基于 python 实现的 http+json 协议接口自动化测试框架（实用改进版）

by:授客 QQ：1033553122

私人博客：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

win7 64 位
JetBrains PyCharm 4.0.5
Python 3.3.5

MariaDB-5.5.45-centos6-x86_64
文件下载地址：http://pan.baidu.com/s/1sj1Lzw5

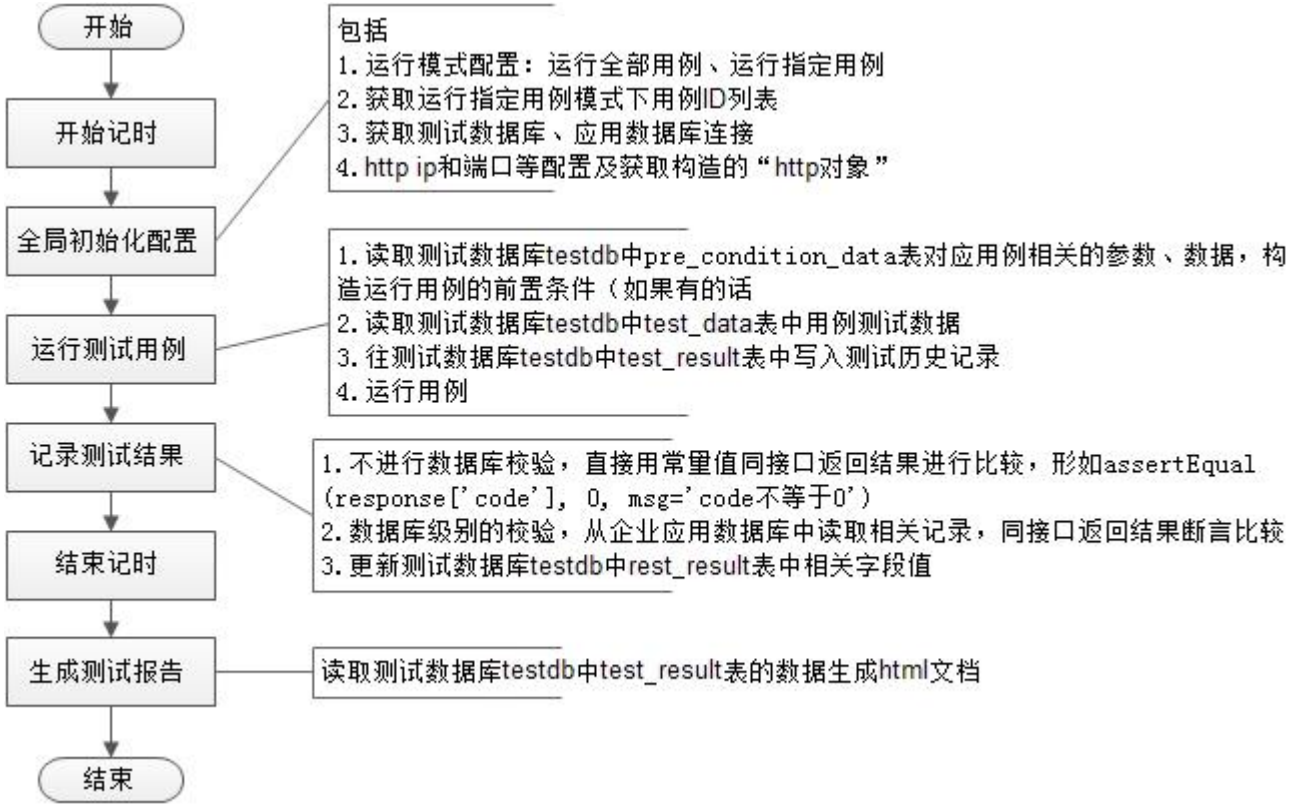
CentOS 6.5-x86_64
下载地址：http://www.centoscn.com/CentosSoft/iso/2013/1205/2196.html

Mysql Connector/Python Windows (x86, 64-bit), MSI Installer Python 3.3
下载地址：http://dev.mysql.com/downloads/connector/python/

其它：公司 Linux mysql 数据库服务器、应用服务器

2、 大致流程

下图展示了框架实现的业务流程



3、 框架简介

1、可通过配置文件 http_config.ini，对要测试接口服务器的 IP、域名，和端口信息进行灵活配置。

2、可通过配置文件 db_config.ini，对测试数据库，应用数据库服务器主机 IP，端口，用户名，密码等灵活配置。

3、可通过配置文件 run_case_config.ini 灵活配置需要用例运行模式，需要运行的用例 ID 列表

4、对常见 HTTP 的 POST，GET 请求方法进行封装（支持自由扩展以便增加其它方法

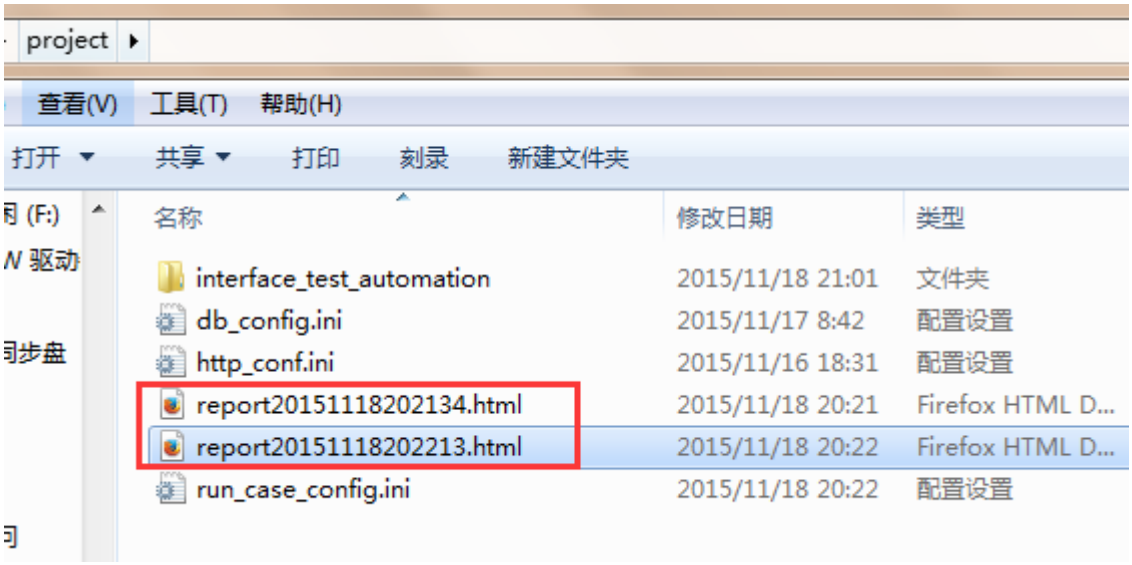
5、支持 JSON(含嵌套对象格式的 json 数据,如{ "orderTotalPrice":95, "goods":[{"shopId":987654354,"goodsId":108, "goodsNumber":1}]}})格式数据提交

6、通过数据库对接口测试用例、前置(数据)条件进行管理，可做到每个用例之间相互独立，互不依赖

7、针对接口返回结果，支持数据库级别的数据校验

8、可按测试时间及给定文件名，生成对应时间的 html 可视化报告，报告内容包含测试耗时，测试执行用例总数，执行成功、失败、出错用例数统计；还有单个用例的执行情况(ID,用例名称，(自定义)接口名称，接口 URL,接口参数，运行结果等)

4、 运行结果展示



test report

测试总耗时：0:00:00.134008

测试用例数：2 成功用例数：2 失败用例数：0 出错用例数：0

用例ID	HTTP方法	接口名称	请求URL	请求参数/数据	测试方法	测试描述	测试结果	失败原因
1	GET	login-normal	/appServer/interface/user/login?	{"mobile":"18259001552", "password":"e10adc3949ba59abbe56e057f20f883e", "model":"小米2S", "SN":"041552E97A96", "osInfo":"android4.0"}	test_login_normal	测试登陆，正向	Pass	
2	POST	modifyPayPwd_normal	/appServer/interface/user/modifyPayPwd	{'newPayPwd': 'e10adc3949ba59abbe56e057f20f883e', 'userId': 2910057590, 'confNewPayPwd': 'e10adc3949ba59abbe56e057f20f883e', 'payPassword': 'e10adc3949ba59abbe56e057f20f883e'}	test_chpasswd_normal	测试更改密码，正向	Pass	

5、 文件与配置

1） http 配置文件

用途：配置接口服务器 IP，端口

http_config.ini

```
[DEFAULT]
[HTTP]
host = 192.168.1.174
port = 9101
```

2） 用例配置文件

run_case_config.ini

```
[RUNCASECONFIG]
runmode = 1
case_id = [1,2]
```

说明：runmode：1--运行全部用例 0--运行指定用例，case_id：list，存放 runmode=0 时需要运行用例的用例 ID，ID 之间采用英文逗号分割

3） 数据库配置文件

db_config.ini

```
[DATABASE1]
host = 192.168.30.80
port = 3306
user = testacc
passwd = test1234
db = testdb
charset = utf8

[DATABASE2]
host = 192.168.1.161
port = 3306
user = yinheonline
passwd = 123456
db = yh_yinheonline
charset = utf8
```

说明：DATABASE1 测试数据库 testdb 的配置，DATABASE2 存放企业应用数据库服务器配置

6、 测试接口样例

■ 1.登陆接口

用于用户登陆。

■ 接口方向

客户端 -> 服务端

■ 接口协议

接口地址：$1dcp_Home/interface/user/login

接口协议：JSON

HTTP 请求方式：GET

■ 消息请求

字段名	数据类型	默认值	必填项	备注
mobile	string		是	手机号
password	string		是	用户密码，采用 MD5加密
model	string		条件	手机型号，比如：IPHONE 5 32GB BLACK
osInfo	string		是	操作系统信息，比如 iOS_6.1.4

SN	string		条件	设备的序列号或唯一标识设备的编码。

说明：必填项为条件表示根据系统的安全级别做限制和约束。客户端和服务端做同步约定。

消息请求样例：

```
?mobile=13812345678&password=xxsdfjddafd&model=iphone5&osInfo=iOS_6.1.4&SN=041552E97A96
```

■ 消息响应

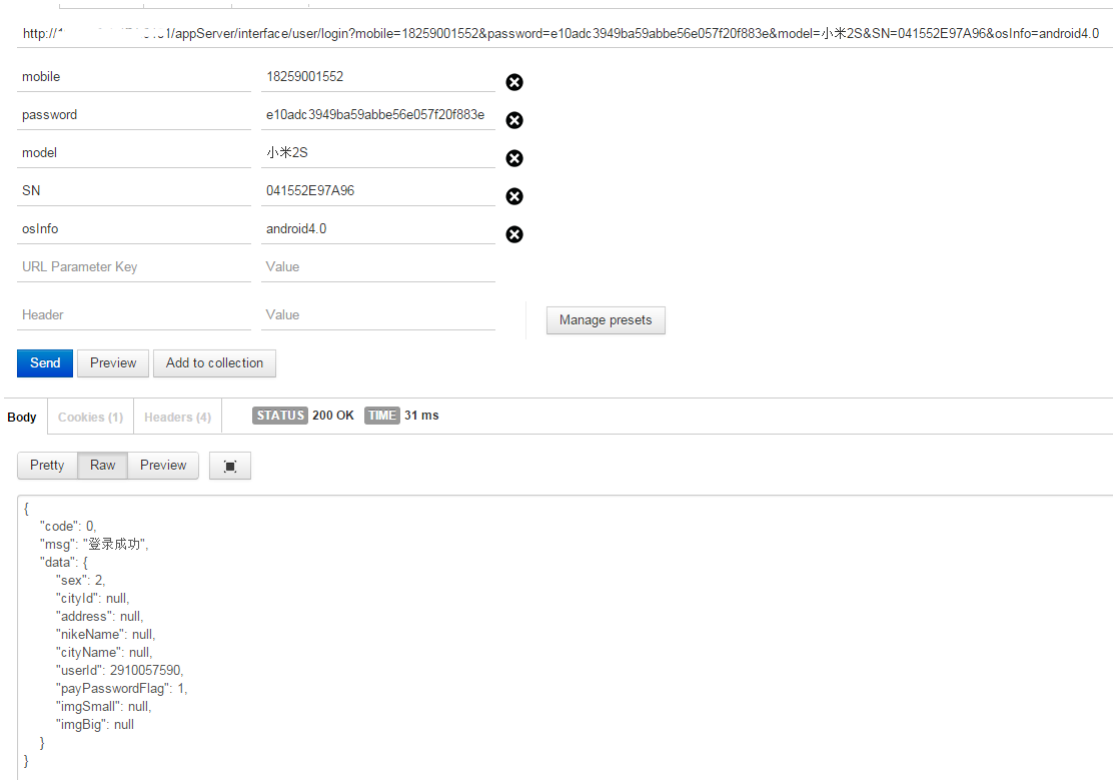
服务器验证成功后，通过在 HTTP response 的 header 中设置 cookie 实现 session 机制。客户端收到 cookie 后，需要在后续的 HTTP request 请求中携带 cookie 信息，否则服务端将鉴权不通过，返回错误码：30003（无效的 cookie 信息）。

字段元素定义如下：

字段名	数据类型	默认值	必填项	备注
userId	int		是	用户 ID
imgBig	String		否	头像（大）
imgSmall	String		否	头像（小）
nikeName	String		是	昵称
sex	int		是	性别：0-男，1-女
address	String		否	长居地
cityId	int		否	市 ID
cityName	String		否	市名
payPasswordFlag	int		是	支付密码标识：0-未设定　1-已设定

成功时，返回 JSON 数据包：

```
{
    "code":0,
    "msg":"登录成功！",
    "data":{
              "userId":"1223434",
              "imgBig":"/user/img/001.png",
              "imgSmall":"/user/img/001_small.png",
              "nikeName":"贪吃羊",
              "sex":0,
              "address":"深圳"
    }
```



■ 2. 支付密码更改接口

会员修改基本资料时，可以对支付密码进行单独修改。

■ 接口方向

客户端 -> 服务端

■ 接口协议

接口地址：$1dcp_Home/interface/user/modifyPayPwd

接口协议：JSON

HTTP 请求方式：POST

■ 消息请求

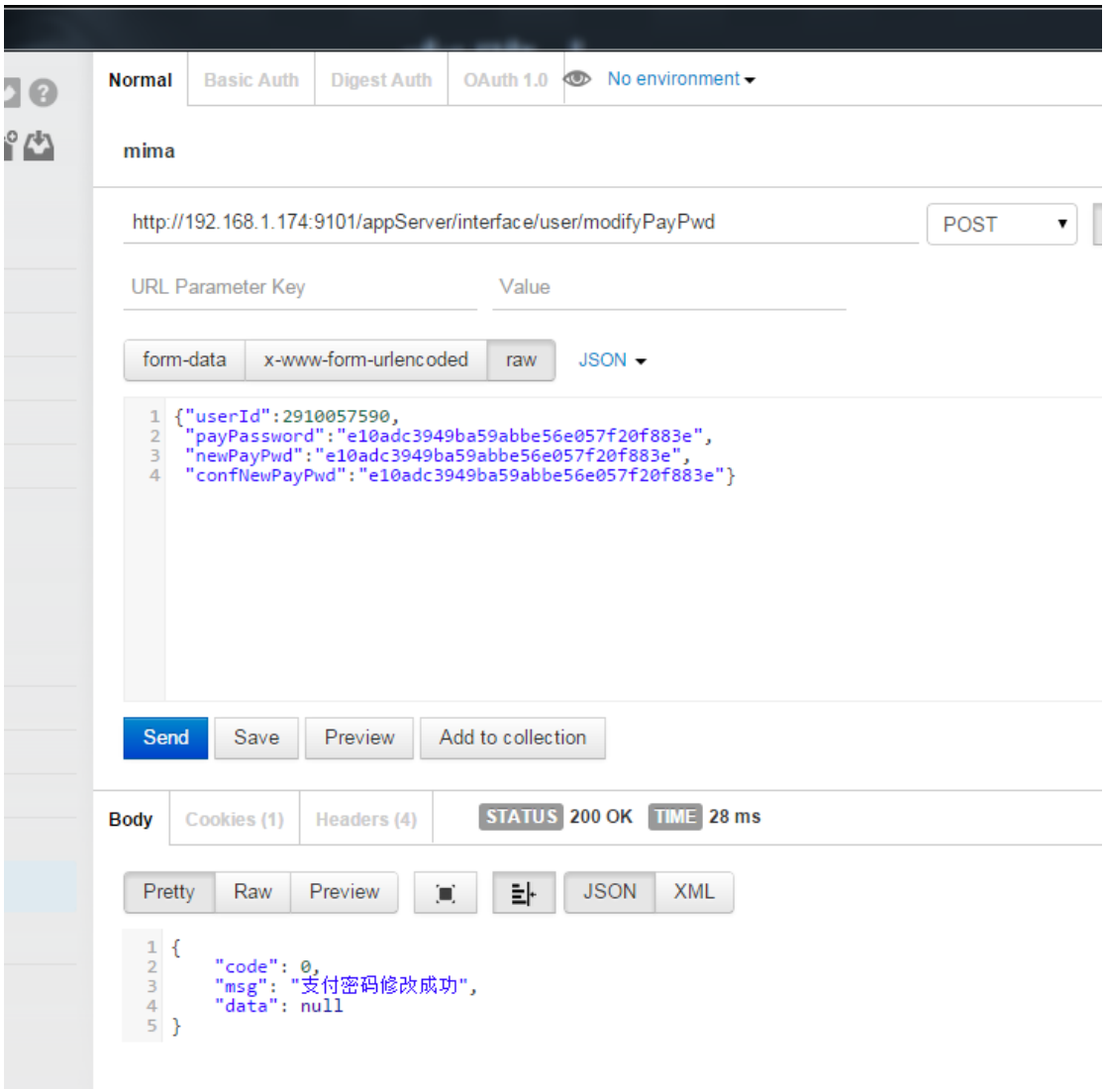
字段列表如下：

字段名	数据类型	默认值	必填项	备注
userId	int		是	会员 ID
payPassword	String		是	原支付密码
newPayPwd	String		是	新支付密码
confNewPayPwd	String		是	确认新支付密码

消息请求样例：

```
{
    "userId":2345,
    "payPassword":"3iu4oi5u3o5o3iu5o3453o3o3",
    "newPayPwd":"40986546i45io6j45o6j3o45",
    "confNewPayPwd":"40986546i45io6j45o6j3o45"
```

```
}
```



7、 数据库设计

1）创建数据库

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

2）testdb 数据库中建立数据表

test_data 存放测试用例(接口)相关数据

```
CREATE TABLE test_data
(
case_id INT NOT NULL UNIQUE,              # 用例 ID,唯一
http_method VARCHAR(5) NOT NULL,          # http 方法（POST、GET
request_name VARCHAR(30),                 # 自定义接口名称 建议格式：接口名-测试简单说明
request_url VARCHAR(200) NOT NULL,        # 接口 URL
request_param VARCHAR(1000) NOT NULL,     # 接口所需的全部或部分参数--python 字典形式的字符串
test_method VARCHAR(50) NOT NULL,         # 测试方法，一个测试用例对应一个方法
test_desc VARCHAR(2000) NOT NULL          # 测试描述--主要描述这个用例的测试点、测试目的
```

```
);

pre_condition_data 存放完成接口运行前置条件所需的数据
CREATE TABLE pre_condition_data
(
case_id INT NOT NULL,                    # 用例 ID
step INT NOT NULL,                       # 执行该用例 ID 需要的第一步、第一个前提条件的 step ID
request_url VARCHAR(200) NOT NULL,       # 接口 URL
request_param VARCHAR(1000) NOT NULL,    # 接口参数--python 字典形式的字符串
other VARCHAR(1000),                     # 保留字段，可能是执行用例需要预先执行的 sql 语句等
test_desc VARCHAR(2000) NOT NULL,        # 数据描述--描述这条数据用途
PRIMARY KEY(case_id, step)
);
```

说明：接口的前提条件往往是另一个接口的预先执行、或预先执行后生成的数据，也就说前一个接口的输出是后一个接口的输入，所以这里主要设计为存储接口 url 和接口参数，供预先执行前一个接口时使用

```
test_result 存放测试结果
CREATE TABLE test_result
(
case_id INT NOT NULL UNIQUE,             # 用例 ID
http_method VARCHAR(5) NOT NULL,         # http 方法（POST、GET
request_name VARCHAR(30),                # 自定义接口名称
request_url VARCHAR(200) NOT NULL,       # 接口 URL
request_param VARCHAR(1000) NOT NULL,    # 接口所需的全部参数--python 字典形式的字符串
test_method VARCHAR(50) NOT NULL,        # 接口测试方法
test_desc VARCHAR(2000) NOT NULL,        # 数据描述--描述测试目的
result VARCHAR(20) NOT NULL,             # 测试结果
reason VARCHAR(20)                       # 测试失败原因
);
```

8、 测试用例、测试数据准备

```
# 测试登录用例数据
INSERT INTO test_data(
case_id,
http_method,
request_name,
request_url,
request_param,
test_method,
test_desc)
VALUES(1,
'GET',
```

```
'login-normal',
'/appServer/interface/user/login?',
'{"mobile":"18259001552",
"password":"e10adc3949ba59abbe56e057f20f883e",
"model":"小米 2S",
"SN":"041552E97A96",
"osInfo":"android4.0"}',
'test_login_normal',
'测试登录，正向'
);

#测试修改支付密码的前置条件数据准备
INSERT INTO pre_condition_data(
case_id,
step,
request_url,
request_param,
other,
test_desc
)
VALUES(2,
1,
'/appServer/interface/user/login?',
'{"mobile":"18259001552",
"password":"e10adc3949ba59abbe56e057f20f883e",
"osInfo":"android4.0"}',
'',
'测试修改密码步骤 1：登录'
);

# 测试修改支付密码数据准备
INSERT INTO test_data(
case_id,
http_method,
request_name,
request_url,
request_param,
test_method,
test_desc)
VALUES(2,
'POST',
'modifyPayPwd_normal',
'/appServer/interface/user/modifyPayPwd',
'{"userId":2910057590,
```

```
 "payPassword":"e10adc3949ba59abbe56e057f20f883e",
 "newPayPwd":"e10adc3949ba59abbe56e057f20f883e",
 "confNewPayPwd":"e10adc3949ba59abbe56e057f20f883e"}',
'test_chpasswd_normal',
'测试更改密码，正向'
);
```

9、 模块与类、函数设计

a) `class confighttp.ConfigHttp`

配置要测试接口服务器的 ip、端口、域名等信息，封装 http 请求方法，http 头设置等

b) `class getdb.GetDB`

负责配置测试数据库，应用数据库服务器的 ip、端口，用户名，密码等信息，返回数据库连接

c) `class configrunmode.ConfigRunMode`

负责配置并获取运行模式，运行用例 ID

d) `class globalconfig.Global`

负责全局初始化配置

e) `class runner.DataStruct`

定义结构体，于接收从测试数据库 testdb 中 test_data 表读取的测试数据,记录要写入测试报告的数据

f) `class test_interface_case.TestInterfaceCase, test_interface_case.ParametrizedTestCase`

负责管理测试用例对应的测试方法,相关的数据处理

g) `runcase.RunCase`

负责运行测试用例及相关的数据处理

h) `htmlreport.HtmlReport`

负责生成测试报告

i) `main`

程序运行入口文件

10、代码实现

a) `class congfighttp.ConfigHttp`

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

__author__ = 'shouke'

import urllib.request
import http.cookiejar
```

```
import urllib.parse
import json
import configparser

# 配置类
class ConfigHttp:
    '''配置要测试接口服务器的 ip、端口、域名等信息，封装 http 请求方法，http 头设置
'''


    def __init__(self, ini_file):
        config = configparser.ConfigParser()

        # 从配置文件中读取接口服务器 IP、域名，端口
        config.read(ini_file)
        self.host = config['HTTP']['host']
        self.port = config['HTTP']['port']
        self.headers = {}  # http 头

        #install cookie
        cj = http.cookiejar.CookieJar()
        opener =
urllib.request.build_opener(urllib.request.HTTPCookieProcessor(cj))
        urllib.request.install_opener(opener)

    def set_host(self, host):
        self.host = host

    def get_host(self):
        return self.host

    def set_port(self, port):
        self.port = port

    def get_port(self):
        return  self.port

    # 设置 http 头
    def set_header(self, headers):
        self.headers = headers

    # 封装 HTTP GET 请求方法
    def get(self, url, params):
        params = urllib.parse.urlencode(eval(params))  # 将参数转为 url 编码字符串
        url = 'http://' + self.host + ':' + str(self.port)  + url + params
```

```
        request = urllib.request.Request(url, headers=self.headers)

        try:
            response = urllib.request.urlopen(request)
            response = response.read().decode('utf-8')  ## decode 函数对获取的字节数据进行解码
            json_response = json.loads(response)  # 将返回数据转为 json 格式的数据
            return json_response
        except Exception as e:
            print('%s' % e)
            return {}

    # 封装 HTTP POST 请求方法
    def post(self, url, data):
        data = json.dumps(eval(data))
        data = data.encode('utf-8')
        url = 'http://' + self.host + ':' + str(self.port)  + url
        try:
            request = urllib.request.Request(url, headers=self.headers)
            response = urllib.request.urlopen(request, data)
            response = response.read().decode('utf-8')
            json_response = json.loads(response)
            return json_response
        except Exception as e:
            print('%s' % e)
            return {}

    # 封装 HTTP xxx 请求方法
    # 自由扩展
```

b) class getdb.GetDB

```
#!/usr/bin/env python

# -*- coding:utf-8 -*-


__author__ = 'shouke'


import configparser

import mysql.connector

import sys
```

```
class GetDB:
    '''配置数据库 IP，端口等信息，获取数据库连接'''
    def __init__(self, ini_file, db):
        config = configparser.ConfigParser()

        # 从配置文件中读取数据库服务器 IP、域名，端口
        config.read(ini_file)
        self.host = config[db]['host']
        self.port = config[db]['port']
        self.user = config[db]['user']
        self.passwd = config[db]['passwd']
        self.db = config[db]['db']
        self.charset = config[db]['charset']

    def get_conn(self):
        try:
            conn = mysql.connector.connect(host=self.host, port=self.port,
user=self.user, password=self.passwd, database=self.db, charset=self.charset)
            return conn
        except Exception as e:
            print('%s', e)
            sys.exit()
```

c) class configrunmode.ConfigRunMode

```
#!/usr/bin/env python
```

```
# -*- coding:utf-8 -*-

__author__ = 'shouke'

import configparser

class ConfigRunMode:
    def __init__(self, run_case_config_file):
        config = configparser.ConfigParser()

        # 从配置文件中读取运行模式
        config.read(run_case_config_file)
        try:
            self.run_mode = config['RUNCASECONFIG']['runmode']
            self.run_mode = int(self.run_mode)

            self.case_list = config['RUNCASECONFIG']['case_id']
            self.case_list = eval(self.case_list)  # 把字符串类型的 list 转换为 list
        except Exception as e:
            print('%s', e)

    def get_run_mode(self):
        return self.run_mode

    def get_case_list(self):
        return  self.case_list
```

d) class globalconfig.Global

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

__author__ = 'shouke'

from getdb import GetDB
from confighttp import ConfigHttp
from configrunmode import ConfigRunMode

class Global:
    def __init__(self):
        # 读取并配置接口服务器 IP，端口等信息
        self.http = ConfigHttp('../http_conf.ini')
```

```
        # 读取并配置数据库服务器 IP，端口等信息
        self.db1 = GetDB('../db_config.ini', 'DATABASE1')
        self.db2 = GetDB('../db_config.ini', 'DATABASE2')

        # 读取运行模式配置
        self.run_mode_config = ConfigRunMode('../run_case_config.ini')

    def get_http(self):
        return self.http

    # 返回测试数据库连接
    def get_db1_conn(self):
        return self.db1.get_conn()

    # 返回应用数据库连接
    def get_db2_conn(self):
        return self.db2.get_conn()

    # 获取运行模式配置
    def get_run_mode(self):
        return self.run_mode_config.get_run_mode()

    # 获取需要单独运行的用例列表
    def get_run_case_list(self):
        return self.run_mode_config.get_case_list()

    # 释放资源
    def clear(self):
        # 关闭数据库连接
        self.db1.get_conn().close()
        self.db2.get_conn().close()
```

e) class datastruct.DataStruct

```
#!/usr/bin/env python

# -*- coding:utf-8 -*-


__author__ = 'shouke'


# 定义结构体
```

```
class DataStruct:
    '''于接收读取的测试数据,记录要写入测试报告的数据'''
    def __init__(self):
        self.case_id = 0        #用例 ID
        self.http_method = ''   #接口 http 方法
        self.request_name = '' #接口 ming
        self.request_url = ''   #接口请求 url
        self.request_param = ''#请求参数
        self.test_method = ''   #测试方法
        self.test_desc = ''     #测试(用力)描述
        self.result = ''        #测试结果
        self.reason = ''        #失败原因
```

注意：断言如果抛出了断言异常，接下去的语句也是不会执行的。

f) class test_interface_case.TestInterfaceCase,test_interface_case.ParametrizedTestCase

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

__author__ = 'shouke'

import  unittest
# 测试用例(组)类
class ParametrizedTestCase(unittest.TestCase):
    """ TestCase classes that want to be parametrized should
        inherit from this class.
    """
    def __init__(self, methodName='runTest', test_data=None, http=None,
db1_cursor=None, db2_cursor=None):
        super(ParametrizedTestCase, self).__init__(methodName)
        self.test_data = test_data
        self.http = http
        self.db1_cursor = db1_cursor
        self.db2_cursor = db2_cursor
```

```
class TestInterfaceCase(ParametrizedTestCase):
    def setUp(self):
        pass

    # 测试接口 1
    def test_login_normal(self):
        # 根据被测接口的实际情况，合理的添加 HTTP 头
        # header =
{'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
        #     'User-Agent':'Mozilla/5.0 (Windows NT 6.1; rv:29.0) Gecko/20100101
Firefox/29.0'
        #     }
        # self.http.set_header(header)
        response = self.http.get(self.test_data.request_url,
self.test_data.request_param)
        if {} == response:
            self.test_data.result = 'Error'
            try:
                # 更新结果表中的用例运行结果
                self.cursor.execute('UPDATE test_result SET result = %s WHERE
case_id = %s', (self.test_data.result, self.test_data.case_id))
                self.cursor.execute('commit')
            except Exception as e:
                print('%s' % e)
                self.cursor.execute('rollback')
            return

        try:
            # 如果有需要，连接数据库，读取数据库相关值，用于和接口请求返回结果做比
较
            self.db2_cursor.execute('SELECT user_id FROM 1dcq_user WHERE mobile
= %s',(eval(self.test_data.request_param)['mobile'],))
            user_id = self.db2_cursor.fetchone()[0]
            self.db2_cursor.close()
            # 断言
            self.assertEqual(response['code'], 0, msg='返回 code 不等于 0')
            self.assertEqual(response['msg'], '登录成功', msg='登录失败')
            self.assertEqual(response['data']['sex'], 2, msg='sex 错误')
            self.assertEqual(response['data']['cityId'], None, msg='cityId 错误')
            self.assertEqual(response['data']['nikeName'], None, msg='nikeName 错误
')
```

```
            self.assertEqual(response['data']['cityName'], None, msg='cityName 错误
')
            self.assertEqual(response['data']['userId'], user_id, msg='userId 错误
')  #2910057590
            self.assertEqual(response['data']['cityName'], None, msg='cityName 错误
')
            self.assertEqual(response['data']['payPasswordFlag'], 1,
msg='payPasswordFlag 错误')
            self.assertEqual(response['data']['imgSmall'], None, msg='imgSmall 错误
')
            self.assertEqual(response['data']['imgBig'], None, msg='imgBig 错误')
            self.test_data.result = 'Pass'
        except AssertionError as e:
            print('%s' % e)
            self.test_data.result = 'Fail'
            self.test_data.reason = '%s' % e # 记录失败原因

        # 更新结果表中的用例运行结果
        try:
            self.db1_cursor.execute('UPDATE test_result SET result = %s WHERE case_id
= %s', (self.test_data.result, self.test_data.case_id))
            self.db1_cursor.execute('UPDATE test_result SET reason = %s WHERE case_id
= %s', (self.test_data.reason, self.test_data.case_id))
            self.db1_cursor.execute('commit')
        except Exception as e:
            print('%s' % e)
            self.db1_cursor.execute('rollback')

    # 测试接口 2
    def test_chpasswd_normal(self):
        header = {'Content-Type':'application/json','charset':'utf-8'}
        self.http.set_header(header)
        # 步骤 1-登录
        self.db1_cursor.execute('SELECT request_url, request_param FROM
pre_condition_data WHERE case_id = %s and step=1', (self.test_data.case_id,))
        temp_result = self.db1_cursor.fetchone()
        request_url = temp_result[0]
        request_param = temp_result[1]
        lgin_response = self.http.get(request_url, request_param)

        # 修改密码
        user_id = lgin_response['data']['userId']    # 获取登录接口返回的 user_id
        payPassword = eval(request_param)['password']    # 获取原密码即登录密码
```

```
        # 拼接参数，作为修改支付密码接口的传入参数
        tmp_dic = {"userId":user_id, "payPassword":payPassword}
        self.test_data.request_param = eval(self.test_data.request_param)
        self.test_data.request_param.update(tmp_dic)

        # 修改密码
        response = self.http.post(self.test_data.request_url,
str(self.test_data.request_param))

        if {} == response:
            self.test_data.result = 'Error'
            try:
                # 更新结果表中的用例运行结果
                self.db1_cursor.execute('UPDATE test_result SET result = %s WHERE
case_id = %s', (self.test_data.result, self.test_data.case_id))
                self.db1_cursor.execute('commit')
            except Exception as e:
                print('%s' % e)
                self.db1_cursor.execute('rollback')
            return
        try:
            self.assertEqual(response['code'], 0, msg='返回 code 不等于 0')
            self.assertEqual(response['msg'],'支付密码修改成功', msg='修改支付密码
失败')
            self.assertEqual(response['data'],None, msg='data 不为 N')
            self.test_data.result = 'Pass'
        except AssertionError as e:
            print('%s' % e)
            self.test_data.result = 'Fail'
            self.test_data.reason = '%s' % e  # 记录失败原因

        # 更新结果表中的用例运行结果
        try:
            self.db1_cursor.execute('UPDATE test_result SET request_param = %s WHERE
case_id = %s', (str(self.test_data.request_param), self.test_data.case_id))
            self.db1_cursor.execute('UPDATE test_result SET result = %s WHERE case_id
= %s', (self.test_data.result, self.test_data.case_id))
            self.db1_cursor.execute('UPDATE test_result SET reason = %s WHERE case_id
= %s', (self.test_data.reason, self.test_data.case_id))
            self.db1_cursor.execute('commit')
        except Exception as e:
            print('%s' % e)
            self.db1_cursor.execute('rollback')
```

```
    def tearDown(self):
        pass
```

g) class runcase.RunCase

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

__author__ = 'shouke'

import unittest
from test_interface_case import TestInterfaceCase
from datastruct import DataStruct

global test_data
test_data = DataStruct()

class  RunCase:
    '''运行测试用例'''

    def __init__(self):
        pass

    # 运行测试用例函数
    def run_case(self, runner, run_mode, run_case_list, db1_conn, db2_conn, http):
        global test_data
        if 1 == run_mode:  # 运行全部用例
            db1_cursor = db1_conn.cursor()
            # 获取用例个数
            db1_cursor.execute('SELECT count(case_id)  FROM test_data')
            test_case_num = db1_cursor.fetchone()[0]
            db1_cursor.close()

            # 循环执行测试用例
            for case_id in range(1, test_case_num+1):
                db1_cursor = db1_conn.cursor()
                db2_cursor = db2_conn.cursor()
                db1_cursor.execute('SELECT http_method, request_name, request_url,
request_param, test_method, test_desc '
                                   'FROM test_data WHERE case_id
= %s',(case_id,))
                # 记录数据
                tmp_result = db1_cursor.fetchone()
                test_data.case_id = case_id
                test_data.http_method = tmp_result[0]
```

```
                    test_data.request_name = tmp_result[1]
                    test_data.request_url = tmp_result[2]
                    test_data.request_param = tmp_result[3]
                    test_data.test_method = tmp_result[4]
                    test_data.test_desc = tmp_result[5]
                    test_data.result = ''
                    test_data.reason = ''
                    try:
                        query = ('INSERT INTO test_result(case_id, http_method,
request_name, request_url,'
                                 'request_param, test_method, test_desc, result,
reason) VALUES(%s, %s, %s, %s, %s, %s, %s, %s, %s)')

                        data =
(test_data.case_id,test_data.http_method,test_data.request_name,
test_data.request_url,
                                test_data.request_param, test_data.test_method,
test_data.test_desc,
                                test_data.result, test_data.reason)
                        db1_cursor.execute(query, data)
                        db1_cursor.execute('commit')
                    except Exception as e:
                        # 回滚
                        print('%s' % e)
                        db1_cursor.execute('rollback')

                    test_suite = unittest.TestSuite()
                    test_suite.addTest(TestInterfaceCase(test_data.test_method,
test_data, http, db1_cursor, db2_cursor))
                    runner.run(test_suite)
                    db1_cursor.close()
                    db2_cursor.close()
            else:   # 运行部分用例
                # 循环执行测试用例
                for case_id in run_case_list:
                    db1_cursor = db1_conn.cursor()
                    db2_cursor = db2_conn.cursor()
                    db1_cursor.execute('SELECT http_method, request_name, request_url,
request_param, test_method, test_desc '
                                       'FROM test_data WHERE case_id
= %s',(case_id,))
                    # 记录数据
                    tmp_result = db1_cursor.fetchone()
                    test_data.case_id = case_id
```

```
                test_data.http_method = tmp_result[0]
                test_data.request_name = tmp_result[1]
                test_data.request_url = tmp_result[2]
                test_data.request_param = tmp_result[3]
                test_data.test_method = tmp_result[4]
                test_data.test_desc = tmp_result[5]
                test_data.result = ''
                test_data.reason = ''

                try:
                    query = ('INSERT INTO test_result(case_id, http_method,
request_name, request_url,'
                             'request_param, test_method, test_desc, result,
reason) VALUES(%s, %s, %s, %s, %s, %s, %s, %s, %s)')

                    data =
(test_data.case_id,test_data.http_method,test_data.request_name,
test_data.request_url,
                            test_data.request_param, test_data.test_method,
test_data.test_desc,
                            test_data.result, test_data.reason)
                    db1_cursor.execute(query, data)
                    db1_cursor.execute('commit')
                except Exception as e:
                    # 回滚
                    print('%s' % e)
                    db1_cursor.execute('rollback')
                test_suite = unittest.TestSuite()
                test_suite.addTest(TestInterfaceCase(test_data.test_method,
test_data, http, db1_cursor, db2_cursor))
                runner.run(test_suite)
                db1_cursor.close()
```

h) htmlreport.HtmlReport

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

__author__ = 'shouke'

from pyh import *
import time
import os

class HtmlReport:
```

```
    def __init__(self, cursor):
        self.title = 'test_report_page'    # 网页标签名称
        self.filename = ''                 # 结果文件名
        self.time_took = '00:00:00'        # 测试耗时
        self.success_num = 0               # 测试通过的用例数
        self.fail_num = 0                  # 测试失败的用例数
        self.error_num = 0                 # 运行出错的用例数
        self.case_total = 0                # 运行测试用例总数
        self.cursor = cursor

    # 生成 HTML 报告
    def generate_html(self,head, file):
            page = PyH(self.title)
            page << h1(head, align='center') # 标题居中

            page << p('测试总耗时：' + self.time_took)

            # 查询测试用例总数
            query = ('SELECT count(case_id) FROM test_result')
            self.cursor.execute(query)
            self.case_total = self.cursor.fetchone()[0]

            # 查询测试失败的用例数
            self.cursor.execute('SELECT count(case_id) FROM test_result WHERE
result = %s',('Fail',))
            self.fail_num = self.cursor.fetchone()[0]

            # 查询测试通过的用例数
            self.cursor.execute('SELECT count(case_id) FROM test_result WHERE
result = %s',('Pass',))
            self.success_num = self.cursor.fetchone()[0]

            # 查询测试出错的用例数
            self.cursor.execute('SELECT count(case_id) FROM test_result WHERE
result = %s',('Error',))
            self.error_num = self.cursor.fetchone()[0]

            page << p('测试用例数：' + str(self.case_total) + ' '*10 + '成功用
例数：' + str(self.success_num) +
                      ' '*10 + '失败用例数：' + str(self.fail_num) + ' '*10
+ '出错用例数：' + str(self.error_num))
            #  表格标题 caption 表格边框 border 单元边沿与其内容之间的空白
cellpadding 单元格之间间隔为 cellspacing
```

```
            tab = table( border='1', cellpadding='1', cellspacing='0', cl='table')
            tab1 = page << tab
            tab1 << tr(td('用例 ID', bgcolor='#ABABAB', align='center')
                       + td('HTTP 方法', bgcolor='#ABABAB', align='center')
                       + td('接口名称', bgcolor='#ABABAB', align='center')
                       + td('请求 URL', bgcolor='#ABABAB', align='center')
                       + td('请求参数/数据', bgcolor='#ABABAB', align='center')
                       + td('测试方法', bgcolor='#ABABAB', align='center')
                       + td('测试描述', bgcolor='#ABABAB', align='center')
                       + td('测试结果', bgcolor='#ABABAB', align='center')
                       + td('失败原因', bgcolor='#ABABAB', align='center'))

            # 查询所有测试结果并记录到 html 文档
            query = ('SELECT case_id, http_method, request_name, request_url,'
                     'request_param, test_method, test_desc, result, reason
FROM test_result')
            self.cursor.execute(query)
            query_result = self.cursor.fetchall()

            for row in query_result:
                tab1<< tr(td(int(row[0]), align='center') + td(row[1]) +
                          td(row[2]) + td(row[3], align='center') +
                          td(row[4]) + td(row[5]) + td(row[6]) +
                          td(row[7], align='center') + td(row[8]))

            self._set_result_filename(file)
            page.printOut(self.filename)

            try:
                query = ('DELETE FROM test_result')
                self.cursor.execute(query)
                self.cursor.execute('commit')
            except Exception as e:
                # 回滚
                print('%s' % e)
                self.cursor.execute('rollback')
            self.cursor.close()

    # 设置结果文件名
    def _set_result_filename(self, filename):
        self.filename = filename
        #判断是否为目录
        if os.path.isdir(self.filename):
            raise IOError("%s must point to a file" % path)
```

```
        elif '' == self.filename:
            raise IOError('filename can not be empty')
        else:
            parent_path, ext = os.path.splitext(filename)
            tm = time.strftime('%Y%m%d%H%M%S', time.localtime())
            self.filename = parent_path + tm + ext

    # 统计运行耗时
    def set_time_took(self, time):
        self.time_took = time
        return self.time_took
```

i) main

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

__author__ = 'shouke'

import datetime
import unittest

from runcase import RunCase
from globalconfig import Global
from htmlreport import HtmlReport

if __name__ == '__main__':
    # 记录测试开始时间
    start_time = datetime.datetime.now()

    # 全局配置
    global_config = Global()
    run_mode = global_config.get_run_mode() # 运行模式
    run_case_list = global_config.get_run_case_list()  # 需要运行的用例列表
    db1_conn = global_config.get_db1_conn()    # 数据库连接
    db2_conn = global_config.get_db2_conn()    # 数据库连接
    http = global_config.get_http()            # http

    # 运行测试用例
    runner = unittest.TextTestRunner()
    case_runner = RunCase()
    case_runner.run_case(runner, run_mode, run_case_list, db1_conn, db2_conn, http)

    # 记录测试结束时间
```

```
end_time = datetime.datetime.now()

# 构造测试报告
html_report = HtmlReport(db1_conn.cursor())
html_report.set_time_took(str(end_time - start_time))  # 计算测试消耗时间

# 生成测试报告
html_report.generate_html('test report', '../report.html')

# # 释放数据库连接资源
global_config.clear()
```

11、源码下载

http://pan.baidu.com/s/1dDoHsOx