**目录**

[一 项目整体介绍 2](#_Toc512444156)

[1. 项目背景 2](#_Toc512444157)

[2. 项目需求 2](#_Toc512444158)

[3. 项目框架设计 2](#_Toc512444159)

[4. 平台使用流程介绍 3](#_Toc512444160)

[二 项目实施过程 4](#_Toc512444161)

[1. 环境搭建 4](#_Toc512444162)

[2. 流程设计 4](#_Toc512444163)

[3. 项目文件结构组织 5](#_Toc512444164)

[三 项目细节设计说明 6](#_Toc512444165)

[1. 模块介绍 6](#_Toc512444166)

[1) api模块 6](#_Toc512444167)

[2) ExcelUtil模块 6](#_Toc512444168)

[3) Logs模块 6](#_Toc512444169)

[4) sendReport模块 6](#_Toc512444170)

[5) Main模块 6](#_Toc512444171)

[2. 配置文件说明 6](#_Toc512444172)

[3. 用例管理功能 8](#_Toc512444173)

[1）用例分类 8](#_Toc512444174)

[2）用例命名 8](#_Toc512444175)

[3）添加用例执行 8](#_Toc512444176)

[四 平台集成项目 9](#_Toc512444177)

[1. 向平台集成测试项目 9](#_Toc512444178)

[2. 已经集成的测试项目（部分完成） 9](#_Toc512444179)

[3. 计划集成的测试项目 9](#_Toc512444180)

[五 项目展望 9](#_Toc512444181)

**自动化测试平台设计文档**

# 一 项目整体介绍

## 1. 项目背景

随着公公司的发展与壮大，相应的项目也越来越多，QA为保证产品的质量，工作的任务量也在急剧增加。而QA大部分时间都花在了功能测试保障、回归测试和产品发布验证等重复且量大的工作上面；在某个项目上面深入展开测试与探究的时间与精力日趋减少，这必然会导致产品深层次的BUG的发现概率降低。

我们的很多产品都已经趋于成熟，比如本地API，云端API，AC的WEB等，我们测试人员也编写了完整的测试用例。在上述产品迭代时，QA的主要工作就是重复性的执行这些测试用例。因此，我们可以展开上述功能的自动化测试工作。

## 2. 项目需求

目前我们自动化项目整理的需求如下：

同一份代码可以在多个项目执行

各个项目的测试数据相互独立，不会互相影响

各个项目的测试环境相互独立，不会互相影响

当用例执行出错时，有详细的日志帮助定位

自动化生成测试报告并发送邮件

较好的可维护性和集群扩展性。

## 3. 项目框架设计

针对项目的需求，自动化测试平台设计了测试计划（test\_plan）和测试任务(test\_job)两个概念。

测试计划是测试报告的生成依据，测试报告与测试计划一一对应。测试平台可以同时开启多个测试计划，一个测试计划下面至少包含一个测试任务。

测试任务是测试用例的执行者，一个测试任务下面可以有很多测试用例。测试任务可以指定单独的测试环境，同时也可以多个测试任务共享一个测试环境。

下面通过几幅图来体现上面几个部分之间的关系。

**测试平台和测试计划的关系图：**

测试平台

测试计划1 测试计划2 测试计划n

*一个测试平台可以包含多个测试计划，没和测试计划的测试环境相互独立。*

**测试计划和测试任务的关系图：**

测试计划

测试环境1 测试环境2 测试环境n

测试任务1 测试任务2 测试任务3 测试任务n

*一个测试计划可以包含多个测试测试任务，多个测试任务可以共享同一个测试环境。*

**测试任务和测试用例关系图：**

测试任务

Case1 case2 case3 case…

*一个测试任务可以包含多个测试用例*

## 4. 平台使用流程介绍

整个平台的使用流程大概如下：

1. 通过配置文件或者web页面（暂未实现）添加测试计划
2. 在测试计划下面添加测试任务，配置测试任务的详细参数，包括使用环境，测试数据等
3. 如果测试计划下面要进行多个测试环境的测试，那么重复步骤2继续添加测试任务即可
4. 完成当前测试计划的指定，如果还有测试计划，重复1-3步骤
5. 执行所有测试计划，即可完成测试

# 二 项目实施过程

1. 环境搭建

首先是语言的选择，由于我们QA同事对于python语言基础较好，因此我们的项目选择python作为开发语言。

项目框架搭建过程中，主要用到了以下python现有框架和工具：

* + - * svn/git：管理代码工程
      * unittest:作为测试核心框架
      * logging：python的内置模块，用来管理日志
      * ddt：python三方框架，用来做数据驱动
      * requests:python内置模块，用来发送接口请求
      * HTMLTestRunner：python三方模块，继承自unittest用来生成测试报告
      * Smtplib：python内置模块，用来自动发送邮件

1. 流程设计

下面的流程图，体现了工具的主要逻辑流程。

Main

Config

Test plan

Test jobs

Report model

test end

Test start

Run plan

Send report

All jobs

All plan

N

Y

N

Y

流程图说明：

Step1. 测试开始，main函数去config文件中读取测试参数

Step2. main函数根据step1获取的测试参数，初始化测试环境，生成测试计划，

Step3. 向测试计划添加测试任务，一个测试计划可以添加多个测试任务

Step4. 运行测试计划

Step5. Main函数调用report模块生成报告

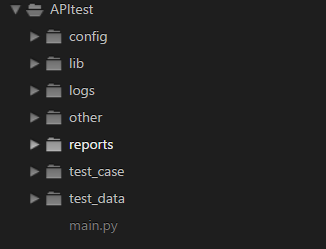
Step6. 判断是否还有测试计划执行，如果有重复之前的步骤运行测试计划

Step7. 执行完所有的测试计划之后，调用邮件发送模块，自动发送测试报告

Step8. 结束测试

1. 项目文件结构组织

上面工具的流程图，在工具的文件目录结构体现如下：



Config目录对应流程图中的config，里面存放测试用的所有配置参数，比如AP的MAC，AC的IP地址。配置文件的格式为Json，用key：value的方式存放配置参数；

Lib目录对应流程图中的libs，里面存放着测试用的封装函数和类，比如API的调用接口类，配置文件的读取工具函数，日志打印类等。

Logs目录存放整个测试过程中的必要信息，有三个文件：

info.txt文件只保存测试用例的标题以及执行结果，方便查看结果；

error.txt文件只存放失败测试用例的失败原因以及失败信息；

debug.txt除了包含上面两个文件的信息之外，还包含测试用例每一步的执行情况。

Reports 目录存放测试执行结束之后生成的报告

Test\_case 目录存放所有额测试用例脚本

Test\_data 目录存放所有的测试用数据，为excel表格形式

# 三 项目细节设计说明

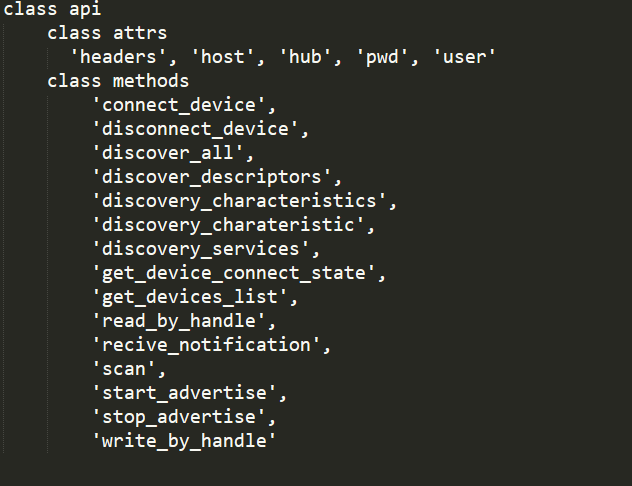
1. 模块介绍

项目的通用模块全部放在libs目录下面，下面分别介绍每个模块的主要功能:

1. api模块

该模块基于request模块，封装了我们所有的本地API和云端API的调用在一个api类中，我们的每一个API接口，都是一个类方法，与我们的API接口名称保持一致。在进行项目的接口功能，系统，稳定性等测试时，直接调用该模块实例化api类，然后调用相应的类方法即可完成我们API的调用。

下面是api类的结构图：



1. ExcelUtil模块

封装了python的excel处理类，该类有两个方法：

get\_single 这个方法返回一个指定名字的sheet数据，返回值类型为list，需要传入参数sheet名称

get\_all 这个方法返回所有sheet的数据，返回值类型为dict，key为sheet名字，value为单个sheet的list格式数据

1. Logs模块

基于python的logging模块，封装了logging的实现，通过该模块可以直接返回log日志的管理对象

1. sendReport模块

该模块功能为发送测试报告，该模块通常由main函数直接调用，用户只需在config配置相应的参数即可

1. Main模块

该模块是程序的驱动模块，程序的启动。运行以及结束全在该模块实现，该模块由config配置文件驱动。

1. 配置文件说明

配置文件结构如下图所示：



Test\_plans

该字段表示一个测试计划的集合，所有的test\_plan都应该在这里面配置。

Test\_plan

该字段表示一个具体的测试计划，之后的大括号里面的所有配置全是该测试计划的具体配置参数；如果需要，一个config文件中，可以同时配置多个test\_plan。

Comment

该字段表示测试计划的解释说明，该字段的内容会直接被测试报告中引用。所以测试的相关说明性内容都可以在这里配置。

Jobs

该字段表示一个测试业务的集合，里面可以配置多个独立的测试业务。

Job

该字段表示一个具体的测试业务，同时也是测试工具最小的执行单元。

Data\_file

该字段指定测试业务需要的配置文件名称，注意：因为程序将数据文件的路径写死了，因此这里只需要指定文件名称即可，同时也意味着数据文件只能放在上面介绍的test\_data目录

Host

该字段指定测试用的AC或者AP。

当测试AC的API时，该字段配置AC的IP地址，例：<http://172.16.30.100/api>

当测试AP的本地API时，该字段配置AP的IP地址，例：<http://172.16.10.10>，注意：没有‘/api’

Hub

该字段指定测试使用的AP mac地址。当测试本地API时，该字段可以不配置。

Model

该字段配置测试用AP的设备类型

User and pwd

这两个字段配置AC的开发者账号和密码，当测试云端API时，该字段必须配置正确，测试本地API时，该字段无需配置

Filter

该字段配置一些扫描过滤相关的参数，该部分配置主要用在scan接口的测试。可以根据自己实际的测试环境，配置需要的过滤参数。

case\_path

该字段指定了case字段配置的case所在项目文件夹，我么test\_case下的每个目录都可以理解成一个测试项目，

如果不配置该字段,系统会默认去test\_case目录下查找所有的项目。

Case

该字段指定了测试业务需要执行的测试用例，最小粒度为一个测试参数，可以同时指定多个测试参数。

1. 用例管理功能

1）用例分类

通过把不同项目的测试用例放在不同的目录，从而区分不同项目的测试用例。

比如，我们把api所有的功能测试用例放在api\_func\_test目录下面；将API所有的系统测试用例全部放在api\_sys\_test目录下面。

这样做的好处就是，当我们想运行API的功能测试时，直接去\*\*/api\_func\_test/目录下面去查找测试用例即可。

2）用例命名

我们每条测试用例的名称，都是严格遵循下面规则的：

Test开头，后面字段为测试模块，后面紧跟该模块的测试点，最后为该功能点下的测试用例编号。

例如：test\_scan\_chip\_600\_1987.py这个测试文件，其中

test开头表示这是一个测试用例

scan表示这个测试用例的测试模块是scan

chip表示这个测试用例测试参数是chip

600\_1987代表这个测试用例的唯一ID，与testlink上面的测试用例ID一一对应

当然，上面说的层级结构不是固定的，当测试的模块比较复杂是，可能会有更多的层级结构，但是规则不会发生变化。

3）添加用例执行

我们上面1 2 两部分的介绍，其实都是为了该部分服务的。通过上面的方式，我们现在想查找一个测试用例就比较容易了。例如，我们想运行scan接口chip参数相关的系统测试用例，我们只需要去api\_sys\_test目录下面查找名字为test\_scan\_chip\_\*的所有测试用例即可。

也就是说，我们通过**文件路径+文件名**的方式定位就可以定位到我们想要运行的测试用例。

因此，我们在conf文件中配置job的case字段时，首先应该在case\_path字段指定要运行的case属于哪个项目，然后在case字段指定我们要测试的模块或者模块功能点。

详细配置举例：

1）如果想运行某个项目下的所有case，只需在case配置”**test\_\*.py**”即可

2）如果想运行某个项目下的某个模块的所有case，只需在case配置”**test\_模块名\*.py**”即可

3）如果想运行某个项目下的某个模块的某个功能点的case，只需在case配置”**test\_模块名\_功能点\*.py**”即可

4）如果想运行某个具体的case，只需要配置该case的全名即可，**test\_模块名\_功能点\_序号.py**。

5）如果想运行多个模块或者功能点的case，只需要按照2），3）中的方式配置多个字段自可，多个字段用英文逗号隔开

# 四 平台集成项目

1. 向平台集成测试项目

向平台集成新的测试项目是很简单的。集成新的项目，就是将测试用例放到平台的case库中，然后在配置文件中制定测试计划和测试任务，然后就可以用平台来自动的执行测试项目用例，生成相应的测试报告。

1. 已经集成的测试项目（部分完成）
2. 全系列产品的本地API功能测试
3. 全系列产品的本地API系统测试
4. 全系列产品的本地API稳定性测试
5. V1版本的云API功能测试
6. V1版本的云API系统测试
7. V1版本的云API稳定性测试
8. 计划集成的测试项目
9. web ui的功能测试
10. web ui的系统测试
11. AC的性能测试
12. AP的CAPWAP等测试
13. 。。。

# 五 项目展望

目前的主要工作是将已经集成的项目的手工测试case尽量转化为可自动化执行且可靠的自动化测试脚本。接下来计划的工作内容如下：

1. 平台的UI界面开发，简化平台的操作步骤，提高易用性。
2. 增加覆盖率统计
3. 基于Jenkins完成持续集成
4. 基于多线程或者多继承运行testplan，提高效率