

## 基于 python+Testlink+Jenkins 实现的接口自动化测试框架 V4.0

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.....	4
4、 数据库的创建.....	6
5、 框架模块详细介绍.....	6
6、 Testlink 相关配置与用例管理.....	14
a) API 相关配置.....	14
b) 项目相关配置.....	18
c) 用例管理.....	20
① 步骤动作和预期结果填写规范.....	21
② 参数化.....	25
③ 用例执行依赖.....	29
④ 禁用用例.....	29
7、 运行结果.....	30
8、 源码下载.....	31
9、 说明.....	31

### 1、 开发环境

win7\Windows Server 2008 R2 x64

PyCharm 4.0.5

setuptools-29.0.1.zip

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

官方下载地址: <https://pypi.python.org/pypi/setuptools#downloads>

python 3.3.2

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

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

mysql-connector-python-2.1.4-py3.3-win64.msi

下载地址: <http://pan.baidu.com/s/1cDtP10>

官方下载地址: <http://dev.mysql.com/downloads/connector/python/>

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](#)”
- “[利用 Build With Parameters Plugin 实现 Jenkins 参数化构建](#)”
- “[利用 HTML Publisher plugin 实现 HTML 文档报告展示](#)”

如果需要支持参数化构建, 需要替换 `main.py`, `globalpkg/global_function.py`, `config/runmodeconfig.py` 文件

Windows batch, `main.py` 参数按如下格式写, 注意顺序必须遵守如下:

```
cd /d C:\Projects\interface_project_for_test
python
main.py %run_env% %runmode% %project_mode% %projects% %project% %plans% %testsuites% %case_id_list% %global_case_str%
```

完整批处理参考: `jenkins 集成 html 报告显示后的 windows 构建批处理命令.txt`

## Project interface\_auto\_test\_for\_test

需要如下参数用于构建项目:

run_env	1	1 - 在测试环境运行; 2 - 在预发布环境运行
runmode	1	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
project_mode	2	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
projects	['pj_wechatno']	按项目运行时, 需要指定项目及项目关联的测试项目
project	pj_wechatno	按计划运行时, 需要指定计划关联的项目
plans	['plan1_of_wechatno']	按计划运行时, 需要指定项目及项目关联的测试计划
testsuites	"162:微商城 125:免费领取卡券接口"	按套件运行,格式: 套件1id:套件1名称套件2id:套件2名称
case_id_list	[70]	[testcase_id1, testcase_id2,...,testcase_idN].按指定用例时, 需要指定需要运行的用例id
global_case_str	"pj_pos  148:test-n-登陆掌贝pos机#pj_shop  969:登录商户后台 275:打开订单管理页面"	格式: 项目名称1  用例id1:用例名称#项目名称2  用例id2:用例名称2 用例id3:用例名称3

开始构建

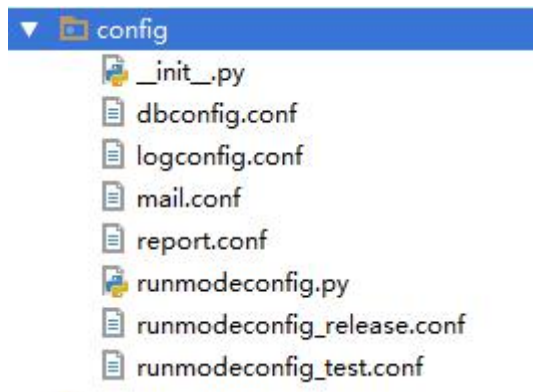
注意: 集成里面的参数默认值只能为英文的, 另外, 字符必须加双引号

#### 4、数据库的创建

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

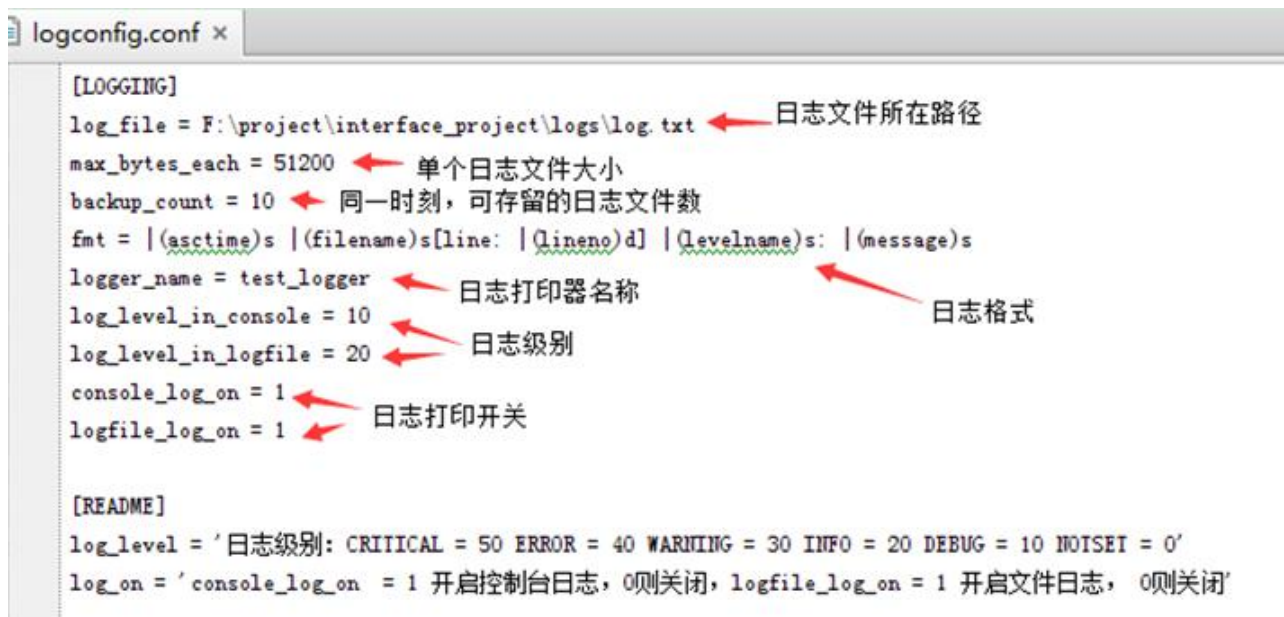
#### 5、框架模块详细介绍

##### 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:



**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



```
[RUNMODE]
runmode = 3

[PROJECTS]
project_mode = 2
projects = [ 'pj_shop', 'pj_wechatno' ]

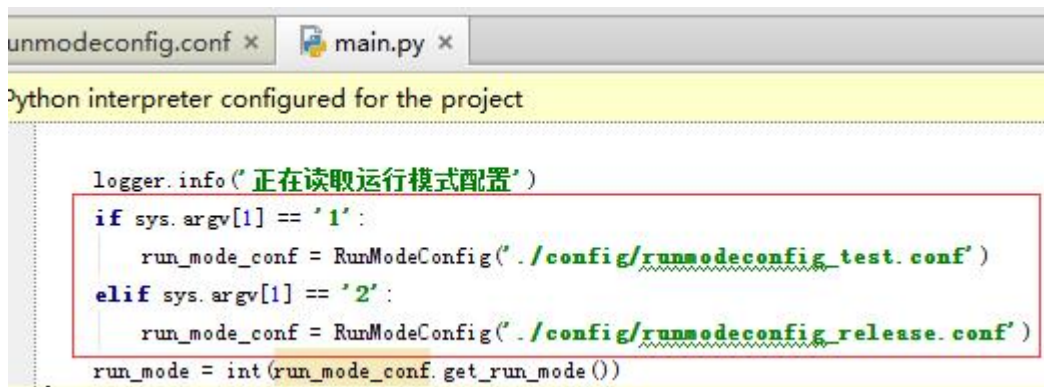
[PLANS]
project = pj_wechatno
plans = [ 'plan1_of_wechatno', 'pj_wechatno' ]

[TESTSUITES]
testsuites = 147:pj-pos登录接口

[TESTCASES]
case_id_list = [148]

[GLOBALCASES]
global_cases_str = pj_pos||148:test-n-登陆宝贝pos机#pj_shop||969:登录商户后台|275:打开订单管理页面

[README]
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'
testsuites = 162:微商城|125:免费领取卡券接口, 格式: 套件1id:套件1名称套件2id:套件2名称
plans = '按计划运行时, 需要指定项目及项目关联的测试计划'
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'
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'
global_cases_str = 需要优先运行的全局初始化用例 格式: 项目名称1||用例id1:用例名称#项目名称2||用例id2:用例名称2|用例id3:用例名称3
```

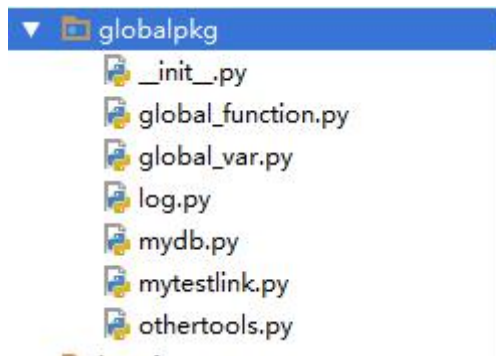


```
logger.info('正在读取运行模式配置')
if sys.argv[1] == '1':
    run_mode_conf = RunModeConfig('./config/runmodeconfig_test.conf')
elif sys.argv[1] == '2':
    run_mode_conf = RunModeConfig('./config/runmodeconfig_release.conf')
run_mode = int(run_mode_conf.get_run_mode())
```

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

```
# tag1
result_list = re.findall('<\/.?>', data) # 查找类似这类内容 <\/pre>、<\/div> 并替换为空
if result_list:
    for item in result_list:
        data = data.replace(item, '')

result_list = re.findall('<?.?[^<]*>', data) # 查找类似这类内容 <pre style="background:
if result_list:
    for item in result_list:
        data = data.replace(item, '')

# 如果是webservice接口, 则tag1往后, 及此之前的代码注释掉, 把如下代码开头和结尾的""去掉
"""result_list = ['<\/p>', '<\/ol>', '<\/div>', '<\/span>', '<\/strong>', '<\/em>', '<\/u>', '<\/g
for item in result_list:
    if data.find(item):
        data = data.replace(item, '')

result_list = []
patterns = ['<br', '<div', '<span', '<li', '<pre', '<p', '<\/ol', '<strong', '<em', '<u',
for pattern in patterns:
    pattern = pattern + '.*?[^<]*>'
    result_list = result_list + re.findall(pattern, data)

if result_list:
    for item in result_list:
        data = data.replace(item, '')"""
```

**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__ = 'lai fuyu'

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

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-MkNilyCI5CY'}}
    global_serial = '10549840601068216320'
    global_openId = 'ous64uFCCLMyXYDJ-MkNilyCI5CY'
    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-MkNilyCI5CY'}}
    global_serial = '10549840601068216320'
    global_openId = 'ous64uFCCLMyXYDJ-MkNilyCI5CY'
    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'}}

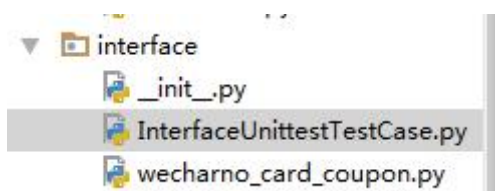
**global\_function.py:** 主要提供全局函数，比如提供运行单个用例的函数 `run_testcase_by_id` 等

### c) logs 及 testreport

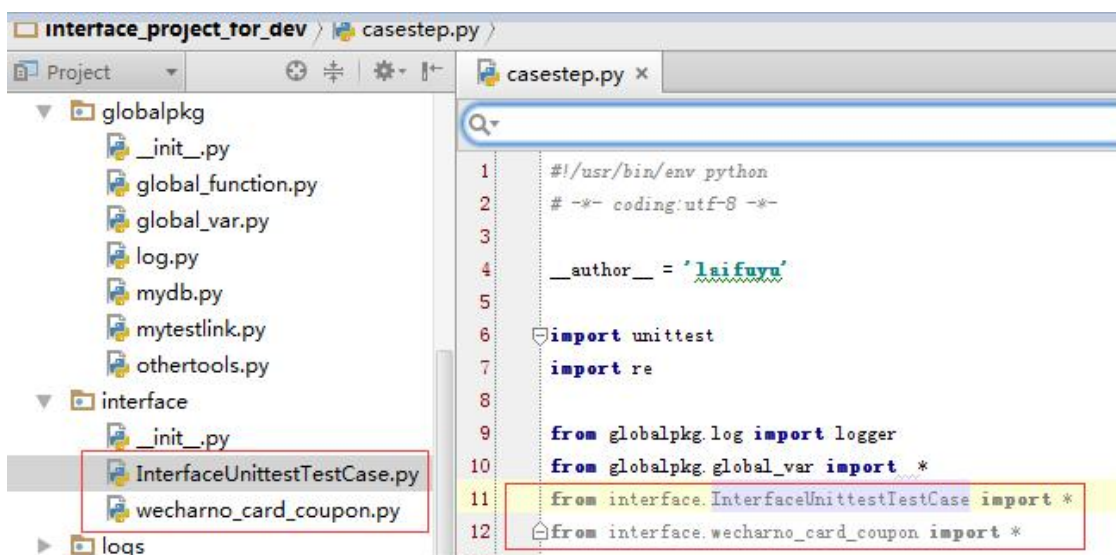
可分别用于存放日志文件，测试报告

### 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



接口单元测试类代码说明





说明: 新版本代码中,

`outputs_list = []` 已经改成 `outputs_dict = {}`

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)

```
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)
```

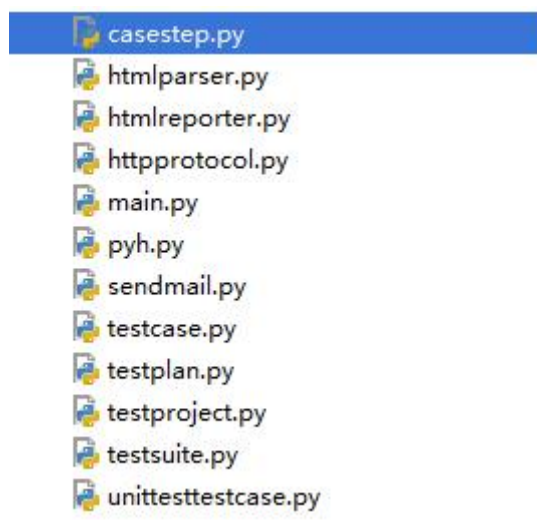
问题:假如要进行数据库后台的逻辑校验咋办?

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。

或者把逻辑写成存储过程,扩展代码,增加存储过程调用

### e) 其它模块

如下, 顾名思义

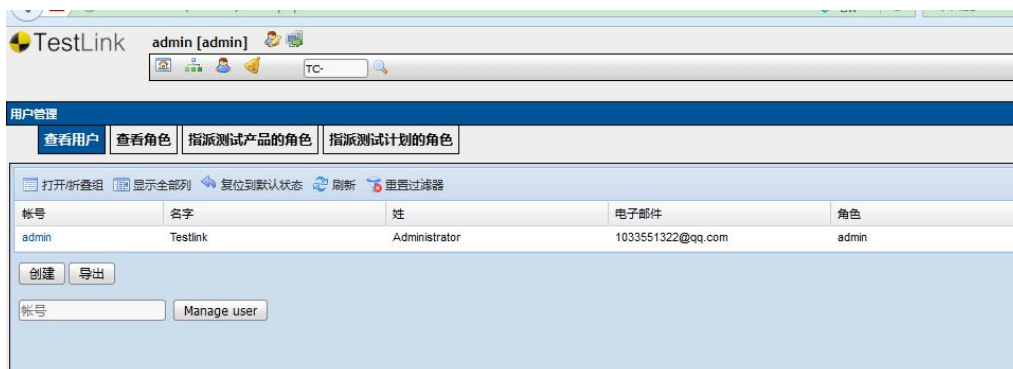


## 6、Testlink 相关配置与用例管理

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

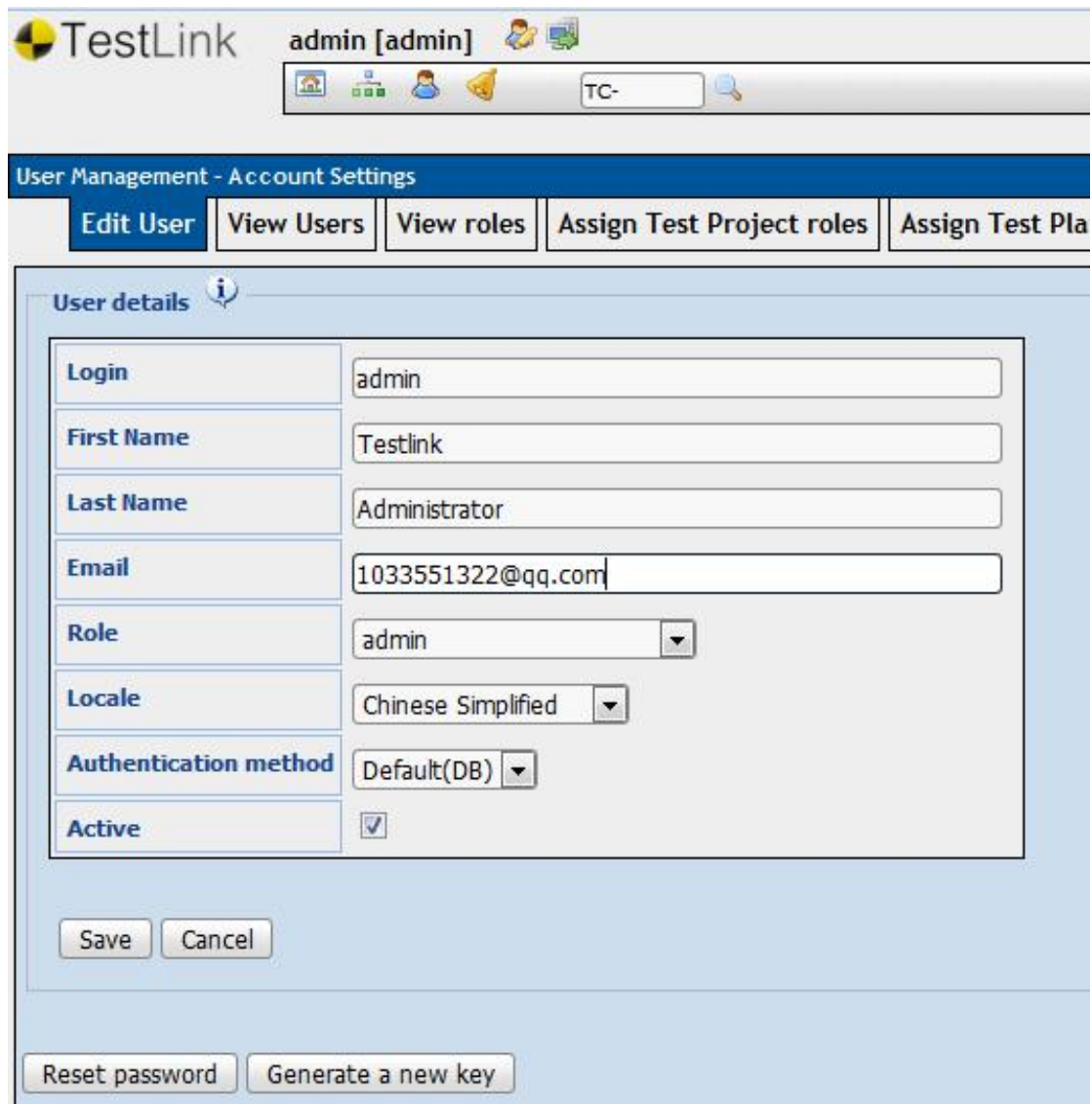
### a) API 相关配置

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 web interface for user management. At the top, the TestLink logo is on the left, and the user 'admin [admin]' is on the right. Below the header is a navigation bar with tabs: 'Edit User' (selected), 'View Users', 'View roles', 'Assign Test Project roles', and 'Assign Test Pla'. The main content area is titled 'User details' and contains a form for editing the user 'admin'. The form fields are: 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 page are 'Reset password' and 'Generate a new key' buttons.

Login	admin
First Name	Testlink
Last Name	Administrator
Email	1033551322@qq.com
Role	admin
Locale	Chinese Simplified
Authentication method	Default(DB)
Active	<input checked="" type="checkbox"/>

Save Cancel

Reset password Generate a new key



The screenshot shows the TestLink interface for API access and login history. At the top is a '修改密码' (Change Password) button. Below is the 'API 接口' (API Interface) section, which shows '个人 API 访问密钥 = 无' (Personal API access key = None) and a '生成新的密钥' (Generate new key) button. At the bottom is the '登录历史' (Login History) section, which shows '上次成功登录' (Last successful login) and a log entry: '2016-03-19 22:14:58 从 '192.168.1.101' 登录 'admin' from '192.168.1.101' 成功'.

修改密码

**API 接口**

个人 API 访问密钥 = 无

生成新的密钥

**登录历史**

上次成功登录

2016-03-19 22:14:58 从 '192.168.1.101' 登录 'admin' from '192.168.1.101' 成功

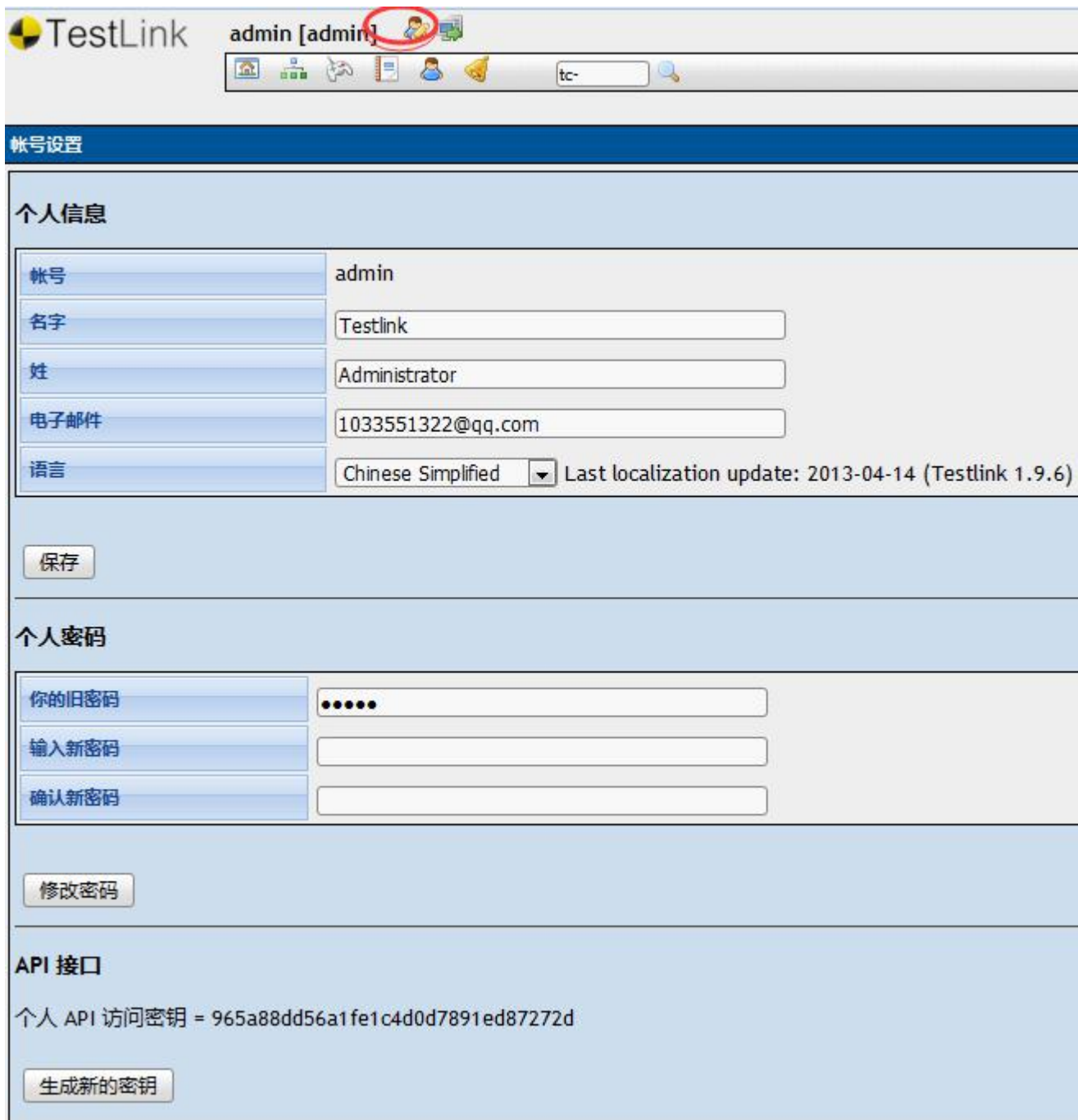
点击生成新的密钥，如下

### 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



The image shows the TestLink user profile page. At the top, there is a navigation bar with the TestLink logo and a user profile dropdown menu showing 'admin [admin]'. Below this is a search bar with the text 'tc-'. The main content area is titled '帐号设置' (Account Settings) and is divided into three sections: '个人信息' (Personal Information), '个人密码' (Personal Password), and 'API 接口' (API Interface). The '个人信息' section contains a table with fields for '帐号' (Username: admin), '名字' (Name: Testlink), '姓' (Surname: Administrator), '电子邮件' (Email: 1033551322@qq.com), and '语言' (Language: Chinese Simplified). Below this table is a '保存' (Save) button. The '个人密码' section contains three input fields: '你的旧密码' (Your old password), '输入新密码' (Enter new password), and '确认新密码' (Confirm new password), followed by a '修改密码' (Change password) button. The 'API 接口' section contains the text '个人 API 访问密钥 = 965a88dd56a1fe1c4d0d7891ed87272d' and a '生成新的密钥' (Generate new key) button.

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="Last localization update: 2013-04-14 (Testlink 1.9.6)"/>

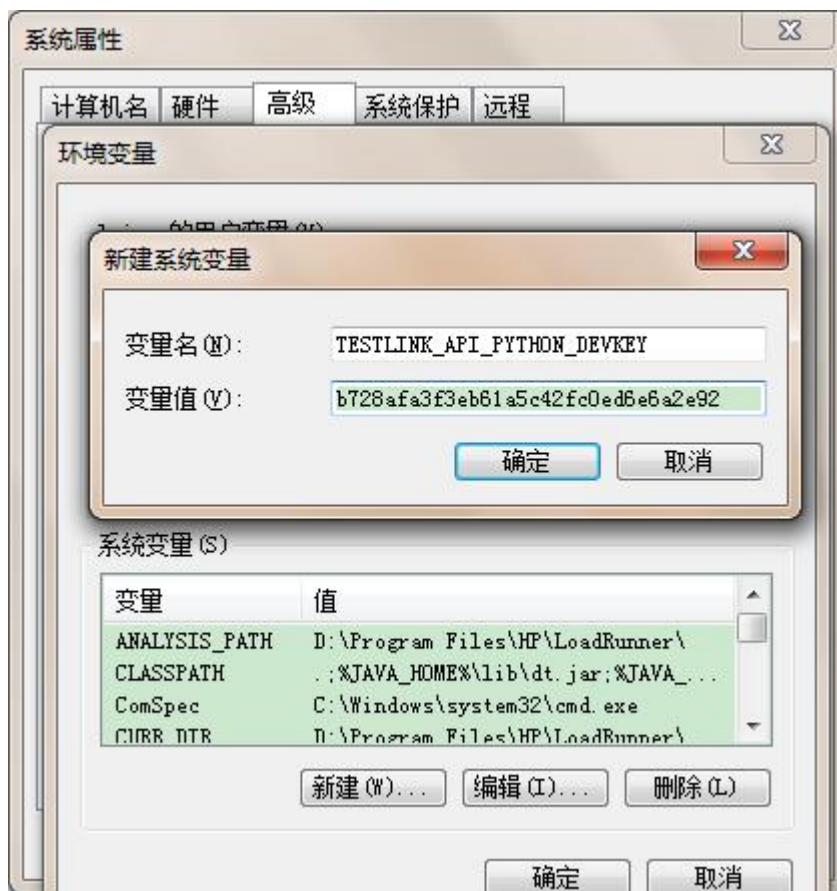
个人密码	
你的旧密码	<input type="password" value="....."/>
输入新密码	<input type="password"/>
确认新密码	<input type="password"/>

**API 接口**

个人 API 访问密钥 = 965a88dd56a1fe1c4d0d7891ed87272d

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\testlink\testlinkhelper.py

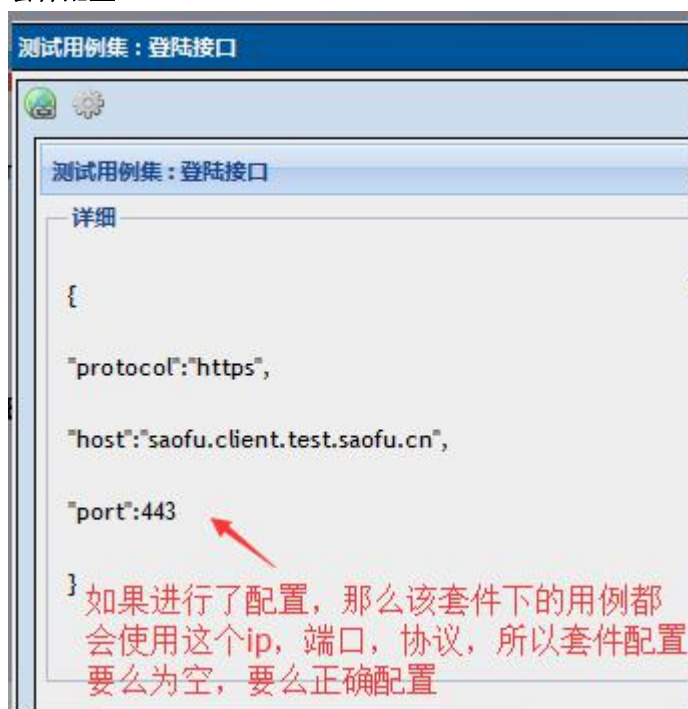
## b) 项目相关配置

测试项目配置



注: 计划, 也存在是否开启的配置, 需要开启才会运行。

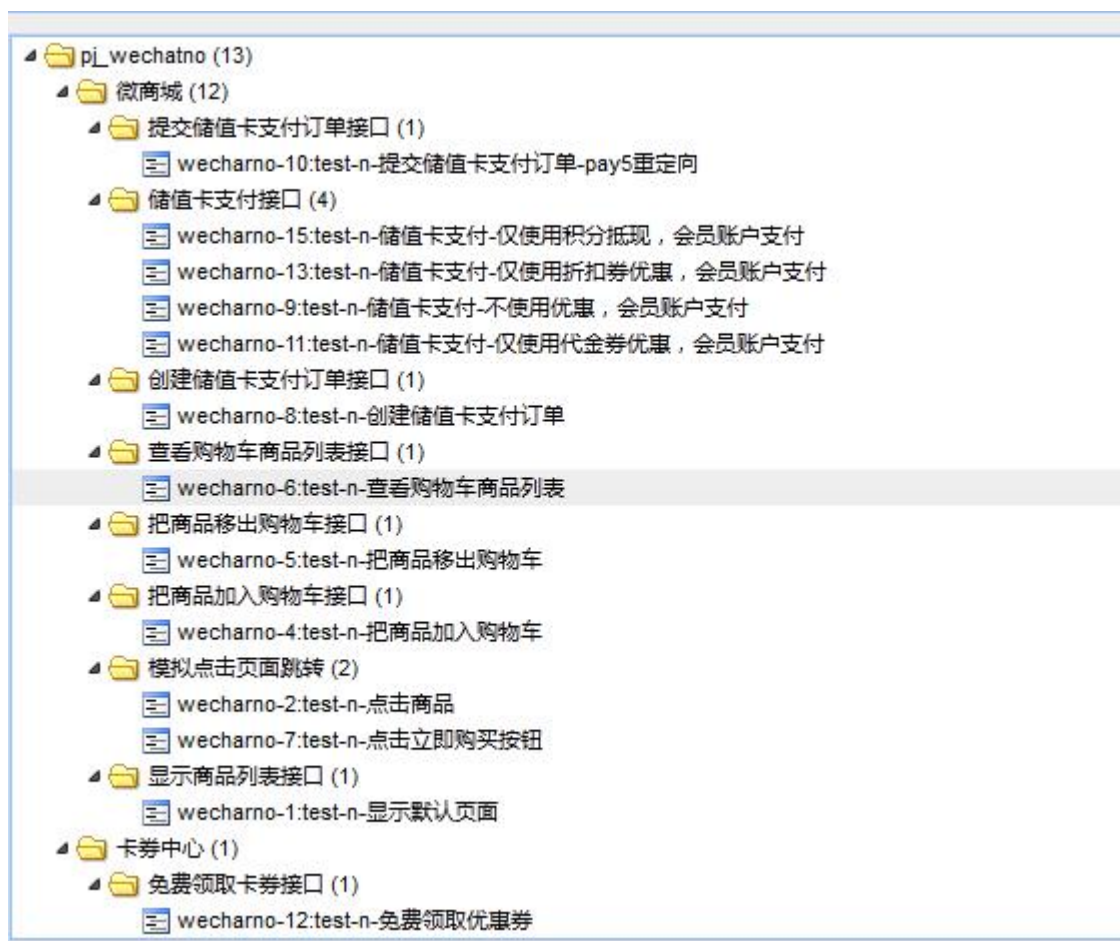
### 套件配置





### 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_urlencode",
  "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\_urlencode
- 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”，有必要的话也是双引号包含，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": {"re": {"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”
- 3、注：匹配规则和条件，要么都写，要么都不写，如下

```
{
  "检查": "body",
  "输出": {"re": {"success": "\"success\":(.+?)", "attach": "attach\": \"(.+?)\""} }
}
```

## 样例

步骤动作	期望的结果	执行
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": ""}} }	{ "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\\":\"(.+?)\""} }	手工  

注: 4.0 中“输出”增加了提取方式, 见上文

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 { "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" }		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",
  "数据库": "db_name",
  "更新": "UPDATE mall_shopping_cart SET amount = 0 WHERE customer_id = %s and closed=1",
  "参数": "([CaseStep.customer_id],)"
}
```

**注意:**

- 1、如果“参数”值为空，则填写为“参数":""
- 2、“数据库”，要从那个数据库查数据，其值来源于 global\_var 里面的数据库对象名称。如下图



注意: 这里如果添加了数据库对象, 需要在 main.py 近末尾处中添加关闭数据库连接的代码 db\_name.close()

## 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",
  "数据库": "db_name",
  "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,...), 如果参数为空则 "参数": "", 如果参数值为字符串类型的, 则按这样填写: (\\"参数值\\",)

## 预期结果

#### json 格式

```

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

或者如下

```

{"检查": "body",
  "输出": {"customer_id": 1},
  "匹配规则": "相等",
}
    
```

"条件": [{"模式": "[CaseStep.device\_no]=123456", "消息": "登录失败, device\_no 不为 123456"}]  
}

**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
1	<pre>{   "步骤类型": "执行sql",   "数据库": "saofudb",   "单条查询": "SELECT branch_id FROM account_operator WHERE id = %s",   "参数": "(1318,)" }</pre>	<pre>{   "检查": "body",   "输出": {"branch_id": 1} }</pre>
2	<pre>{   "步骤类型": "执行sql",   "数据库": "saofudb",   "单条查询": "SELECT device_no, serial_no FROM device WHERE shop_branch_id = %s",   "参数": "([CaseStep.branch_id],)" }</pre>	<pre>{   "检查": "body",   "输出": {"device_no": 1, "serial_no": 2} }</pre>

其它一些样例

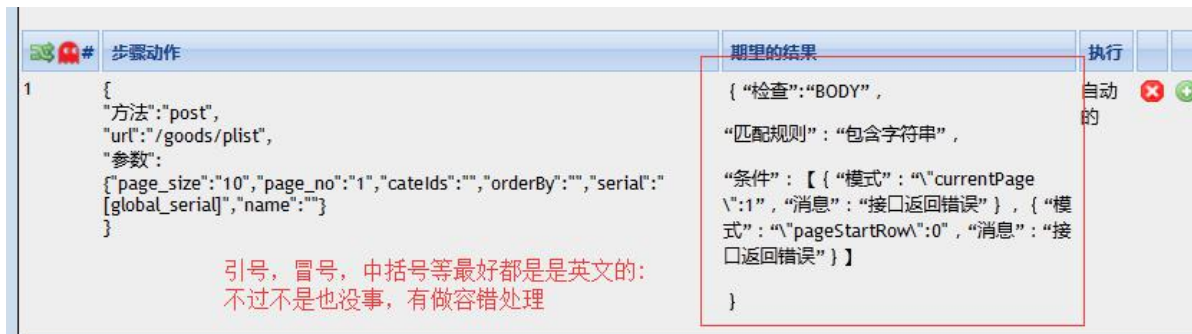
发送请求参数为 xml 格式

#	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、预期结果

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”或者预期结果“条件”中的字典 key 对应的值，按如下格式填写

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

{
.....,
"条件": [{"模式": {"key": "[CaseStep.branch_id]"},"消息": "key 错误"}]
.....
}
```

注意：所有变量名均不能为纯数字，比如[1]，因为需要支持 xpath 提取，这类纯数字的都视为非变量

## 2) 针对执行 sql 的步骤动作、预期结果

参数位于步骤动作、步骤预期结果，按如下格式填写

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

{
"检查":"body",
"匹配规则":"相等",
"条件": [{"模式": "[CaseStep.device_no]=123456", "消息": "登录失败, device_no 不为 123456"}]
}
```

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
1	<pre>{   "步骤类型": "执行sql",   "数据库": "saofudb",   "单条查询": "SELECT branch_id FROM account_operator WHERE id = %s",   "参数": "(1318,)" }</pre>	<pre>{   "检查": "body",   "输出": {"branch_id": 1} }</pre>
2	<pre>{   "步骤类型": "执行sql",   "数据库": "saofudb",   "单条查询": "SELECT device_no, serial_no FROM device WHERE shop_branch_id = %s",   "参数": "([CaseStep.branch_id],)" }</pre>	<pre>{   "检查": "body",   "输出": {"device_no": 1, "serial_no": 2} }</pre>

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

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	期望的结果
<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>
<pre>1 {   "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>	
<pre>2 {   "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:

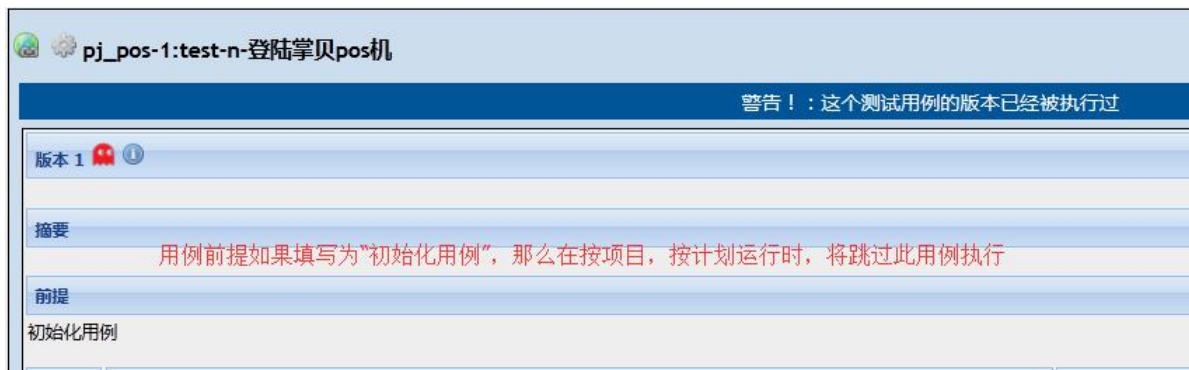
- 1、当调用的函数为默认类名, 即 `InterfaceUnittestTestCase` 类中定义的方法时, 这里 “ClassName.” 可以不写, 如果是调用其它新增类中的方法时, 需要显示的填写
- 2、当且仅当变量是通过正则表达式提取的时候, 采用 `[ClassName.var_name_N]`, 的形式 N 从 1, 2, 3, …… 以此类推

#### 4) 结果断言和服务端返回内容提取

参考 “结果断言和提取服务端返回结果.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

```
[PROJECTS]
project_mode = 2
projects = [ 'pj_shop', 'pj_wechatno' ]

[PLANS]
project = pj_wechatno
plans = [ 'plan1_of_wechatno', 'pj_wechatno' ]

[TESTSUITES]
testsuites = 162:微商城|125:免费领取卡券接口

[TESTCASES]
case_id_list = [148]

[GLOBALCASES]
global_cases_str = pj_pos||148:test-n-登陆掌贝pos机#pj_shop||969:登录商户后台|275:打开订单管理页面

[README]
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'
testsuites = 162:微商城|125:免费领取卡券接口, 格式: 套件1id:套件1名称套件2id:套件2名称
plans = '按计划运行时, 需要指定项目及项目关联的测试计划'
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'
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'
global_cases_str = 需要优先运行的全局初始化用例 格式: 项目名称1||用例id1:用例名称#项目名称2||用例id2:用例名称2|用例id3:用例名称3
```

注意：没有则全局初始化用例保持为空 global\_cases\_str =

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

### interface\_autotest\_report

测试总耗时: 0:00:24.085058

用例总数: 23    成功用例数(Pass): 15    失败用例数(Fail): 3    出错用例数(Error): 1    未执行用例数(Block): 4

#####用例执行摘要#####

###测试计划【项目名称: pj\_pos, 计划名称: [plan1 of pos](#)】

ID	执行编号	用例ID	用例外部ID	用例名称	测试套件	执行结果	运行时间
1588	20170308232635	152	<a href="#">pj_pos-2</a>	test-n-修改积分	扫码-修改积分接口	Fail	2017-03-08 23:26:37

###测试计划【项目名称: pj\_pos\_eshop, 计划名称: [plan1 of wechatno](#)】

ID	执行编号	用例ID	用例外部ID	用例名称	测试套件	执行结果	运行时间
1589	20170308232635	711	<a href="#">mall-77</a>	联系电话查询	订单中心-搜索查询	Fail	2017-03-08 23:26:39
1590	20170308232635	708	<a href="#">mall-76</a>	联系人查询	订单中心-搜索查询	Fail	2017-03-08 23:26:39
1591	20170308232635	714	<a href="#">mall-78</a>	订单来源(桌号)查询	订单中心-搜索查询	Pass	2017-03-08 23:26:40
1592	20170308232635	705	<a href="#">mall-75</a>	订单号查询	订单中心-搜索查询	Pass	2017-03-08 23:26:41

###测试计划【项目名称: pj\_wechatno, 计划名称: [plan1 of wechatno](#)】

ID	执行编号	用例ID	用例外部ID	用例名称	测试套件	执行结果	运行时间
1593	20170308232635	93	<a href="#">wecharno-5</a>	test-n-把商品移出购物车	微商城-把商品移出购物车接口	Pass	2017-03-08 23:26:43
1594	20170308232635	88	<a href="#">wecharno-4</a>	test-n-把商品加入购物车	微商城-把商品加入购物车接口	Pass	2017-03-08 23:26:43
1595	20170308232635	126	<a href="#">wecharno-12</a>	test-n-免费领取优惠券	卡券中心-免费领取卡券接口	Error	2017-03-08 23:26:46
1596	20170308232635	195	<a href="#">wecharno-17</a>	微信商城首页	微信商城-微信端-微信商城-打开页面	Pass	2017-03-08 23:26:47
1597	20170308232635	98	<a href="#">wecharno-6</a>	test-n-查看购物车商品列表	微商城-查看购物车商品列表接口	Pass	2017-03-08 23:26:47
1598	20170308232635	120	<a href="#">wecharno-11</a>	test-n-储值卡支付-仅使用代金券优惠, 会员账户支付	微商城-储值卡支付接口	Block	2017-03-08 23:26:48
1599	20170308232635	80	<a href="#">wecharno-2</a>	test-n-点击商品	微商城-模拟点击页面跳转	Pass	2017-03-08 23:26:48
1600	20170308232635	76	<a href="#">wecharno-1</a>	test-n-显示默认页面	微商城-显示商品列表接口	Pass	2017-03-08 23:26:49
1601	20170308232635	88	<a href="#">wecharno-4</a>	test-n-把商品加入购物车	微商城-把商品加入购物车接口	Pass	2017-03-08 23:26:50
1602	20170308232635	106	<a href="#">wecharno-8</a>	test-n-创建储值卡支付订单	微商城-创建储值卡支付订单接口	Pass	2017-03-08 23:26:54
1603	20170308232635	110	<a href="#">wecharno-9</a>	test-n-储值卡支付-不使用优惠, 会员账户支付	微商城-储值卡支付接口	Block	2017-03-08 23:26:52



#####用例执行明细#####

##测试计划【项目名称: pj\_pos, 计划名称: plan1\_of\_pos】

>>>测试用例【用例外部ID: pj\_pos-2, 名称: test-n修改积分】

步骤	协议方法	协议	主机	端口	ACTION	预期结果	运行结果	原因分析	运行时间
1	post	https	192.168.1.100	443	{函数: 'test_interface_of_json', '请求头': {'Content-Type': 'application/json;charset=utf-8', 'DeviceId': '[CaseStep.device_no]', 'ProtocolVersion': '3.2.12C', 'ProductVersion': '3.2.12C', 'Token': '[Token]', 'SerialNo': '[CaseStep.serial_no]', 'OperatorId': '[OperatorId]', '方法': 'post', 'url': 'http://192.168.1.100:443/api/v3', '参数': {'bonus': 37593, 'code': '17817738178'}}	{'条件': [{'消息': '修改积分失败', '模式': 'success:true'}], '检查': 'body', '匹配规则': '包含字符串'}	Fail	Traceback (most recent call last): File E:\Projects\interface_project_for_dev\interface\interfaceUnitTestCase.py, line 133, in test_interface_of_json self.assert_result(body) File E:\Projects\interface_project_for_dev\unittestcase.py, line 70, in assert_result self.assertIn(pattern_str, response_to_check, item[消息])AssertionError: success:true not found in {success:false,code:4000,message:Server Error} : 修改积分失败	2017-03-08 23:26:37

##测试计划【项目名称: pj\_pos\_eshop, 计划名称: plan1\_of\_wechatno】

>>>测试用例【用例外部ID: mail-77, 名称: 联系电话查询】

步骤	协议方法	协议	主机	端口	ACTION	预期结果	运行结果	原因分析	运行时间
1	post	http	192.168.1.100	80	{'方法': 'post', '函数': 'test_interface_of_json', 'url': 'http://192.168.1.100:80/api/v3', '参数': {'key': '135601564', 'page': '1'}}	{'条件': [{'消息': '搜索联系电话135601564,查询失败', '模式': '{code: '4001'}'], {'消息': '搜索联系电话135601564,查询失败', '模式': '{orderList: [{sourceMobile: '13560156409'}], '消息': '搜索联系电话135601564,查询失败', '模式': '{success: True}'}], '检查': 'body', '匹配规则': '键值相等'}	Fail	Traceback (most recent call last): File E:\Projects\interface_project_for_dev\interface\interfaceUnitTestCase.py, line 133, in test_interface_of_json self.assert_result(body) File E:\Projects\interface_project_for_dev\unittestcase.py, line 118, in assert_result self.assertEqual(dict_level_list[len(dict_level_list)-1], last_value, item[消息])AssertionError: 13560156409 != None : 搜索联系电话135601564,查询失败	2017-03-08 23:26:39

8、源码下载

下载地址:

下载后解压，用 pycharm 导入项目即可，必要时可能需要修改.idea/workspace.xml 中 python 程序所在路径

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。