

## 基于 python+Testlink+Jenkins 实现的接口自动化测试框架 V3.0

by:授客 QQ: 1033553122

博客: <http://blog.sina.com.cn/ishouke>

欢迎加入软件性能测试交流 QQ 群: 7156436

### 目录

1、 开发环境.....	1
2、 主要功能逻辑介绍.....	2
3、 框架功能简介.....	3
4、 数据库的创建.....	4
5、 框架模块详细介绍.....	5
6、 Testlink 相关配置与用例管理.....	13
a) API 相关配置.....	13
b) 项目, 计划, 套件等相关配置.....	17
c) 用例管理.....	18
7、 运行结果.....	28
8、 源码下载.....	28
9、 说明.....	28

#### 1、 开发环境

win7

PyCharm 4.0.5

python 3.3.2

mysql-connector-python-2.1.3-py3.3-win32

下载地址: <http://pan.baidu.com/s/1kTRqRht>

testlink-1.9.14

下载地址: <http://pan.baidu.com/s/1pLrcunT>

安装教程: [http://blog.sina.com.cn/s/blog\\_13cc013b50102w9am.html](http://blog.sina.com.cn/s/blog_13cc013b50102w9am.html)

TestLink-API-Python-client-master

下载地址 1: <http://pan.baidu.com/s/1c16H500>

下载地址 2:

<https://github.com/lczub/TestLink-API-Python-client#testlink-api-python-client-developers>

chardet-2.3.0

下载地址 1: <https://pypi.python.org/pypi/chardet/>

下载地址 2: <http://pan.baidu.com/s/1nu7XzjN>

## 2、 主要功能逻辑介绍



### 3、 框架功能简介

- 1、框架集成了 Testlink, 可使用 Testlink 灵活对测试项目, 测试计划, 测试用例进行管理
- 2、可通过配置文件灵活配置运行模式:
  - 支持按测试项目运行: 一次运行单个、多个指定的项目或者全部项目;
  - 支持按测试计划运行: 一次运行单个、多个指定的测试计划;
  - 支持按测试套件运行: 一次运行单个、多个指定的测试套件(注: 支持套件嵌套, 套件 -- testlink 中的测试集)
  - 支持按用例运行: 一次运行单个\多个用例, 这点对特别方便开发阶段时, 对单个接口的实现代码进行调试
- 3、支持 HTTPS, HTTP, Webservice 协议, 支持 POST, GET 方法, 支持 JSON, 非 JSON 数据格式的请求, 支持多种形式的数据库校验, 包含数据库级别的数据校验

#### 4、支持在界面化操作, 无须写代码就可以实现如下操作:

- a) 自定义变量存储 web 服务器、数据库服务器返回请求/查询结果
- b) 根据自定义模式对 web 服务器返回结果进行自动校验, 支持多种模式的校验, 包含字符串, 不包含字符串, 键值提取, 包含成员, 不包含成员, 匹配/不匹配正则表达式, 完全匹配列表/元组/集合/字典
- c) 根据界面输入的 sql 语句, 执行 sql 查询/更新操作, 针对只对返回单条记录的 sql 查询, 还支持对查询结果进行提取, 保存
- d) 支持 url 及参数体的动态参数化, 支持全局动态参数, 非全局动态参数 (如存储某个接口返回结果的自定义变量)

#### 5、针对脚本中已经支持的常见协议及常用数据格式, 且不需对接口执行结果进行数据库级别的逻辑校验, 支持界面直接增加用例而不需要改动脚本代码, 即不会编码的人也可以使用本框架

#### 6、支持不同编码(utf8,ascii,gb2312)的返回结果, 且可自由扩展

#### 7、可自动生成 HTML 可视化接口测试报告

#### 8、可根据配置在测试完成后, 自动发送测试报告邮件, 邮件发送支持 SSL 加密发送和非 SSL 加密发送, 同时支持往多个邮箱发送邮件

#### 9、支持文件、控制台的日志打印, 可分别控制开关

#### 10、支持模块化开发

#### 11、支持测试环境的“一键”切换: python main.py arg, eg python main.py 1

其中,arg: 1-测试环境 2-预发布环境 3-集成环境, 可根据实际需要在代码、配置文件中做适当调整, 支持自由扩展和更改

#### 12、可集成 Jenkins 自动运行脚本

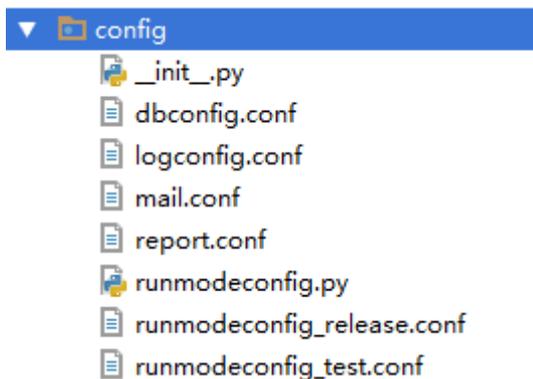
参考文章: [为 Jenkins 添加 Windows Slave 远程执行 python 项目脚本](#)

## 4、数据库的创建

```
CREATE DATABASE IF NOT EXISTS interface_autotest DEFAULT CHARACTER SET utf8;
```

## 5、 框架模块详细介绍

### a) config



**dbconfig.conf:** 包含测试数据库，应用数据库的配置信息

**logconfig.conf:** 包含日志配置信息，具体如下：

```

[LOGGING]
log_file = F:\project\interface_project\logs\log.txt ← 日志文件所在路径
max_bytes_each = 51200 ← 单个日志文件大小
backup_count = 10 ← 同一时刻，可存留的日志文件数
fmt = |(asctime)s |(filename)s[line: |(lineno)d] |(levelname)s: |(message)s
logger_name = test_logger ← 日志打印器名称
log_level_in_console = 10 ← 日志级别
log_level_in_logfile = 20 ← 日志级别
console_log_on = 1 ← 日志打印开关
logfile_log_on = 1 ← 日志打印开关

[README]
log_level = '日志级别: CRITICAL = 50 ERROR = 40 WARNING = 30 INFO = 20 DEBUG = 10 NOTSET = 0'
log_on = 'console_log_on = 1 开启控制台日志, 0则关闭, logfile_log_on = 1 开启文件日志, 0则关闭'
    
```

**mail.conf:** 包含邮件发送配置信息，如下，

```

1  [SMTP]
2  login_user = laiyuhenshuai@163.com
3  login_pwd = xxozhe
4  from_addr = laiyuhenshuai@163.com
5  to_addrs = ['1033553122@qq.com', 'xxxxx@163.com']
6  host = smtp.163.com
7  port = 25
8  encrypt = 0
9
10 [README]
11 encrypt = '1 - send via SSL, 0-sent not via SLL'
    
```

注: 不同类型的邮箱(发件人邮箱), 需要修改配置文件为对应的 host 和端口  
 smtp.163.com:25  
 smtp.qq.com:465

**report.conf:** 包含测试报告文件配置信息, 如下

```

[REPORT]
dir_of_report = F:\\project\\report\\
report_name = test_report.html

[README]
dir_of_path = '不能加引号,'D:\\tset\\tkise\\, r'D:\\tset\\tkise\\
report_name = '不能加引号'
    
```

**runmodeconfig\_xxxx.conf:** 包含运行模式配置信息, runmodeconfig\_test.conf 和 runmodeconfig\_release.conf 分别为测试环境和预发布环境的运行时配置信息, 可根据实际情况进行调整

```

runmodeconfig.conf x
1  [RUNMODE]
2  runmode = 1
3
4  [PROJECTS]
5  project_mode = 2
6  projects = ['项目1', '项目2', '项目3', '项目4']
7
8  [PLANS]
9  project = 项目1
10 plans = ['项目1-测试计划1', '项目1-测试计划2', '项目1-测试计划3', '项目1-测试计划4', '项目-测试计划4', '项目1-测试计划5']
11
12 [TESTSUITES]
13 testsuites = [3, 11, 31, 35, 38, 43, 48, 56, 65, 69]
14
15 [TESTCASES]
16 case_id_list = [109]
17
18 [GLOBALCASES]
19 global_case_id_list = [114]
20
21 [README]
22 runmode = 'runmode: 1 - 按项目运行 2 - 按计划运行 3 - 按套件运行 4 - 运行指定用例'
23 testsuites = ['套件1id', '套件2id', '...', '套件Nid']
24 plans = '按计划运行时, 需要指定项目及项目关联的测试计划'
25 projects = '如果project_mode=2, 那么需要配置需要运行的项目, 如果project_mode配置为1, 则运行所有项目'
26 case_id_list = '[testcase_id1, testcase_id2, ..., testcase_idN], 按指定用例时, 需要指定需要运行的用例id'
27 global_case_id_list = '[testcase_id1, testcase_id2, ..., testcase_id2], 需要优先运行的全局初始化用例'
28

```

```

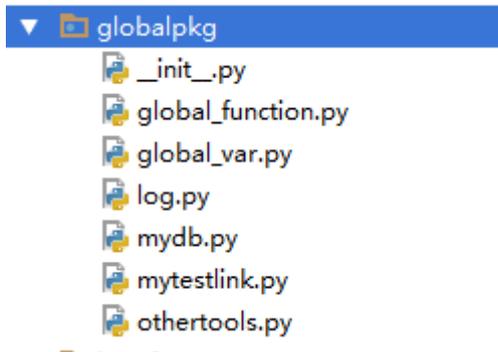
runmodeconfig.conf x  main.py x
Python interpreter configured for the project

logger.info('正在读取运行模式配置')
if sys.argv[1] == '1':
    run_mode_conf = RunModeConfig('./config/runmodeconfig_test.conf')
elif sys.argv[1] == '2':
    run_mode_conf = RunModeConfig('./config/runmodeconfig_release.conf')
run_mode = int(run_mode_conf.get_run_mode())

```

runmodeconfig.py: 运行配置类

b) globalpkg



**log.py:** 实现日志打印类

**mydb.py:** 实现数据库类, 封装数据库相关操作

**mytestlink.py:** 主要用于获取 testlink 连接实例

**othertools.py:** 实现其它通用功能, 比如数据转换, 批量创建目录等

**global\_var.py:** 主要提供全局变量, 全局实例等

```

global_var.py x
#!/usr/bin/env python
# -*- coding:utf-8 -*-

__author__ = 'lsifuyu'

import time
import sys

from globalpkg.log import logger
from globalpkg.mydb import MyDB
from globalpkg.mytestlink import TestLink
from globalpkg.othertools import OtherTools

__all__=['global_headers', 'global_openId', 'global_serial', 'global_protocol_version',
        'global_product_version', 'saofudb', 'testdb', 'mytestlink',
        'other_tools', 'executed_history_id', 'testcase_report_tb', 'case_step_report_tb']

# 根据自己的实际需要进行合理的调整
if sys.argv[1] == '1':
    logger.info('当前运行环境: 测试环境')
    logger.info('正在初始化数据库[名称: SAOFUDB1]对象')
    saofudb = MyDB('./config/dbconfig.conf', 'SAOFUDB1')
elif sys.argv[1] == '2':
    logger.info('已选择运行环境: 预发布环境')
    logger.info('正在初始化数据库[名称: SAOFUDB2]对象')
    saofudb = MyDB('./config/dbconfig.conf', 'SAOFUDB2')

logger.info('正在初始化数据库[名称: TESTDB]对象')
testdb = MyDB('./config/dbconfig.conf', 'TESTDB')

logger.info('正在获取testlink')
mytestlink = TestLink().get_testlink()

other_tools = OtherTools()

executed_history_id = time.strftime('%Y%m%d%H%M%S', time.localtime()) # 流水记录编号
# testcase_report_tb = 'testcase_report_tb' + str(executed_history_id)
# case_step_report_tb = 'case_step_report_tb' + str(executed_history_id)
testcase_report_tb = 'testcase_report_tb'
case_step_report_tb = 'case_step_report_tb'
    
```

定义的全局变量都在这里添加



```

__all__=['global_headers', 'global_openId', 'global_serial', 'global_protocol_version',
        'global_product_version', 'saofudb', 'testdb', 'mytestlink',
        'other_tools', 'executed_history_id', 'testcase_report_tb', 'case_step_report_tb']
    
```

可根据需要, 在这添加应用db



可根据实际需要, 未每次测试运行新建测试用例及步骤报表, 也可以设置为两张固定的表



```

# 请求都携带的公用请求头、请求参数
if sys.argv[1] == '1': # 测试环境的全局变量
    global_headers = {'saofu_client.test.saofu.cn': {}, 'm.test.saofu.cn': {'Cookie': '10549840601068216320=ous64uFCCLMyXYDJ-MkHilyCI5CY'}}
    global_serial = '10549840601068216320'
    global_openId = 'ous64uFCCLMyXYDJ-MkHilyCI5CY'
    global_product_version = '3.2.12C'
    global_protocol_version = '4.0'
elif sys.argv[1] == '2': # 预发布环境的全局变量
    global_headers = {'m.test.saofu.cn': {'Cookie': '10549840601068216320=ous64uFCCLMyXYDJ-MkHilyCI5CY'}}
    global_serial = '10549840601068216320'
    global_openId = 'ous64uFCCLMyXYDJ-MkHilyCI5CY'
    global_product_version = '3.2.12C'
    global_protocol_version = '4.0'
# 自己自由扩展和更改

```

1、全局变量，比如以global\_打头  
2、针对不同站点/host的全局请求头，必须按以下格式填写 {'host\_name': 'header\_name': 'header\_value'}

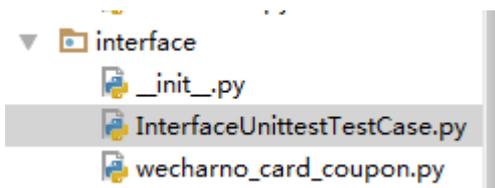
**global\_function.py:** 主要提供全局函数，比如提供运行单个用例的函数 `run_testcase_by_id` 等

**c) logs 及 testreport**

可分别用于存放日志文件，测试报告

**d) interface**

封装接口测试方法类



说明：可根据需要，每个接口对应一个模块，对应一个类；也可以多个接口对应一个模块，对应一个类  
需要注意的是，这里添加的模块及类，需要在 `casestep.py` 中导入

```

1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3
4  __author__ = 'laifuyu'
5
6  import unittest
7  import re
8
9  from globalpkg.log import logger
10 from globalpkg.global_var import *
11 from interface.InterfaceUnittestTestCase import *
12 from interface.wecharno_card_coupon import *

```

接口单元测试类代码说明

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

__author__ = 'laiyu'

import urllib.request
import json
import chardet

from globalpkg.log import logger
from globalpkg.global_var import *
from unittesttestcase import MyUnitTestCase
from httpprotocol import MyHttp

__all__ = ['InterfaceUnitTestCase']

class InterfaceUnitTestCase(MyUnitTestCase):

    outputs_list = []

    def test_interface_of_urlencode(self): # 针对请求体为url编码的: b'i=i=1318&password=e10adc3949ba59abbe56e057f20f883e'
        headers = self.headers.copy()
        for host in global_headers:
            # 判断该host下是否有全局请求头
            host_of_interface = self.http.get_host()
            if host == host_of_interface:
                headers.update(global_headers[host])
        self.http.set_header(headers)

        method = self.method.lower()
        if method == 'post':
            logger.info('正在发起POST请求...')
            self.params = urllib.parse.urlencode(self.params) # 将参数转为url编码字符串# 注意, 此处params为字典类型的数据
            self.params = self.params.encode('utf-8')
            response = self.http.post(self.url, self.params)
        elif method == 'get':
            logger.info('正在发起GET请求...')
            self.params = urllib.parse.urlencode(self.params)
            response = self.http.get(self.url, self.params)
```

对于自己新增的自定义模块, 这部分的内容可直接复制黏贴, 不能少, 也可以在此基础上新增其它

添加的每个模块都包含一个\_\_all\_\_ list列表, 并存储模块中添加的类

为每个类都添加这样一个相同名称的全局变量

每个test方法代表着一类具备相同请求处理方式的请求, 可根据实际情况进行添加, 建议格式为 test\_interface\_of\_type

其它未框选部分的代码, 针对同种处理方式也是固定不变的, 复制黏贴即可

说明: 处理方式目前支持 3 中 test\_interface\_of\_urlencode (针对请求体为 url 编码), test\_interface\_of\_json(针对请求体为 json 格式), test\_interface\_of\_xml(针对请求体为 xml 格式), test\_interface\_of\_outer(针对外部接口),

```

encoding = chardet.detect(response[0])['encoding']
logger.info('正在对服务器返回body进行解码')
if encoding == 'GB2312':
    body = response[0].decode('gbk') # decode函数对获取的字节数据进行解码
elif encoding == 'utf-8':
    body = response[0].decode('utf-8')
elif encoding == 'ascii':
    body = response[0].decode('unicode_escape')
else:
    logger.info('解码失败,未知编码:%s,不对body做任何解码' % encoding)
    body = response[0]

```

这里可根据实际返回结果的编码,动态新增elif分支判断处理,其它未框选部分固定,复制黏贴即可

```

header = response[1]
code = response[2]

logger.info('服务器返回结果"响应体(body)": %s' % body)
logger.info('服务器返回结果"请求头(headers)": %s' % header)
logger.info('服务器返回结果"状态码(code)": %s' % code)

```

```

if self.expected_result == '': # 不对结果做任何处理
    self.assertEqual(1, 1, msg='测试通过')
    return

```

```

response_to_check = (self.expected_result['检查']).lower()

```

```

if response_to_check == 'body':
    self.assert_result(body)
    # 断言成功后,保存结果
    self.save_result(body)

```

这部分代码固定不变,复制黏贴即可

```

elif response_to_check == 'header':
    self.assert_result(header)
    # 断言成功后,保存结果
    self.save_result(header)
elif response_to_check == 'code':
    self.assert_result(code)
    # 断言成功后,保存结果
    self.save_result(code)

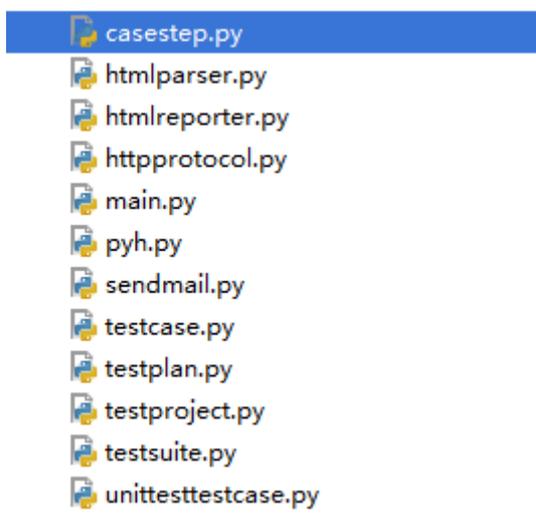
```

### 问题:假如要进行数据库后台的逻辑校验咋办?

针对核心用例我们可以这样,复制整个 test\_xxx 为 1-到 n 份,修改方法名,然后在代码最下方新增数据库后台校验的代码,同时如果有必要,对其它代码进行适当的修改(新增、删除、修改等),然后用例步骤中显示指定要调用的方法即可。

### e) 其它模块

如下, 顾名思义

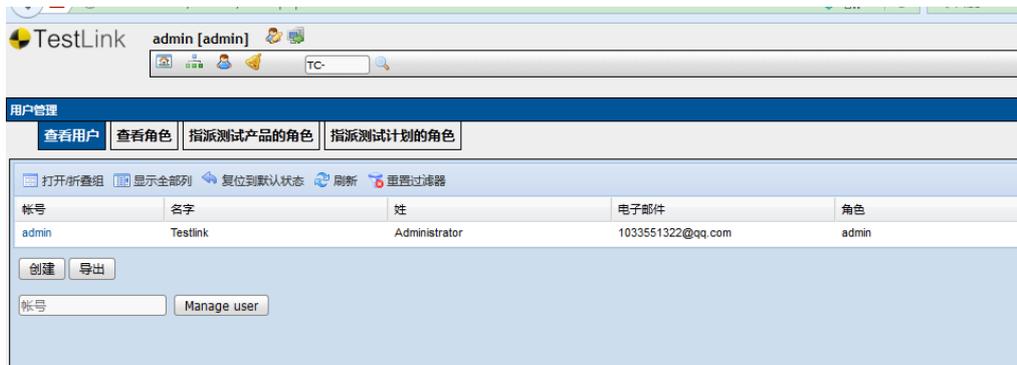


## 6、Testlink 相关配置与用例管理

为了批量设置接口 ip, 端口(主要是这两个), 协议信息(仅用于展示), 需要对项目, 计划, 套件等必要的配置, 以及客户端环境变量配置

### a) API 相关配置

如下, 登陆 Testlink, 进入用户管理-查看用户, 如下



点击目标用户(例中为 admin), 打开如下界面

The screenshot shows the TestLink interface for editing a user. At the top, the TestLink logo is on the left, and the user 'admin [admin]' is on the right. Below the logo is a navigation bar with icons for home, users, a user profile, a bell, and a search box containing 'TC-'. The main header is 'User Management - Account Settings' with tabs for 'Edit User', 'View Users', 'View roles', 'Assign Test Project roles', and 'Assign Test Pla'. The 'Edit User' tab is active. The 'User details' section contains a form with the following fields: Login (admin), First Name (Testlink), Last Name (Administrator), Email (1033551322@qq.com), Role (admin), Locale (Chinese Simplified), Authentication method (Default(DB)), and Active (checked). Below the form are 'Save' and 'Cancel' buttons. At the bottom of the form area are 'Reset password' and 'Generate a new key' buttons.

This section is titled 'API 接口' (API Interface). It shows '个人 API 访问密钥 = 无' (Personal API access key = None). Below this is a button labeled '生成新的密钥' (Generate new key). The section is followed by '登录历史' (Login History) with an information icon. Under '上次成功登录' (Last successful login), it displays: '2016-03-19 22:14:58 从 '192.168.1.101' 登录 'admin' from '192.168.1.101' 成功'.

点击生成新的密钥，如下

### API 接口

个人 API 访问密钥 = b728afa3f3eb61a5c42fc0ed6e6a2e92

生成新的密钥

或者是点击“编辑用户按钮”按钮进入界面

TestLink admin [admin]

帐号设置

#### 个人信息

帐号	admin
名字	<input type="text" value="Testlink"/>
姓	<input type="text" value="Administrator"/>
电子邮件	<input type="text" value="1033551322@qq.com"/>
语言	Chinese Simplified <input type="button" value="v"/> Last localization update: 2013-04-14 (Testlink 1.9.6)

#### 个人密码

你的旧密码	<input type="password" value="•••••"/>
输入新密码	<input type="password"/>
确认新密码	<input type="password"/>

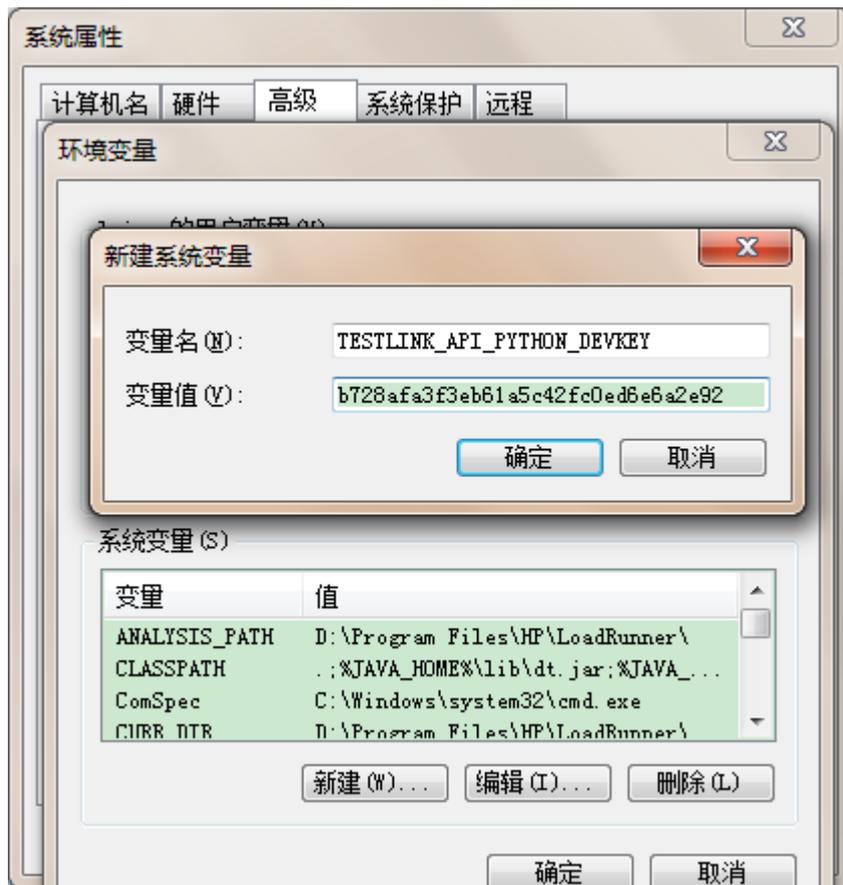
#### API 接口

个人 API 访问密钥 = 965a88dd56a1fe1c4d0d7891ed87272d

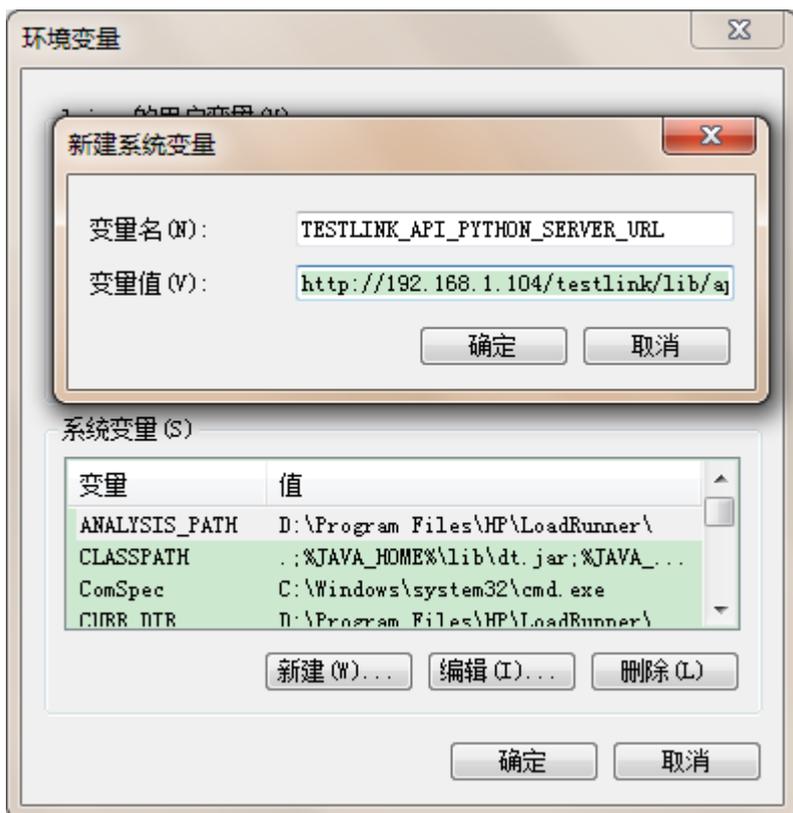
生成新的密钥

在运行 python 脚本端进行环境变量的配置，如下：

1、新建系统环境变量“TESTLINK\_API\_PYTHON\_DEVKEY”，变量值为上述秘钥



2、新建“TESTLINK\_API\_PYTHON\_SERVER\_URL”系统环境变量，变量值为“<http://{host}/testlink/lib/api/xmlrpc/v1/xmlrpc.php>”，其中 host 为 testlink 的访问地址



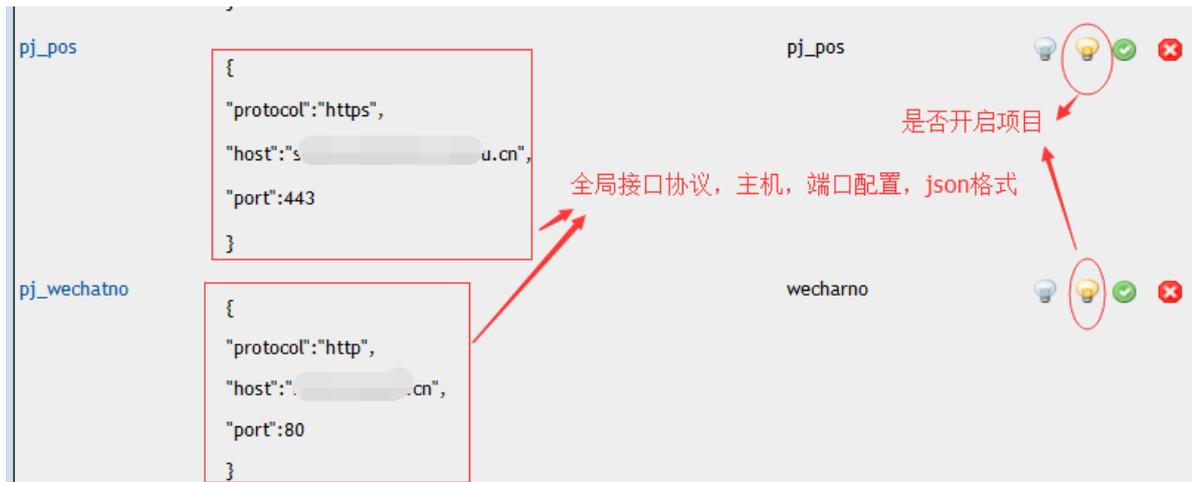
测试是否生效:

```
C:\Users\laiyu>python
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:03:43) [MSC v.1600 32
tel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import testlink
>>> tls = testlink.TestLinkHelper().connect(testlink.TestlinkAPIClient)
>>> tls.testLinkVersion()
'1.9.14'
```

注意: 如果还不行, 提示 404 错误, 则还需要配置 testlinkhelper.py 中的 DEFAULT\_SERVER\_URL, 将其设置为 <http://{host}/testlink/lib/api/xmlrpc/v1/xmlrpc.php>  
python\_installation\_home\Lib\site-packages\TestLink-API-Python-client-master\build\lib\t  
estlink\testlinkhelper.py

**b) 项目, 计划, 套件等相关配置**

测试项目配置



### 测试计划配置

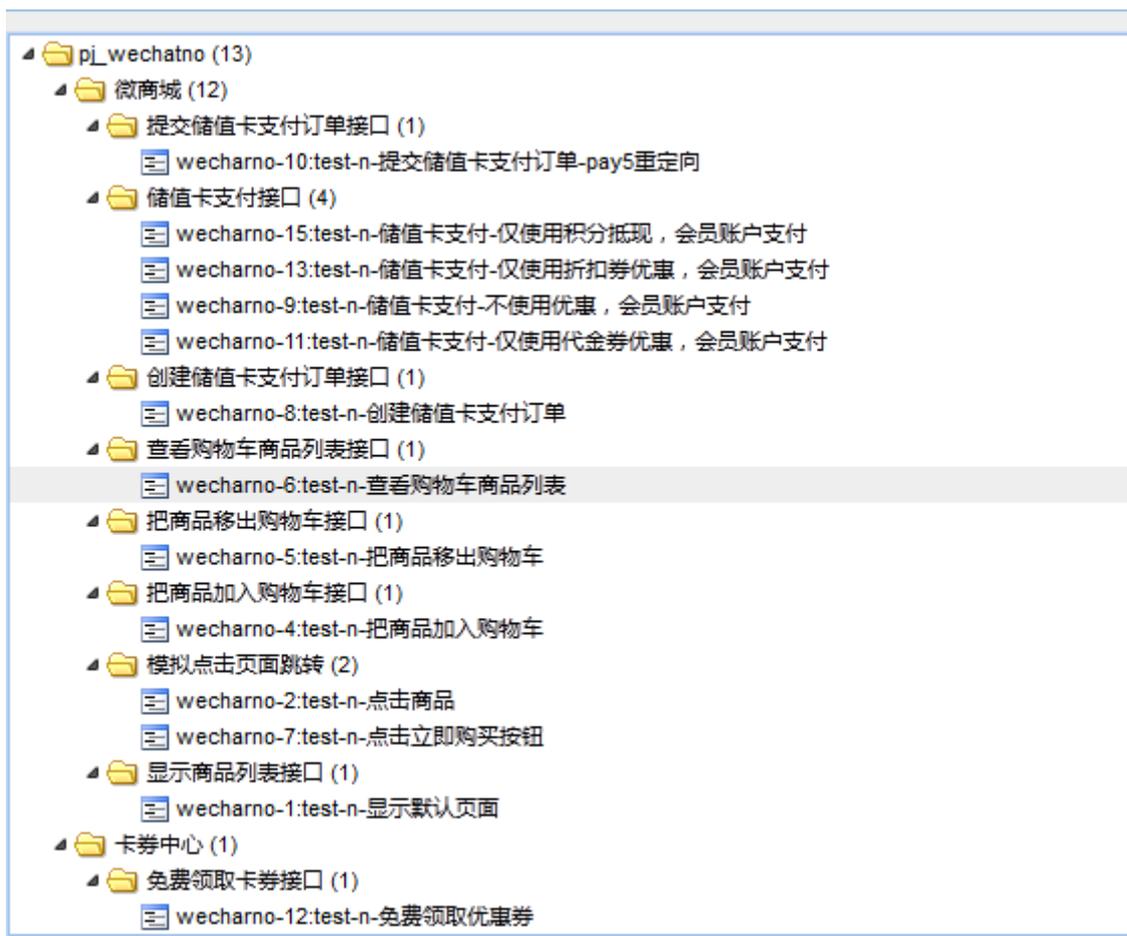


### 测试套件配置



### c) 用例管理

如下, 用例层级, 支持嵌套管理



用例编写如下，左侧步骤动作，必须按如下格式编写

① 步骤动作和预期结果填写规范

针对 http,https 请求，且步骤不是执行单个用例、不是执行 sql，按如下填写：

步骤动作

json 格式

```
{
  "类名": "InterfaceUnittestTestCase",
  "函数": "test_interface_of_urllencode",
  "请求头": {"Content-Type": "application/x-www-form-urlencoded", "charset": "utf-8"},
  "方法": "post",
  "url": "/cmorder/1/4",
  "参数": {"serial": "[global_serial]", "openId": "[global_openId]", "fm": 1, "mallGoodsId": "26838", "amount": 1, "cartIds": "", "fetchTime": "", "remark": ""}
}
```

注：

- 1、如果不填写“类名键值对”，那么类名默认为 InterfaceUnittestTestCase，如果不填写“函数键值对”，那么函数默认为 test\_interface\_of\_urllencode

- 2、键对应的参数值为空，则填写为 ""，"键"，用双引号包含，键对应的"值"，有必要的也是双引号包含，预期结果的填写也是一样，不再赘述
- 3、如果"参数"值为空，则填写为 "参数": ""
- 4、用例步骤对应的 test\_xxx 方法实现，要尽量做到接口测试数据和业务逻辑分离，增强复用性和用例的可维护性。

## 预期结果

### json 格式

```
{
  "检查": "body",
  "匹配规则": "包含字符串",
  "条件": [{"模式": "\\success\\:true", "消息": "创建储值卡支付订单失败， success 不为 True"}],
  "输出": {"success": "\\success\\:(.+?)", "attach": "attach\\:\\(.+?)\\"}
}
```

注:

- 1、预期结果要么按规范填写，要么为空，啥都不写
- 2、如果不想定义变量存储服务器返回结果，则不填写“输出键值对”

## 样例

步骤动作	期望的结果	执行
<pre>1 {   "请求头": {"Content-Type": "application/x-www-form-urlencoded", "charset": "utf-8"},   "方法": "post",   "url": "/cmorder/1/4",   "参数": {"serial": "[global_serial]", "openId": "[global_openId]", "fm": 1, "mallGoodsId": "26838", "amount": 1, "cartIds": "", "fetchTime": "", "remark": ""} }</pre>	<pre>{   "检查": "body",   "匹配规则": "包含字符串",   "条件": [{"模式": "\\success\\:true", "消息": "创建储值卡支付订单失败， success不为True"}],   "输出": {"success": "\\success\\:(.+?)", "attach": "attach\\:\\(.+?)\\"} }</pre>	手工 <span style="color: red;">✖</span> <span style="color: green;">+</span>

针对 http,https 请求，且步骤执行单个用例、不是执行 sql，按如下填写，json 格式

### 步骤动作

```
{
  "步骤类型": "执行用例",
  "用例 id": "106",
  "用例名称": "test-n-创建储值卡支付订单"
}
```

## 预期结果

为空，不填写

## 样例

#	步骤动作	期望的结果	执行
1	<pre>{   "步骤类型": "执行用例",   "用例id": "106",   "用例名称": "test-n-创建储值卡支付订单" }</pre>		手工 

针对步骤行 sql, 按如下填写:

#### 更新(UPDATE)

##### 步骤动作

##### json 格式

```
{
  "步骤类型": "执行 sql",
  "更新": "UPDATE mall_shopping_cart SET amount = 0 WHERE customer_id = %s and closed=1",
  "参数": "([CaseStep.customer_id],)"
}
```

##### 预期结果

为空, 不填写

#### 查询(SELECT)

##### 步骤动作

##### json 格式

```
{
  "步骤类型": "执行 sql",
  "单条查询": "SELECT id FROM customer WHERE channel_serial=%s",
  "参数": "(\\"[global_openId]\\",)"
}
```

**注: 参数值为元组的字符串类型表示 (value1,value2,……)**

##### 预期结果

##### json 格式

```
{"检查": "body",
"输出": {"customer_id": 1}
}
```

**注: 预期结果要么为空, 要么严格按照上述填写**

#### 样例

#	步骤动作	期望的结果	执行
1	<pre>{   "步骤类型": "执行sql",   "单条查询": "SELECT id FROM customer WHERE channel_serial=%s",   "参数": "(\"[global_openId]\",)" }</pre>	<pre>{   "检查": "body",   "输出": {"customer_id": 1} }</pre>	手工 <span style="color: red;">✖</span> <span style="color: green;">+</span>
2	<pre>{   "步骤类型": "执行sql",   "更新": "UPDATE mall_shopping_cart SET amount = 0 WHERE customer_id = %s and closed=1",   "参数": "([CaseStep.customer_id],)" }</pre>		手工 <span style="color: red;">✖</span> <span style="color: green;">+</span>

其它一些样例

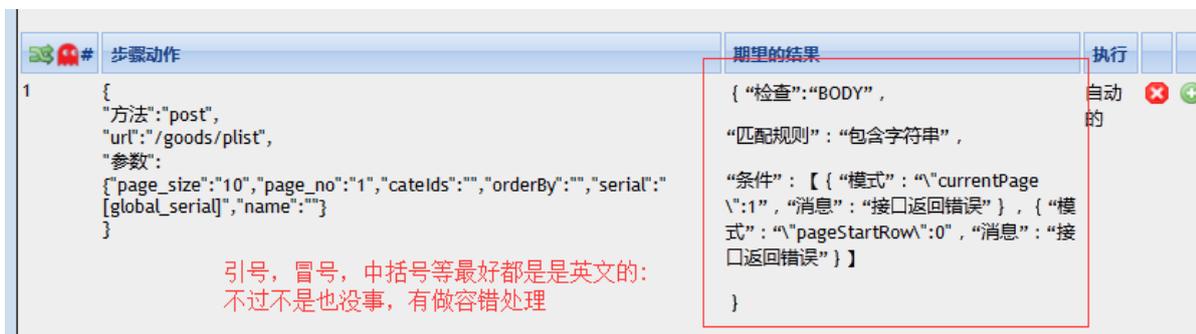
发送请求参数为 xml 格式

#	步骤动作	期望的结果	执行
1	<pre>{   "请求头": {"Content-Type": "text/xml", "charset": "utf-8"},   "函数": "test_interface_of_xml",   "方法": "post",   "url": "/WebServices/WeatherWebService.asmx?",   "参数": "&lt;soapenv:Envelope xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\" xmlns:web=\"http://WebXml.com.cn/\"&gt;     &lt;soapenv:Header/&gt;     &lt;soapenv:Body&gt;       &lt;web:getSupportProvince/&gt;     &lt;/soapenv:Body&gt;   &lt;/soapenv:Envelope&gt;" }</pre>	<pre>{   "检查": "body",   "匹配规则": "包含字符串",   "条件": [{"模式": "黑龙江", "消息": "返回结果不包含黑龙江"}] }</pre>	手工 <span style="color: red;">✖</span> <span style="color: green;">+</span>

发送请求参数为 json 格式



### 容错处理



## ② 参数化

### 1) 针对 https, http 请求, 且不是执行 sql, 也不是执行整个用例的步骤动作

如果参数位于“参数”、“请求体”, 且“参数”、“请求体”的值为字典, 按如下格式填写

"[var\_name]"

```
{
```

```
.....,
```

```
"参数": {"key1": "[global_var1]", "key2": "[global_var2]"}
}
```

```
}
```

```
{
```

```
.....,
```

```
"请求头": {"DeviceId": "[CaseStep.device_no]", "SerialNo": "[CaseStep.serial_no]",
```

```
"Content-Type":"application/json;charset=utf-8","ProductVersion":"[global_product_version]","ProtocolVersion":"[global_product_version]","OperatorId":"[OperatorId]","Token":"[Token]",  
.....  
}
```

如果参数位于“url”，按如下格式填写

```
[var_name]  
{  
.....,  
"url":"/kq/getincoupon/40758966216286146560/[global_openId]",  
.....  
}
```

## 2) 针对执行 sql 的步骤动作

按如下格式填写

```
[var_name]或者 \"[var_name]\"  
{  
.....,  
"参数":\"(\\"[global_openId]\",)\"  
.....  
}
```

注: 如果待替换“动态变量” [var\_name]为字符串类型的变量, 需要在其左右两侧加 \" 符号, 形如 \"[global\_openId]\"

## 3) 变量划分

全局变量[global\_var], 来自 globalpkg.global\_var.py 中定义的, var\_name 和定义的全局变量名称保持一致即可

非全局变量[ClassName.var\_name], 可能来自 sql 查询结果, 也可能来自 http(s)请求返回结果中提取的

A) “步骤动作”中的非全局变量来源于某个 sql 查询结果, 则按如下格式填写

```
[CaseStep.var_name]
```

样例

#	步骤动作	期望的结果	执行
1	<pre>{   "步骤类型": "执行sql",   "单条查询": "SELECT id FROM customer WHERE channel_serial=%s",   "参数": "(\"[global_openId]\",)", }</pre>	<pre>{   "检查": "body",   "输出": {"customer_id": 1} }</pre>	手工  
2	<pre>{   "步骤类型": "执行sql",   "更新": "UPDATE mall_shopping_cart SET amount = 0 WHERE customer_id = %s and closed=1",   "参数": "([CaseStep.customer_id],)", }</pre>		手工  

B) “步骤动作”中的非全局变量来源于某个 http(s) 请求返回结果，则按如下格式填写

**[ClassName.var\_name]、[ClassName.var\_name\_N]**

样例

wecharno-8:test-n-创建储值卡支付订单

警告! : 这个测试用例的版本已经被执行过

版本 1

摘要

前提

#	步骤动作	预期的结果
1	<pre>{   "请求头": {"Content-Type": "application/x-www-form-urlencoded", "charset": "utf-8"},   "方法": "post",   "url": "/cmorder/1/4",   "参数": {"serial": "[global_serial]", "openId": "[global_openId]", "fm": 1, "mallGoodsId": "26838", "amount": 1, "cartIds": "", "fetchTime": "", "remark": ""} }</pre>	<pre>{ "检查": "body",   "匹配规则": "包含字符串",   "条件": [{"模式": "\\success\\": true, "消息": "创建储值卡支付订单失败, success不为True"}],   "输出": [{"success": "\\success\\": "(.+?)", "attach": "attach\\": "(.+?)\\\""}] }</pre>
1	<pre>{   "步骤类型": "执行用例",   "用例id": 106,   "用例名称": "test-n-创建储值卡支付订单" }</pre>	
2	<pre>{   "方法": "get",   "url": "/pay/5?",   "参数": {"orderId": "[attach_1]", "serial": "[global_serial]"} }</pre>	<pre>{ "检查": "body",   "匹配规则": "包含字符串",   "条件": [{"模式": "确认支付", "消息": "提交储值卡支付订单-pay5重定向失败, 没确认支付按钮"},   {"模式": "储值卡", "消息": "提交储值卡支付订单-pay5重定向失败, 没打开页面"}] }</pre>

注:

1、当调用的函数为默认类名, 即 `InterfaceUnittestTestCase` 类中定义的方法时, 这里“`ClassName.`”可以不写, 如果是调用其它新增类中的方法时, 需要显示的填写

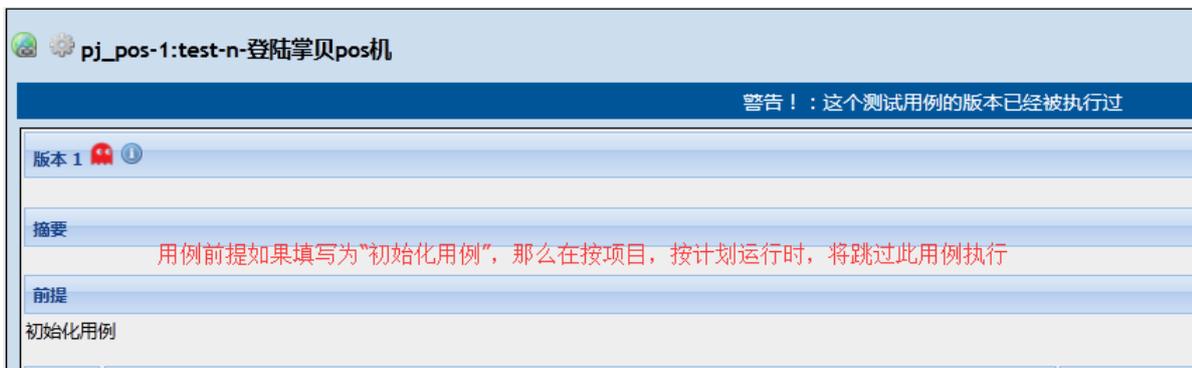
2、当且仅当变量是通过正则表达式提取的时候, 采用`[ClassName.var_name_N]`, 的形式 N 从 1, 2, 3, …… 以此类推

#### 4) 结果断言和服务器返回内容提取

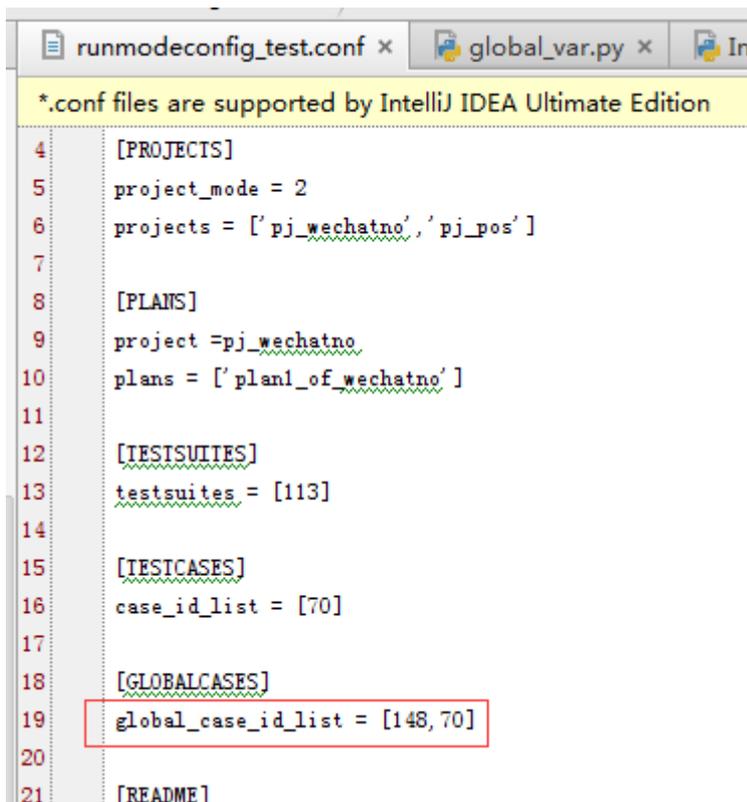
参考“结果断言和提取服务器返回结果.txt”文档

### ③ 用例执行依赖

全局初始化用例



注: 全局初始用例在 runmodeconfig\_运行环境.conf 中进行配置, 优先于其它用例被执行



当把某个用例作(假设为“用例 1”)为其它用例的某个步骤, 不能满足需求(比如提供的接口输入参数值不一样)时, 可以复用“用例 1”的步骤信息并对参数进行适当的调整

### ④ 禁用用例



注意: 被禁用的用例不参与执行, 计划, 项目也是如此

## 7、运行结果

见源码附件

## 8、源码下载

下载地址: 目前只限群内分享, 欢迎加入 QQ 群: 7156436 于群共享获取

下载后解压, 用 pycharm 导入项目即可

## 9、说明

时间有限, 精力有限, 暂且就到这吧, 有需要的可以自己扩展、修改框架。

注: 目前还存在个 bug, 测试报告中, 类似 xml 格式参数的参数值没显示出来, 有兴趣的烦先自己解决下。