("请求方式= %s",datas['method'])**  **# log.info("请求格式= %s",datas['paramType'])**  **# log.info("请求头部信息= %s",header)**  **responseData = HttpClient().request(datas['method'],url,datas['paramType'],datas['param'],header) #实际结果**  **log.info("实际响应结果：%s",responseData)**  **responseResult = CheckResult.check(responseData, datas["exceptResponse"])**  **log.info("判断预期结果与实际结果是否一致：0 为 False 1 为 True = %s ",responseResult)**  **# 判断是否将token写入yaml对应的文件中**  **stort\_token(responseData,webapp,setpath,responseResult,datas['istoken'],datainfo['token'],datainfo['Authorization']).token()**  **#查找CASE\_trueResponse中value值对应excel的列位置**  **column = testdata2.index(datainfo['CASE\_trueResponse'])**  **# 写入excel中**  **log.info("将实际响应结果集数据写入excel中对应列" )**  **#写入excel数据必须是字符串，不能是dict**  **xls = xlsxoper(file\_path).writeXLS\_dict(datas['caseNo'],column,str(responseData),datainfo['SheetName2'])**  **time.sleep(1)**  **if \_\_name\_\_ == '\_\_main\_\_':**  **unittest.main()** |

**判断脚本 action check\_result.py store\_token.py**

|  |
| --- |
| **class CheckResult():**  **"""**  **responseData:判断实际响应结果与预期结果是否一致**  **exceptResponse ：**  **"""**  **@classmethod**  **def check(self, responseData, exceptResponse):**  **if isinstance(responseData ,str):**  **responseData = eval(responseData)**  **if isinstance(exceptResponse , str):**  **exceptResponse = eval(exceptResponse)**  **for key, value in exceptResponse.items():**  **if key in responseData:**  **if responseData[key] == value:**  **responseResult = 1**  **else:**  **responseResult = 0**  **return responseResult**  **else:**  **responseResult = 0**  **return responseResult**  **return responseResult** |
| **class stort\_token:**  **"""**  **stort\_token只是为了满足权限系统而作**  **"""**  **def \_\_init\_\_(self,responseData,setApp,setpath,responseResult,istoken,token\_value,Authorization):**  **self.responseData = responseData**  **self.istoken = istoken**  **self.setpath = setpath**  **self.setApp = setApp**  **self.responseResult = responseResult**  **self.tokenvalue =token\_value**  **self.Authorization = Authorization**  **self.log = Log().logger()**  **def token(self):**  **#print(self.responseData,"\n\t",self.istoken ,"\n\t",self.setpath ,"\n\t", self.setApp ,"\n\t" ,self.responseResult ,"\n\t",self.tokenvalue ,"\n\t",self.Authorization )**  **if self.istoken :**  **if self.responseResult :**  **if "data" in self.responseData.keys():**  **if self.responseData["data"] == None:**  **self.log.info("data层为null")**  **else:**  **if self.tokenvalue in self.responseData['data'].keys():**  **reToken = self.responseData["data"][self.tokenvalue]**  **#在权限系统中生成"Bearer "+applicationToken 对应Authorization**  **reAuthorization = "Bearer "+self.responseData["data"]["applicationToken"]**  **self.log.info("reToken:%s" % reToken)**  **if self.setApp == 1 :**  **data = yamlUtil(self.setpath).get\_yalm()**  **if self.tokenvalue in data.keys():**  **yamlUtil(self.setpath).update\_yaml(self.tokenvalue,reToken)**  **else:**  **Token = {self.tokenvalue: reToken}**  **yamlUtil(self.setpath).write\_yaml(Token)**  **if self.Authorization in data.keys():**  **yamlUtil(self.setpath).update\_yaml(self.Authorization,reAuthorization)**  **else:**  **Authorization = {self.Authorization: reAuthorization}**  **yamlUtil(self.setpath).write\_yaml(Authorization)**  **elif self.setApp == 2 :**  **data = yamlUtil(self.setpath).get\_yalm()**  **if self.tokenvalue in data.keys():**  **yamlUtil(self.setpath).update\_yaml(self.tokenvalue,reToken)**  **else:**  **reToken = {self.tokenvalue : reToken}**  **yamlUtil(self.setpath).write\_yaml(reToken)**  **else:**  **self.log.info("登录没有返回token")**  **else:**  **self.log.info("response没有data层")**  **else:**  **self.log.info("登录失败" )**  **else:**  **self.log.info("不是登录")** |

**测试报告实例：按照日期生成对应测试报告**





**执行excel数据结果：异常测试以报告为准**

