# JoyGamesQAHTTP 文档

| 日期 | 更新说明 | 作者 |
| --- | --- | --- |
| 2017-04-12 | 初始化 JoyGamesQAHTTP 文档 | 吴炜 |

## 一. 简介

本工具（框架），旨在将接口测试过程中编写代码的工作量减少，使 QA 只需要专心于设计测试用例，从而提高工作效率.

一般的接口测试流程为：准备测试用例套件->描述测试场景函数->发起客户端请求->收集服务器响应->断言->出测试报告.

使用本框架后，QA 只需要准备测试用例套件及进行少量的描述测试场景函数（大多数情况下只有参数表值调整的工作，已提供 update_request_param 函数，可以很便捷地完成操作）. 其余调用测试用例、发起请求、收集响应数据、断言、出测试报告的工作都不需要编写代码. 大多数情况下，Client 进行简单配置文件修改，Server 端通过 MySQL 提交测试用例，即可进行接口测试，并生成 HTML 报告.

为什么不把 signature 加密的额外工作也处理了？

这和本工具的定位有关. 本工具是为了将一般的接口测试流程框架化，同时提供开源的描述测试场景函数入口，方便 QA 调整测试策略，而不是再做一个 POSTMAN.

## 二. 开发环境

Debian 3.2.81-1 x86_64

Python 3.5.0

MySQL Ver 14.14 Distrib 5.5.49

## 三. 使用环境

已验证能在以下 Client 端环境正常使用：

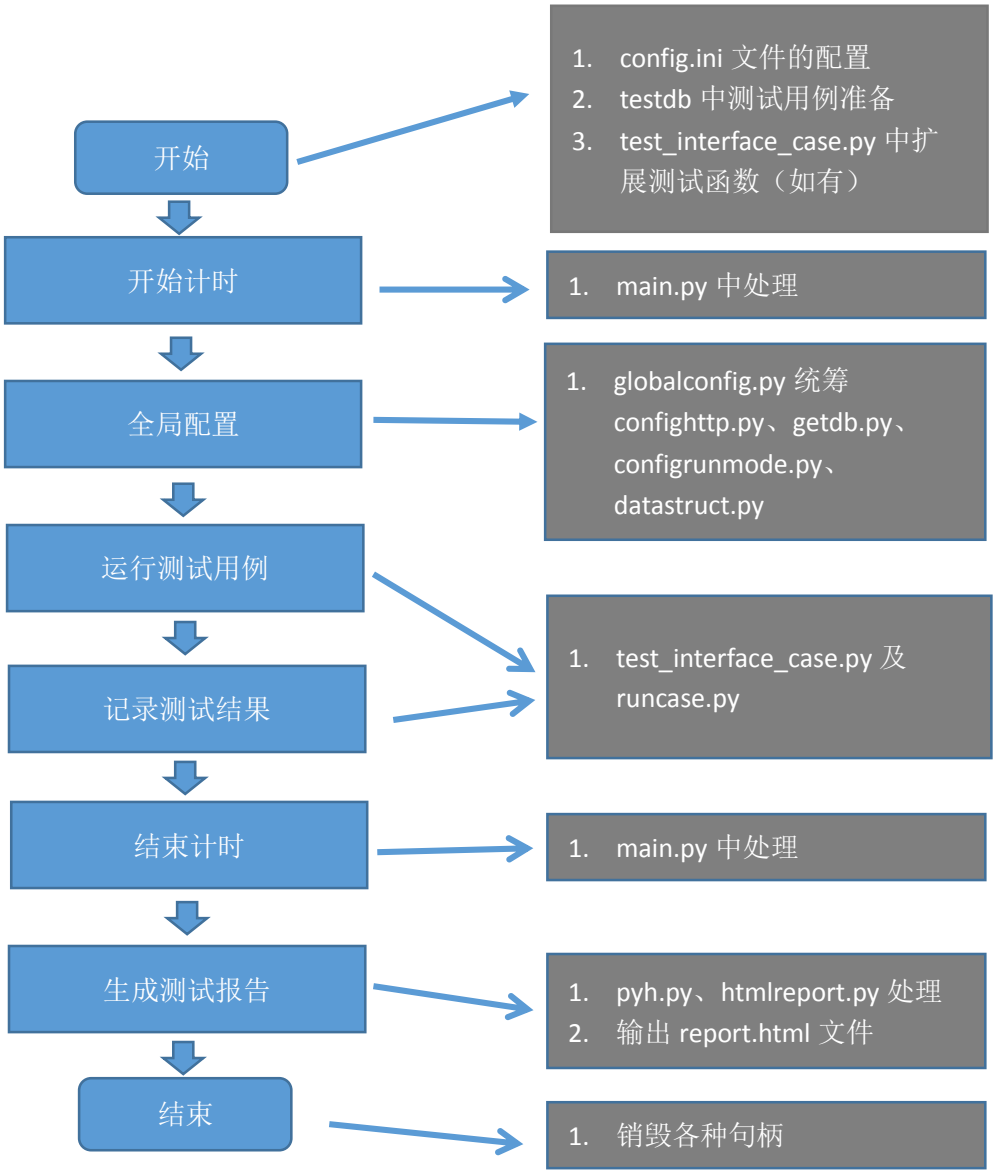
1. 环境 1：Debian 3.2.81-1 x86_64 + Python 3.5.0
2. 环境 2：Window 7 + Python 3.4.3

注意：为提供更自由的测试场景描述，本工具开源，且不以打包可执行文件方式进行分发.

## 四. 框架说明

1. 支持 Request Method 扩展，Get/Post/Put/Delete 等，甚至是 socket.
2. 支持自定义测试接口场景的描述扩展，目前统一用 test_interface_case()通用接口测试函数，对于比较复杂的测试流程，可以额外定义函数，支持拓展.
3. 支持 http/https 双协议头选择，支持自定义 headers 描述，支持 cookie.
4. 支持通过在 HTTP response 的 header 中设置 cookie 实现 session 机制。
5. 支持测试结果 HTML 报告输出，报告支持 Windows/Linux 中文编码.
6. 支持通过 config.ini[[DATABASE]] [db、user、passwd]的配置，实现请求不同的数据库，从而达到测试用例分离，多人同时使用的效果.

## 五. 流程说明




## 六. 配置文件说明

配置文件：config.ini

[HTTP] 配置 HTTP 相关信息
protocol = http/https
host = URI 里的主机地址，可以为 ip 或域名
port = 服务监听端口，http 默认 80，https 默认 443
headers = 自定义 http 头

[DATABASE]配置数据库信息，对普通使用者来说，就是存放测试用例的地方
host = 139.196.230.110 （MySQL 主机地址）
port = 3306 （MySQL 默认监听端口）

user = JoyGamesQA （MySQL 帐号）
passwd = Aofei123 （MySQL 密码）
db = testdb （选择的数据库）
charset = utf8 （编码）

[RUNCASECONFIG]配置测试用例的运行模式
runmode = 1（1 为运行所有用例，0 为运行指定范围用例）
case_id = [3,4]（运行指定范围用例，case_id from 3 to 4）

## 七. 数据库说明

举个例子：
数据库：testdb
表：1. test_data 用来存放测试用例
2. test_result 程序运行过程中使用，不介绍

| test_data 表字段说明 | |
|---|---|
| **case_id** | 测试用例编号，从 1 开始自增 |
| **http_method** | POST/GET |
| **request_name** | 自定义的描述，比如通用注册接口 |
| **request_url** | 请求接口的 URL |
| **request_param** | 需要传给接口的参数 |
| **test_method** | 指定 test_interface_case.py 中的测试函数，默认 test_interface_case |
| **response_expectation** | 期望返回值 |
| **test_desc** | 备注，可以自定义描述 |

## 八. 脚本文件说明

说明：所有源码里都写了详细的注释，及遇到的坑的处理，这里只做简要说明

1. confighttp.py，配置要测试接口服务器的协议,ip、端口、域名等信息，封装 http 请求方法，http 头设置等.
2. getdb.py，配置数据库 IP，端口等信息，获取数据库连接.
3. configrunmode.py，从配置文件中读取运行模式.
4. globalconfig.py，负责配置的全局初始化，包括 ConfigHttp、GetDB、ConfigRunMode.
5. datastruct.py，定义结构体，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.
6. test_interface_case.py，负责管理测试用例对应的测试方法,相关的数据处理.
7. runcase.py，运行测试用例.
8. htmlreport.py，生成测试报告.
9. pyh.py，开源 python-html 库，修改了 pyh.py 源码的 printOut()函数，支持跨平台中文编码. windows/linux.
10. main.py，统筹总体流程.

## 九. 一般使用流程

1. 编写 config.ini 进行配置
2. testdb 中测试用例准备
3. test_interface_case.py 中扩展测试函数（如有）
4. python 运行 main.py 脚本
5. 打开 report.html 文件查看测试报告

## 十. 测试报告展示

Test Report

测试总耗时：0:00:02.830004

测试用例数：4　　成功用例数：1　　失败用例数：3　　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 | POST | Android init接口 | /init/android/sdk/ | {"appID":"1106230001", "channelMasterID":"9998","patchVersion":"0","signature":"59b568f43e31a63a28f32501647730e5"} | test_interface_case | 成功 | Pass | |
| 2 | POST | Android init接口 | /init/android/sdk/ | {"appID":"110001", "channelMasterID":"9998","":"0","signature":"59b568f43e31a63a28f32501647730e5"} | test_interface_case | appID不存在 | Fail | Raise Error:10002 != 999 , Response:{'result': 0, 'msg': 'signature校验失败', 'data': None, 'code': 10002} |
| 3 | POST | Android init接口 | /init/android/sdk/ | {"appID":"110001", "channelMasterID":"9998","patchVersion":"0","signature":"59b58f32501647730e5"} | test_interface_case | signature有误 | Fail | Raise Error:10002 != 999 , Response:{'result': 0, 'msg': 'signature校验失败', 'data': None, 'code': 10002} |
| 4 | POST | Android init接口 | /init/android/sdk/ | {"appID":"1106230001", "channelMasterID":"9998","patchVersion":"0","signature":""} | test_interface_case | 懒得写 | Fail | Raise Error:10001 != 999 , Response:{'result': 0, 'msg': '参数缺失', 'data': None, 'code': 10001} |

## 十一. 下一步计划

目前是 C/S 架构的接口测试框架，将来需要改进版本，实现转为基于 B/S 架构的工具，所有操作通过前端 WEB 页面配置与提交，所有结果通过 WEB 页面展现，不再需要客户端脚本分发.