

# 测试框架使用手册 V1.0.0

---

## 一、介绍

---

luban-common 是一款面向鲁班内部的 HTTP(S) 协议的通用测试框架。他的设计理念是把所有接口看成一个一个的模块，像搭积木一样，把相关接口进行组装，行成场景测试用例；说到场景用例那肯定有单接口测试，单接口测试会直接通过框架来生成，无需人工参与或少量参与即可完成，我们的目标是把可标准化、重复性的工作让机器来完成，让测试人员更多的关注场景和其它异常类测试。

### 1.1 为什么创建这套框架

有如下几方面原因致使我们创建这个框架：

**统一技术栈：**确定框架选型，确定人员技术栈，不用纠结用什么框架，学什么语言

**方便推广：**技术、经验可更好的积累和传播

**方便管理：**有问题和需求可统一实现和修改，主框架调整不影响已有接口和数据，目录结构统一后，其它人员介入也变得简单

**技术可控：**世面上是有很多框架，但要找到适合自己项目的几乎没有，或多或少都要进行部分修改，二次开发的工程量也不小，我们可以基于一个比较成熟的开源框架开发和优化想要的功能，由于功能都是自己开发和修改的，有问题有需求能快速调整。

### 1.2 核心特性

- 基于 `pytest` 扩展，继承 `pytest` 的全部特性，测试前后支持完善的 hook 机制
- 继承 `Requests` 的全部特性，轻松实现 HTTP(S) 的各种测试需求
- 采用 `YAML` 的形式保存环境配置，可动态指定配置文件
- 支持完善的测试用例分层机制，充分实现复用
- 响应结果支持丰富的校验机制
- 通过 `luban new` 创建项目脚手架命令，可快速构建一个完整的项目目录结构
- 通过 `luban swagger` 生成 `swagger` 接口命令，可快速生成接口方法
- 通过 `luban swaggerCase` 生成测试用例命令，可快速生成简单的测试用例

## 二、安装

---

### 2.1 安装方式

PyPI安装(版本稳定后会托管到PyPI上)

```
pip install luban-common
```

本地安装

```
pip install luban_common-0.5.27-py3-none-any.whl
```

从git安装

```
pip install git+https://github.com/mongnet/luban_common@master
```

## 2.2 版本升级

假如你之前已经安装过了 luban-common，现在需要升级到最新版本，那么你可以使用 `-u` 参数。

```
pip install -u luban-common
pip install -u git+https://github.com/mongnet/luban_common@master
或
pip install -u luban_common-0.5.27-py3-none-any.whl
```

## 2.3 安装验证

运行如下命令，若正常显示版本号，则说明 luban-common 安装成功

```
C:\Users\admin>luban -v
Luban version 3.0.0
```

# 三、框架结构

luban-common 框架项目结构如下：

```
├─config
│   ├──config.yaml
│   ├──parameConfig.yaml
│   ├──cases.mustache
│   └─interface.mustache
├─console
│   ├──analysis_swagger.py
│   ├──application.py
│   ├──new_command.py
│   └─swagger_command.py
├─msg
│   ├──weixin.py
│   └─youdu.py
├─operation
│   ├──excel_file.py
│   ├──ini_file.py
│   ├──xml_file.py
│   └─yaml_file.py
├─base_assert.py
├─base_requests.py
└─base_utils.py
```

**config**：中放的是一些配置文件

**config.yaml**：其它默认配置项

**parameConfig.yaml**：黑名单列表和默认参数等配置，在生成swagger接口时，会把匹配到的参数进行替换和过滤

**cases.mustache**：模生成case接口时的板文件，一般不需要动

**interface.mustache**：模生成swagger接口时的板文件，一般不需要动

**console**：命令行工具的一些实现方法，现实现了新建项目、生成swagger方法二个功能，这块可以不用了解

**msg**：消息模块

**robot.py**：企业微信机器人消息模块

**weixin.py**：企业微信消息模块

**operation**：封装了四种文通用件的操作方法

**excel\_file.py**：excel 文件的操作方法

**ini\_file.py**：ini 文件的操作方法

**xml\_file.py**：xml 文件的操作方法

**yaml\_file.py**：yaml 文件的操作方法

**base\_assert.py**：封装了一些通用断言，方便后续调整

**base\_requests.py**：封装了requests库

**base\_utils.py**：封装了一些工具方法，比如：获取文件名、MD5、生成手机号、生成邮箱等

## 3.1 msg 模块

消息模块，封装了 企业微信机器人 消息推送功能

### 3.1.1 企业微信机器人消息

当前微信机器人消息封装了5种消息格式，分别为 文本、卡片、markdown、发送图片、发送文件消息，可针对不同场景使用对应消息，消息样式如下



### 3.1.1.1 文本消息

send\_message\_text() 函数可发送文本消息，需要传三个参数，调用方式如下：

```
send.send_message_text(hookkey, content, mentioned_mobile_list)
```

**hookkey**：webhook的key

**content**：消息内容

**mentioned\_mobile\_list**：手机号列表，提醒手机号对应的群成员(@某个成员)，例如：  
["13800001111"]

例：

```
from luban_common.msg.robot import weixin

send = weixin()
send.send_message_text(hookkey="ae0fdeb8-8b10-4388-8abb-d8ae21ab8d42",content="发现一个公众号：彪哥的测试之路，很不错。",mentioned_mobile_list=["13916829124"] )
```

### 3.1.1.2 图文卡片消息

send\_message\_card() 函数可发送卡片消息，传三个参数，调用方式如下：

```
send.send_message_card(hookkey,title,url,content,picurl)
```

**hookkey**：webhook的key

**title**：消息标题

**url**：点击后跳转的链接

**content**：消息内容

**picurl**：图文消息的图片链接，支持JPG、PNG格式，较好的效果为大图 1068X455，小图 150X150。

例：

```
from luban_common.msg.robot import weixin

send = weixin()
send.send_message_card(hookkey="ae0fdeb8-8b10-4388-8abb-d8ae21ab8d42",title="测试开发",content="发现一个公众号：彪哥的测试之路，很不错，可以点击查看更多跳转到网页",url="http://")
```

### 3.1.1.3 markdown消息

send\_message\_markdown() 函数可发送 markdown 消息，需要传二个参数，调用方式如下：

```
send.send_message_markdown(hookkey,content)
```

**hookkey**：webhook的key

**content**：消息内容

例：

```
from luban_common.msg.robot import weixin

send = weixin()
markdown_content = """
># 这是`markdown`消息
```

```
>事 项: <font color=\"info\">公众号</font>
>组织者: @彪哥的测试之路
>
>会议室: <font color=\"info\">上海</font>
>日 期: <font color=\"warning\">2020年8月18日</font>
>时 间: <font color=\"comment\">上午9:00-11:00</font>
>
>请**准时**参加会议。
>
>如需修改会议信息, 请点击: [这里还可以有连接]
(https://work.weixin.qq.com)""
send.send_message_markdown(hookkey="ae0fdeb8-8b10-4388-8abb-
d8ae21ab8d42", content=markdown_content)
```

#### 3.1.1.4 发送图片

send\_image() 发送图片, 最大不能超过2M, 支持JPG,PNG格式, 调用方式如下:

```
send.send_image(hookkey, file)
```

**hookkey**: webhook的key

**imgBase64**: 图片 (base64编码) 最大不能超过2M, 支持JPG,PNG格式

例:

```
from luban_common.msg.robot import weixin

send = weixin()
send.send_image(hookkey="ae0fdeb8-8b10-4388-8abb-
d8ae21ab8d42", file="../../data/20201222101200.png")
```

#### 3.1.1.5 发送文件

send\_file() 发送其它文件, 调用方式如下:

```
send.send_file(hookkey, file)
```

**hookkey**: webhook的key

**file**: 文件相对路径

例:

```
from luban_common.msg.robot import weixin

send = weixin()
send.send_file(hookkey="ae0fdeb8-8b10-4388-8abb-d8ae21ab8d42", file="weixin.py")
```

## 3.2 operation 模块

操作模块、封装了对 excel、xml、ini、yaml 文件的操作方法

### 3.2.1 excel文件

excel操作方法比较简单，暂时未做过多的封装，共封装四个方法，获取总行数、获取总列数、获取指定行、获取指定单元格

#### 3.2.1.1 获取总行数

get\_lines() 可获取excel表的总行数，调用格式如下：

```
from luban_common.operation.excel_file import OperationExcel

oper = OperationExcel(file_path="../../data/Quality_check_lib.xls", sheetID=0)
oper.get_lines()
```

#### 3.2.1.2 获取总列数

get\_cells() 可获取excel表的总列数，调用格式如下：

```
from luban_common.operation.excel_file import OperationExcel

oper = OperationExcel(file_path="../../data/Quality_check_lib.xls", sheetID=0)
oper.get_cells()
```

#### 3.2.1.3 获取指定行

get\_line(number) 可获取excel表的指定行，需要传一个参数，表示要取第几行的数据，调用格式如下：

```
from luban_common.operation.excel_file import OperationExcel

oper = OperationExcel(file_path="../../data/Quality_check_lib.xls", sheetID=0)
oper.get_line(0)
```

**number**：指定行数，从0开始

#### 3.2.1.4 获取指定单元格

get\_cell(number1,number2) 可获取指定单元格数据，需要传二个参数，单元格的x和y轴坐标，调用格式如下：

```
from luban_common.operation.excel_file import OperationExcel

oper = OperationExcel(file_path="../../data/Quality_check_lib.xls", sheetID=0)
oper.get_cell(0,0)
```

**number1**：指定行数，从0开始

**number2** : 指定行数, 从0开始

### 3.2.2 xml文件

待完善...

### 3.2.3 ini文件

配置管理, 现实现了对 ini 文件的读取和写入

#### 3.2.3.1 获取指定节点配置

getConfig() 获取指定节点下的项目信息, 调用格式如下:

```
from luban_common.operation.ini_file import ManageConfig

cf = ManageConfig(file_path='.././data/intranet.ini')
rf = cf.getConfig(section='openapi')
rf["openapiurl"]
```

**file\_path** : 文件路径

**section** : 节点名

#### 3.2.3.2 获取全部节点配置

getAllConfig() 获取全部节点配置信息, 调用格式如下:

```
from luban_common.operation.ini_file import ManageConfig

cf = ManageConfig(file_path='.././data/intranet.ini')
allconf = cf.getAllConfig()
allconf['openapi']["openapiurl"]
```

**file\_path** : 文件路径

#### 3.2.3.3 向指定节点写配置

writeConfig() 写入配置信息到指定的节点, 调用格式如下:

```
from luban_common.operation.ini_file import ManageConfig

cf = ManageConfig(file_path='.././data/intranet.ini')
#写入信息
cf.writeConfig(section='pds', key='cas', value='http://cas.com')
```

**file\_path** : 文件路径

**section** : 节点名

**key** : 配置的key信息



**value** : 配置的value信息

### 3.2.4 yaml文件

yaml操作模板当前只支持获取yaml数据功能，其它功能未实现

#### 3.2.4.1 获取指定的yaml文件

get\_yaml\_data() 传入yaml文件路径，返回yaml文件内的数据，返回类型为dict，调用格式如下：

```
from luban_common.operation import yaml_file

yaml_file.get_yaml_data(file_path='.././data/config1.yaml')
```

**file\_path** : 文件路径

#### 3.2.4.2 获取指定目录下全部yaml文件

get\_yaml\_data\_all() 获取指定目录下全部yaml文件，并返回yaml文件内的数据，返回类型为dict，调用格式如下：

```
from luban_common.operation import yaml_file

yaml_file.get_yaml_data_all(catalogue='.././config/global')
```

**catalogue** : 指定的文件夹路径

#### 3.2.4.3 写yaml文件

writer\_yaml() 写yaml文件，调用格式如下：

```
from luban_common.operation import yaml_file

yaml_file.writer_yaml(file='.././config/global/te.yaml',data=yamldata)
```

**file** : 文件路径

---

## 3.3 base\_requests.py

封装了 requests 库，处理了单点登录时302跳转问题

---

## 3.4 base\_assert.py

封装了大部分assert方法，建议用这里面的方法来进行断言，方便后续基于断言的一些其它应用

### 3.4.1 校验status\_code和code

assert\_all\_code()函数可校验接口的http响应值和接口code值，调用格式如下：

```
Assertions.assert_all_code(response, expected_http_code, expected_code)
```

**response**：响应数据

**expected\_http\_code**：预期的http状态码

**expected\_code**：预期code状态码

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_all_code(response,200,200)
```

注：推荐使用 assert\_code()

### 3.4.2 校验等于预期值

assert\_equal\_value()函数可校实际值是否等于验预期值，调用格式如下：

```
Assertions.assert_equal_value(reality_value, expected_value)
```

**reality\_value**：实际值

**expected\_value**：预期值

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_equal_value(response["code"][0],200)
```

### 3.4.3 校验数据集中指定key的值

assert\_assign\_attribute\_value()函数可校验接口的http响应值和接口code值，只支持 list、str，调用格式如下：

```
Assertions.assert_assign_attribute_value(data, key, expected_value)
```

**data**：校验的data

**key**：预期key

**expected\_value**：预期值

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_assign_attribute_value(response,"key_name","胡彪")
```

### 3.4.4 校验数据集中存在预期值

assert\_in\_value()函数可校验一组数据中是否存在指定的值，只支持 list、str，调用格式如下：

```
Assertions.assert_in_value(data, expected_value)
```

**data**：校验的数据

**expected\_value**：预期值

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_in_value(data, 200)
```

### 3.4.5 校验字典中存在预期key

assert\_in\_key()函数可校验字典中存在指定的key，只支持 dict，调用格式如下：

```
Assertions.assert_in_key(data, key)
```

**data**：校验的数据

**key**：预期key

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_in_key(data, "key_name")
```

### 3.4.6 校验数据集中不存在预期值

assert\_not\_in\_value()函数可校验一组数据中是否不存在指定的值，支持 list、str，调用格式如下：

```
Assertions.assert_not_in_value(data, expected_value)
```

**data**：校验的数据

**expected\_value**：预期值

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_not_in_value(data, "hubiao")
```

### 3.4.7 校验字典中不存在预期key

assert\_not\_in\_key()函数可校验字典中不存在指定的key，只支持 dict，调用格式如下：

```
Assertions.assert_not_in_key(data, expected_key)
```

**data**：校验的数据

**expected\_key**：预期key

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_not_in_key(data, "key_name")
```

### 3.4.8 校验以预期值开头

assert\_startswith()函数可校验以预期值开头，只支持字符串，调用格式如下：

```
Assertions.assert_startswith(response, expected_value)
```

**data**：校验的数据

**expected\_value**：预期值

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_startswith(data["url"], "http://")
```

### 3.4.9 校验以预期值结尾

assert\_endswith()函数可校验以预期值结尾，只支持字符串，调用格式如下：

```
Assertions.assert_endswith(response, expected_value)
```

**data**：校验的数据

**expected\_value**：预期值

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_endswith(data["url"], "163.com")
```

### 3.4.10 校验是否等于None

assert\_isNone()函数可校验指定的值是否为None，调用格式如下：

```
Assertions.assert_isNone(reality_value)
```

**reality\_value**：实际值

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_isNone(data["name"])
```

### 3.4.11 校验时间小于预期

assert\_time()函数可校验实际时间小于预期时间，调用格式如下：

```
Assertions.assert_time(reality_time, expected_time)
```

**reality\_time**：预期的http状态码

**expected\_time**：预期code状态码

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_time(reality_time, expected_time)
```

### 3.4.12 校验字典或列表是否相等

assert\_dictOrList\_eq()函数可校验字典和列表是否相等，调用格式如下：

```
Assertions.assert_dictOrList_eq(reality, expected)
```

**reality**：实际值

**expected**：预期值

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_dictOrList_eq(dict1, dict2)
```

### 3.4.13 校验列表中是否有重复项

assert\_list\_repetition()函数可校验列表中是否有重复项，调用格式如下：

```
Assertions.assert_list_repetition(list)
```

**list** : 实际值

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_list_repetition(list)
```

### 3.4.14 校验code或status\_code

assert\_code()函数可校验response响应体中的code或status\_code状态码，调用格式如下：

```
Assertions.assert_code(response, reality_code, expected_code)
```

**response** : 响应数据

**reality\_code** : 预期的code状态码

**expected\_code** : 实际code状态码

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_code(response, 200, 200)
```

### 3.4.15 校验为空

assert\_isEmpty()函数可校验传入的数据是为空，当传入值为None、False、空字符串""、0、空列表[]、空字典{}、空元组()都会判定为空，调用格式如下：

```
Assertions.assert_isEmpty(reality_value)
```

**reality\_value** : 实际值

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_isEmpty(reality_value)
```

### 3.4.16 校验不为空

assert\_isNotEmpty()函数可校验传入的数据是不为空，当传入值为None、False、空字符串""、0、空列表[]、空字典{}、空元组()都会判定为空，调用格式如下：

```
Assertions.assert_isEmpty(reality_value)
```

**reality\_value** : 实际值

例：

```
from luban_common.base_assert import Assertions

Assertions.assert_isNotEmpty(reality_value)
```

### 3.4.17 校验字典或列表是否相等

调用格式如下：

```
Assertions.assert_dictOrList_eq(reality, expected)
```

**reality**：实际字典或列表

**expected**：预期字典或列表

例：

```
from luban_common.base_assert import Assertions

list1 = [100, "abcd", "公众号：彪哥的测试之路", False, None]
list2 = [100, "公众号：彪哥的测试之路", "abcd"]

Assertions.assert_dictOrList_eq(list1, list2)

# 输出
AssertionError: 二个列表不相等,第一个列表比第二个列表多了: [False, None]
```

### 3.4.18 校验列表中是否有重复项

调用格式如下：

```
Assertions.assert_list_repetition(lists)
```

**lists**：实际值

例：

```
from luban_common.base_assert import Assertions

list3 = ["89010001#89", "89010001#89", "89010001#89", "96003010#96"]

Assertions.assert_list_repetition(list3)

# 输出
AssertionError: 列表中有重复项,重复项为: {'89010001#89': 3}
```

---

## 3.5 base\_utils.py

封装了一些常用工具，比如获取md5、时间戳、文件名、文件大小、生成手机号等方法

### 3.5.1 获取文件MD5

调用格式如下：

```
getFileMD5(file_Path)
```

**file\_Path**：文件路径可以是相对路径，也可以是相对路径

例：

```
from luban_common import base_utils

base_utils.getFileMD5("data/image/178k.png")
```

### 3.5.2 获取文件大小

调用格式如下：

```
getFileSize(file_Path)
```

**file\_Path**：文件路径可以是相对路径，也可以是相对路径

例：

```
from luban_common import base_utils

base_utils.getFileSize("data/image/178k.png")
```

### 3.5.3 获取文件名

调用格式如下：

```
getFileName(file_Path)
```

**file\_Path**：文件路径可以是相对路径，也可以是相对路径

例：

```
from luban_common import base_utils

base_utils.getFileName("data/image/178k.png")
```

### 3.5.4 判断文件是否存在

调用格式如下：

```
file_is_exist(file_Path)
```



**file\_Path** : 文件路径可以是相对路径，也可以是绝对路径

例：

```
from luban_common import base_utils

base_utils.file_is_exist("data/image/178k.png")
```

### 3.5.5 获取字符串的MD5

调用格式如下：

```
getStrMD5(String)
```

**String** : 字符串

例：

```
from luban_common import base_utils

base_utils.getStrMD5("公众号：彪哥的测试之路")
```

### 3.5.6 字符串Sha1加密

调用格式如下：

```
getStrSha1(String)
```

**String** : 字符串

例：

```
from luban_common import base_utils

base_utils.getStrSha1("公众号：彪哥的测试之路")
```

### 3.5.7 字符串转Base64编码

调用格式如下：

```
ToBase64(String)
```

**String** : 字符串

例：

```
from luban_common import base_utils

base_utils.ToBase64("公众号：彪哥的测试之路")
```

### 3.5.8 Base64编码转字符串

调用格式如下：

```
FromBase64(String)
```

**String**：Base64编码的字符串

例：

```
from luban_common import base_utils

base_utils.FromBase64("公众号： 彪哥的测试之路")
```

### 3.5.9 获取时间戳

调用格式如下：

```
getUnix(date=None)
```

**date**：传入的时间，格式为：'2017-05-09 18:31:22'，不传表单获取当前时间

例：

```
from luban_common import base_utils

base_utils.getUnix()
```

### 3.5.10 时间戳转时间

调用格式如下：

```
UnixToTime(unix)
```

**unix**：unix 时间戳

例：

```
from luban_common import base_utils

base_utils.UnixToTime("1598586432000")
```

### 3.5.11 获取近一个月的开始时间

获取近一个月的开始时间,比如今天是2016-12-15 12:25:00，那么返回的时间为2016-11-15 00:00:00，调用格式如下：

```
getRecentMonthOfDay()
```

例：

```
from luban_common import base_utils  
  
base_utils.getRecentMonthOfDay()
```

### 3.5.12 根据年月，返回当月天数

调用格式如下：

```
calday(month, year)
```

**month**：月份

**year**：年份

例：

```
from luban_common import base_utils  
  
base_utils.calday(8,2015)
```

### 3.5.13 执行CMD命令

调用格式如下：

```
shell(cmd)
```

**cmd**：cmd命令

例：

```
from luban_common import base_utils  
  
base_utils.shell("dir")
```

### 3.5.14 生成随机字符串

调用格式如下：

```
generate_random_str(randomlength=8)
```

**randomlength**：随机字符串的长度，默认8位

例：

```
from luban_common import base_utils

base_utils.generate_random_str()
```

### 3.5.15 生成随机邮件地址

调用格式如下：

```
generate_random_mail()
```

例：

```
from luban_common import base_utils

base_utils.generate_random_mail()
```

### 3.5.16 生成随机手机号

调用格式如下：

```
generate_random_mobile()
```

**file\_Path**：文件路径可以是相对路径，也可以是

例：

```
from luban_common import base_utils

base_utils.generate_random_mobile()
```

### 3.5.17 通过数据生成字典

调用格式如下：

```
responseData(indict)
```

**indict**：接口响应的数据

**return**：重新组装dict，之前嵌套的dict 会组装成“dict\_dict”的形式

例：

```
from luban_common import base_utils

base_utils.ResponseData(Response)
```

### 3.5.18 搜索指定html标签内是否有指定文本

调用格式如下：

```
Search_tag_text(url,tag,text)
```

**url**：指定要检查的连接地址

**tag**：指定要检查的html或xml标签，不要尖括号，如:h2

**text**：指定要检查是否存在的文本

例：

```
from luban_common import base_utils

base_utils.Search_tag_text(url="http://www.lubansoft.com",tag="h1",text="鲁班软件")
```

### 3.5.19 获取时间差

调用格式如下：

```
time_difference(start_time,end_time)
```

**start\_time**：开始时间

**end\_time**：结束时间

例：

```
from luban_common import base_utils

base_utils.time_difference(start_time="2020-08-28
12:16:26.132615",end_time="2020-08-28 12:16:27.133615")
```

### 3.5.20 jsonpath封装

调用格式如下：

```
jpath(data,check_key,check_value=None,sub_key=None)
```

**data**：需要获取的数据,类型必须是dict,否则返回False

**check\_key**：检查key,例子中的functionKey

**check\_value**：检查value,辅助定位,例子中的'D-2'

**sub\_key**：检查子key,辅助定位,例子中的openStatus,当指定sub\_key时,只返回sub\_key对应的values,其它数据不返回

**return**：返回一个list,当匹配不到数据时,返回False

例：

```
from luban_common import base_utils
# jsonpath的写法
# jsonpath.jsonpath(data,$..[?(@.functionKey=='D-2')]..openStatus)

base_utils.jpath(data,check_key="functionKey",check_value="D-2",sub_key="openStatus")
```

### 3.5.21 获取全部字典的key

调用格式如下：

```
get_all_key(data)
```

**data** : list或dict数据

例：

```
from luban_common import base_utils

dict5 = {'busiModuleList': {'type': 'array', 'description': '流程类型id列表',
'items': {'type': 'string'}}}
base_utils.get_all_key(dict5)
```

### 3.5.22 获取全部value

可取list和dict的value值，调用格式如下：

```
get_all_value(data)
```

**data** : list或dict数据

例：

```
from luban_common import base_utils

dict5 = {'busiModuleList': {'type': 'array', 'description': '流程类型id列表',
'items': {'type': 'string'}}}
base_utils.get_all_value(dict5)
```

### 3.5.23 字符串类型的列表转为列表

调用格式如下：

```
strListToList(string)
```

**string** : "[ '1', '2', '3' ]" ---> ['1', '2', '3']

例：

```
from luban_common import base_utils

strlist = "['1', '2', '3']"
base_utils.strListToList(strlist)
```

### 3.5.24 生成uuid

调用格式如下：

```
gen_uuid(filter)
```

**filter**：生成的uuid默认会带有 '-'，当filter为False时不过滤 '-'，默认为False

例：

```
from luban_common import base_utils

base_utils.gen_uuid(True)
```

### 3.5.25 生成签名

根据时间戳和secret生成签名，使用场景：请求数据时发送当前时间戳和生成的签名，接受方根据约定的secret和发送过来的时间戳，以相同方式获取签名，如生成的签名一致，表示签名有效，调用格式如下：

```
gen_sign(timestamp, secret)
```

**timestamp**：时间戳

**secret**：秘钥

例：

```
from luban_common import base_utils

base_utils.gen_sign(getUnix(), "123456")
```

### 3.5.26 返回文件绝对路径

通过文件相对路径，返回文件绝对路径，调用格式如下：

```
file_absolute_path(rel_path)
```

**rel\_path**：相对于项目根目录的路径，如data/check\_lib.xlsx

例：

```
from luban_common import base_utils

base_utils.file_absolute_path('data/Quality_check_lib.xls')
```

### 3.5.27 递归替换字典值

调用格式如下：

```
recursion_replace_dict_value(source, replaceDict)
```

**source**：需要替换的字典或字典组成的列表

**replaceDict**：要检查并替换的字段，key为要检查的值，value为需要替换的值

例：

```
from luban_common import base_utils

source = {"ex": null, "state": false, "age": 38}
replaceDict = {"null": None, "false": False}

base_utils.recursion_replace_dict_value(source, replaceDict)

# 输出
{"ex": None, "state": False, "age": 38}
```

## 3.6 Global\_Map.py

全局变量函数

### 3.6.1 set 设置变量

设置变量到全局

调用格式如下：

```
Global_Map.set(key, value)
```

**key**：变量名称

**value**：变量值

```
from luban_common.global_map import Global_Map
Global_Map.set("username", "hubiao")
```

### 3.6.2 sets 设置多变量

调用格式如下：



```
Global_Map.sets(dict_kwargs)
```

**dict\_kwargs** : 添加多个指定变量到全局变量

```
from luban_common.global_map import Global_Map  
  
Global_Map.sets({"公众号": "彪哥的测试之路", "projId": 113692})
```

### 3.6.3 del\_key 删除指定变量

从全局变量中删除指定的变量

调用格式如下：

```
Global_Map.del_key(key)
```

**del\_map** : 需要删除的变量名

例：

```
from luban_common.global_map import Global_Map  
Global_Map.del_key("username")
```

### 3.6.4 get 获取指定变量

从全局变量中获取指定的变量

调用格式如下：

```
Global_Map.get(*args)
```

**args** : 需要获取的变量名，当只有一个变量且名称为'all' 或 没有args参数时，返回全部变量; 当获取多个变量时，只返回存在的变量

```
from luban_common.global_map import Global_Map  
Global_Map.get("username")  
或  
Global_Map().get("username", "age")
```

## 四、框架能力

### 4.1 命令行工具

在 luban-common 安装成功后，系统中会新增如下命令：

**luban** : 核心命令，不可单独执行，必须携带参数

#### 4.1.1 新建项目

`tuban new`：可通过 `new` 快速构建一个完整的项目目录结构，格式如下：

```
tuban new <name>
```

**name**：项目名称

例：

```
tuban new centerApi
```

### 4.1.2 通过Swagger生成接口文件

`tuban swagger`：生成 `swagger` 接口命令，可快速生成接口方法，格式如下：

```
tuban swagger [-p [<...>]] <swagger-url-json> <project-directory>
```

**swagger-url-json**：swagger url 地址（必须要是json地址），必填参数

**project-directory**：生成到指定的目录，必填参数

**-p**：项目名或**basePath**地址，如指定会把他和接口地址合并成新的接口地址（接口文件中的 resource 字段），可选参数

例：生成接口文件到swagger目录下的 `builder` 目录

```
tuban swagger http://192.168.13.197:8989/builder/v2/api-docs builder
```

例：生成接口文件到swagger目录下的 `builder` 目录，且指定项目名为 `builder`

```
tuban swagger http://192.168.13.197:8989/builder/v2/api-docs builder -p builder
```

### 4.1.3 通过Swagger生成Case（推荐）

`tuban swaggerCase`：生成测试用例命令，可快速生成简单测试用例，格式如下：

注：必须要在项目根目录下执行，会在对应的 `swagger` 和 `testcases` 目录下同时生成swagger接口方法和对应测试用例，如果指定了 `-p` 参数时会在 `testcases` 目录下生成对应的项目目录，并把测试用例放在里面

```
tuban swaggerCase [-p [<...>]] [-b] [-t <...>] [-s] <swagger-url-json> <project-directory> <case-directory>
```

**swagger-url-json**：swagger url 地址（必须要是json地址），必填参数

**project-directory**：接口文件生成到的目录，一般为接口所属项目名称，会在 `swagger` 目录下生成指定的目录，也会做为 case 脚本中的引用文件路径，必填参数

**case-directory**：用例生成到的目录，一般为用例分类，会在 `testcases` 目录下生成指定的目录，必填参数

**-p**：项目名或**basePath**地址，如指定会把他和接口地址合并成新的接口地址（接口文件中的 resource 字段），可选参数

**-b** : 是否生成请求体, 当接口有请求体时, 默认生成请求体, 可选项

**-t** : 生成的默认 token fixture 名称, 默认为 `token`, 可选参数

**-s** : 是否生成 swagger 脚本, 默认生成 swagger 脚本, 可选项

例: 生成接口文件到swagger目录下的 `builder` 目录, 生成测试用例到 `center` 目录

```
luban swaggerCase http://192.168.13.197:8989/builder/v2/api-docs builder center
```

例: 生成接口文件到swagger目录下的 `builder` 目录, 生成测试用例到 `center` 目录, 且指定项目名称为 `builder`

```
luban swaggerCase http://192.168.13.197:8989/builder/v2/api-docs builder center  
-p builder
```

#### 4.1.4 发送微信消息

`luban weixin` : 发送 企业微信机器人 消息命令, 格式如下:

```
luban weixin [-m <...>] [-t <...>] [-u <...>] [-o <...>] <hookkey> <content>
```

**hookkey** : webhook连接中的key, 必填参数

**content** : 消息内容, 必填参数

**-m** : 手机号字符串, 多个手机号用|隔开, 如: "13800138000|13700137000", `text` 消息时有效, 可选参数

**-t** : 消息标题, `card` 消息时有效, 可选参数

**-u** : 点击后跳转的链接, `card` 消息时有效, 可选参数

**-o** : 消息类型, 三种消息类型 `text`、`card`、`markdown`, 可选参数, 类型为 `markdown` 时, content 支持微信机器人官方支持的 `markdown` 语法

例: 发送 `text` 消息

```
luban weixin ae0fdeb8-8b10-4388-8abb-d8ae21ab8d42 "彪哥的测试之路" -m  
"13916829124"
```

例: 发送 `card` 消息

```
luban weixin ae0fdeb8-8b10-4388-8abb-d8ae21ab8d42 "彪哥的测试之路" -o "card" -t "测试开发" -u "http://demo.com"
```

例: 发送 `markdown` 消息

```
luban weixin ae0fdeb8-8b10-4388-8abb-d8ae21ab8d42 "# Hello! `彪哥的测试之路`" -o  
"markdown"
```

## 4.2 pytest命令行参数

新增如下命令行参数：

- `--lb-env`：环境配置文件，如 `dev`、`enterprise`、`preRelease`、`release`

通过如下方式可在命令行中指定需要测试的环境配置

```
pytest --lb-env config/dev/config.yaml
```

- `--lb-driver`：UI自动化时使用的 driver 类型可从命令行或配置文件浏览器

```
pytest --lb-driver firefox
```

- `--lb-base-url`：UI自动化或接口自动化时可从命令行或配置文件指定url地址

```
pytest --lb-base-url http://www.lbuilder.cn
```

- `--lb-robot`：指定机器人消息id，当执行失败时，发消息到企业微信

```
pytest --lb-robot ae0fdeb8-8b10-4388-8abb-d8ae21ab8d42
```

注意：

`--lb-env` 参数是必须指定的参数

`--lb-robot` 参数指定后会替换 `--lb-env` 配置文件中的 `robot`，且会忽略 `message_switch` 配置

如果参数不常变，也可直接写在 `pytest.ini` 中，类似如下形式

```
[pytest]
addopts =
    --lb-env=config/release/config.yaml
    --lb-driver=firefox
```

## 4.3 pytest.ini配置

在 `pytest.ini` 文件中新增如下配置：

- `is_local`：是否走本地初始化，为True时走本地配置文件，为False时走线上初始化数据，默认为False
- `message_switch`：消息通知开关，True为开启消息通知，False为关闭消息通知，默认为False
- `success_message`：成功时是否发送消息通知，默认为False
- `is_clear`：用例执行成功后是否清理数据，默认为True
- 默认使用 `pytest-html` 插件生成报告，生成在当前执行目录的 `reports/report.html` 中
- 其它，指定了 `pytest` 的最低版本号为 `7.0`，只到 `testcases`、`testsuites` 下搜索用例

## 4.4 系统自带fixture

系统自带了2个fixtrue

#### 4.4.1 env\_conf

环境配置，合并了 `pytest.ini` 配置中 `--lb-env` 和 `config/global` 文件夹中的 `yaml` 数据，使用字典的方式取值，使用方法为：

```
env_conf.get("center").get("username")
```

例：获取产品 header 信息等

```
class Token:
    """
    通用token登录类
    """
    def __init__(self, username, password, productId, envConf, loginType=None):
        self.productId = productId
        self.username = username
        self.password = password
        self.header = envConf.get("headers").get("json_header")
        self.Login = base_requests.Send(envConf.get('base_url'), envConf)
        self.epid = ""
        self.token = ""
        self.loginType = "CENTER_WEB" if loginType is None else loginType
        self.envConf = envConf
```

**提示：**读取到的 `env_conf` 数据也会同时写入到 `Global_Map` 中，方便数据共享，但 `Global_Map` 中的数据是可以修改的，修改后 `env_conf` 中不会同步修改

#### 4.4.2 base\_url

基础URL，`base_url` 获取的是 `--lb-base-url` 的值，可在 `pytest.ini` 中指定，或在cmd命令中使用 `--lb-base-url` 指定，使用时只要把 `base_url` 当参数传入对应的函数即可，使用方法为：

web框架时使用，暂未使用到

### 4.5 自定义fixture

项目目录下的 `fixtures` 文件夹用来存放自定义 `fixture`，测试启动时会自动匹配和加载 `fixtures` 文件夹下以 `fixture` 开头，且以 `.py` 结尾的文件，前提是这些文件中的函数要指定了 `@pytest.fixture`。

**规范：** `conftest` 中不要出现自定义的 `fixture`，`conftest` 中只要引入 `pytest_plugins = all_plugins()` 即可

`conftest.py` 格式如下：

```
from luban_common.plugin import all_plugins

pytest_plugins = all_plugins()
```

### 4.5.1 登录并获取token的fixture

在 fixtures 文件夹下，新建 fixture\_login.py，代码如下：

```
# -*- coding: utf-8 -*-
# @Time : 2022-12-4 16:38
# @Author : hubiao
# @Email : 250021520@qq.com
# @公众号 : 彪哥的测试之路

@pytest.fixture(scope='session')
def token(env_conf):
    """
    数据管理平台获取登录凭证
    :return:
    """
    resule = public_login.Token(env_conf.get('iworksApp').get('username'),
                                env_conf.get('iworksApp').get('password'), env_conf.get('productId').get('iworksw
                                eb'), env_conf, env_conf['productId']['iworksweb'])
    yield resule.login()
    resule.logout()
```

这样一个自定义的 fixture 就定义完成了

## 五、如何开始

### 5.1 创建项目

定位到需要创建项目的目录，如：`E:\Automation`，然后在命令行中输入如下命令并回车

```
luban new CenterAutomation
```

**luban**：框架提供的命令入口

**CenterAutomation**：项目名称，可修改为自己想要的名称

看到 `Successfully Created CenterAutomation` 表示项目创建成功，生成的项目结构如下

```
├─business
│   └─dev
│       └─public_login.py
│           └─common.py
│               └─__init__.py
├─config
│   └─dev
│       └─config.yaml
│   └─enterprise
│       └─config.yaml
│   └─preRelease
│       └─config.yaml
│   └─release
│       └─config.yaml
│   └─global
│       └─globalConf.yaml
```

```
├─data
├─fixtures
│   └─fixture_login.py
│   └─__init__.py
├─reports
├─swagger
│   └─__init__.py
│   └─builder
│       └─org.py
│       └─__init__.py
├─testcases
│   └─__init__.py
├─testsuites
│   └─__init__.py
│   └─test_center_demo.py
├─utils
│   └─__init__.py
│   └─utils.py
├─.gitignore
├─conftest.py
└─pytest.ini
```

**business**：存放和当前业务相关代码的包

**config**：配置文件夹

**config.yaml**：默认生成4个环境的配置文件，对应“开发”、“企业部署”、“预发布”、“正式”环境，可自己修改

**globalConf.yaml**：全局配置文件，把不会随环境变化或固定的配置放这里，比如产品ID、请求头等

**data**：存放测试数据的文件夹

**reports**：默认测试报告存放文件夹

**swagger**：通过 swagger 生成的接口方法存放在这里，每个项目一个子文件夹，**builder** 是一个演示项目，里面放的是相关接口

**testcases**：测试用例文件夹，后面单接口用例都放这里面

**testsuites**：测试集文件夹，test\_center\_demo.py 是一个测试的demo

**utils**：工具类，用来存放自定义方法

**.gitignore**：默认的git配置

**conftest.py**：定义了大部分通用 **fixture**

**pytest.ini**：pytest 配置文件，有些默认配置

**fixtures**：存放自定义 **fixture**

## 5.2 执行测试

执行测试有二种方式，生成项目时默认会生成一份演示数据，进入 **CenterAutomation** 目录：

**使用默认配置执行**：在命令行中输入如下命令，表示使用 **pytest.ini** 中的默认配置执行测试

```
pytest
```

**指定环境执行**：在命令行中输入如下命令，表示使用 `dev` 环境配置执行测试

```
pytest --lb-env config/dev/config.yaml
```

## 六、项目实战

框架默认已封装了 `运维后台`、`Center`、`iworksAPP`、`iworkswb`、`OpenAPI`、`Bimapp`、`Mylubanweb`、`Bussiness`、`算量` 产品的登录功能，直接调用对应的 `fixture` 即可，目前登录功能封装在 `business/public_login.py` 文件中，后续会封装成pytest库，直接安装即可。

### 6.1 已封装产品

以 `iworkswb` 的进度计划为例，新建一个进度计划的测试用例。

#### 6.1.1 创建测试项目

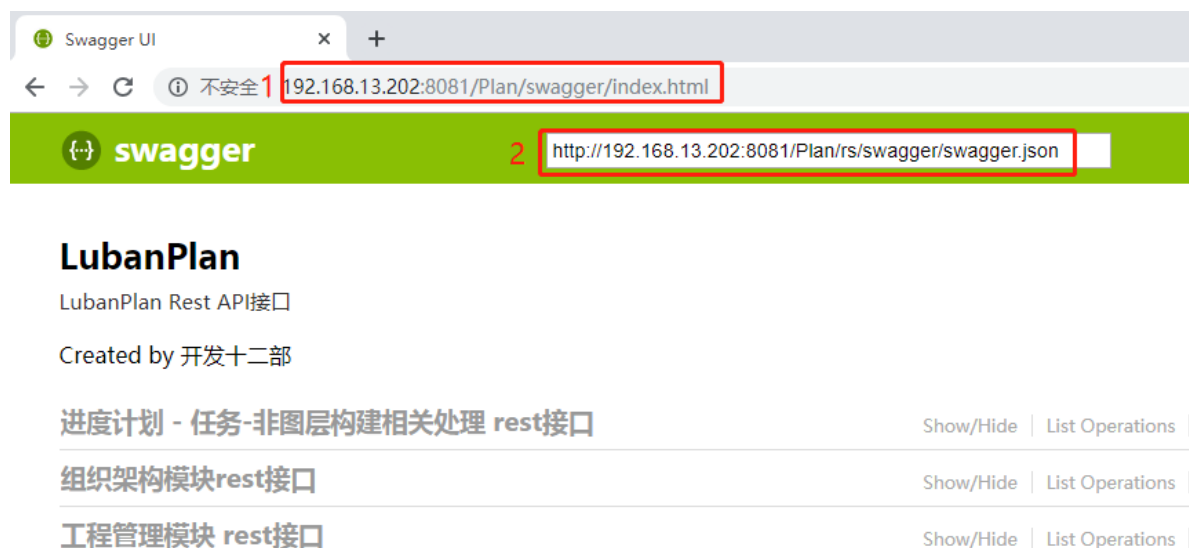
新建一个名称为 `iworkswb` 的测试项目，在 CMD 中进入需要新建项目的目录，并输入命令如下

```
luban new iworkswb
```

看到 `Successfully Created iworkswb` 表示项目创建成功，生成的项目信息可参考“如何开始”，命令问题可查看“命令行工具”中命令的具体介绍

#### 6.1.2 通过swagger生成Case和接口方法

在 CMD 中进入 `iworkswb` 目录，然后找到要生成case的 swagger 接口地址，如下图



第1个是swagger地址

第2个是swagger对应的json地址，这个地址就是我们需要的地址

在命令行中输入如下命令生成用例和接口方法，如果不知道 `luban swaggerCase` 怎么使用，可看前面的说明

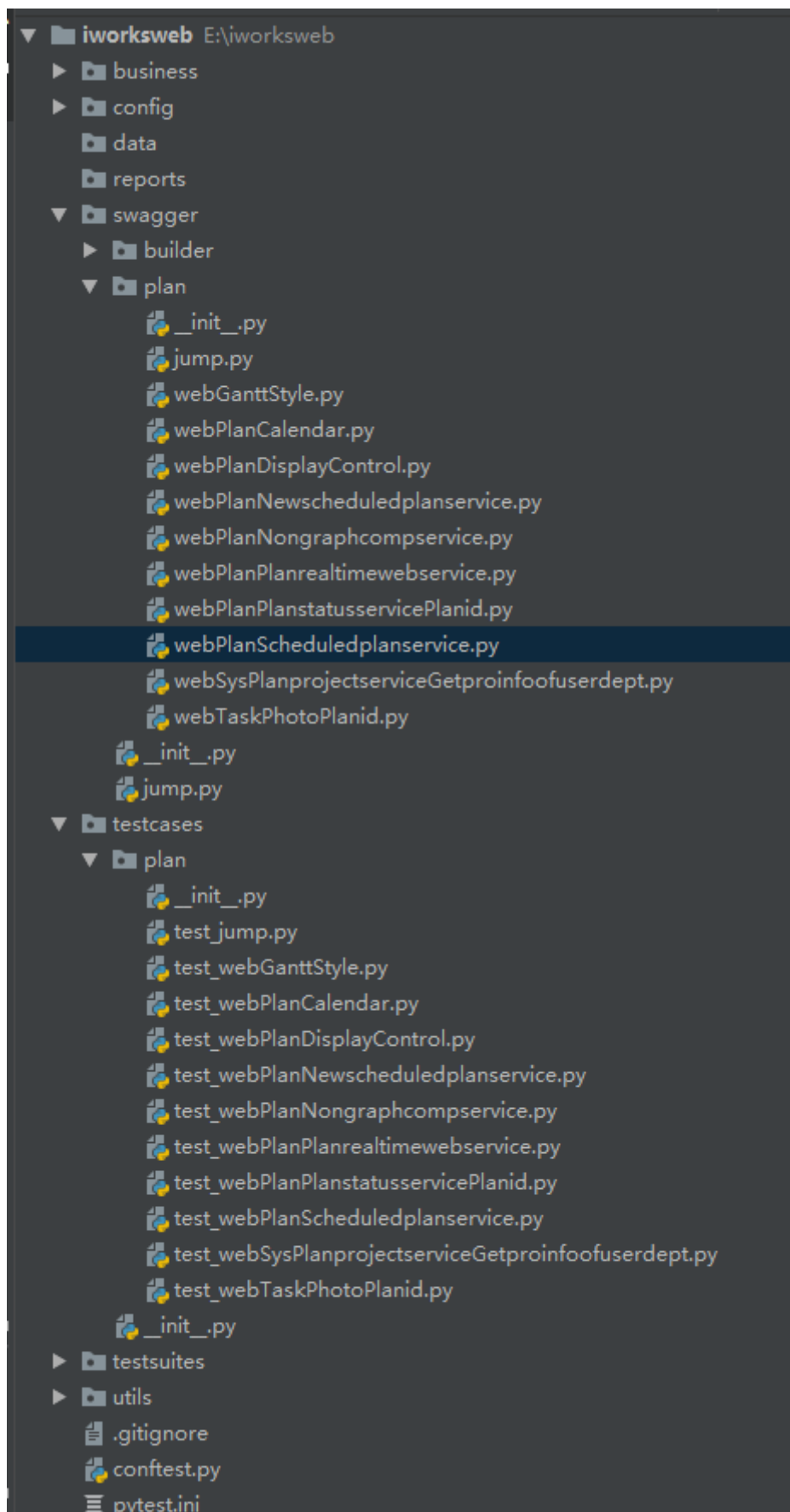


```
luban swaggerCase http://192.168.13.246:8182/Plan/rs/swagger/swagger.json plan
plan
```

```
E:\iworkswb>luban swaggerCase http://192.168.13.246:8182/Plan/rs/swagger/swagger.json plan plan
Created file: E:\iworkswb\swagger\plan\webGanttStyle.py
Created file: E:\iworkswb\swagger\plan\jump.py
Created file: E:\iworkswb\swagger\plan\webPlanPlanrealtimebservice.py
Created file: E:\iworkswb\swagger\plan\webPlanDisplayControl.py
Created file: E:\iworkswb\swagger\plan\webPlanScheduledplanservice.py
Created file: E:\iworkswb\swagger\plan\webPlanCalendar.py
Created file: E:\iworkswb\swagger\plan\webPlanNongraphcompervice.py
Created file: E:\iworkswb\swagger\plan\webSysPlanprojectserviceGetproinfoofuserdept.py
Created file: E:\iworkswb\swagger\plan\webPlanNewscheduledplanservice.py
Created file: E:\iworkswb\swagger\plan\webPlanPlanstatusservicePlanid.py
Created file: E:\iworkswb\swagger\plan\webTaskPhotoPlanid.py
Successfully generate swagger

Created file: E:\iworkswb\testcases\plan\test_webGanttStyle.py
Created file: E:\iworkswb\testcases\plan\test_jump.py
Created file: E:\iworkswb\testcases\plan\test_webPlanPlanrealtimebservice.py
Created file: E:\iworkswb\testcases\plan\test_webPlanDisplayControl.py
Created file: E:\iworkswb\testcases\plan\test_webPlanScheduledplanservice.py
Created file: E:\iworkswb\testcases\plan\test_webPlanCalendar.py
Created file: E:\iworkswb\testcases\plan\test_webPlanNongraphcompervice.py
Created file: E:\iworkswb\testcases\plan\test_webSysPlanprojectserviceGetproinfoofuserdept.py
Created file: E:\iworkswb\testcases\plan\test_webPlanNewscheduledplanservice.py
Created file: E:\iworkswb\testcases\plan\test_webPlanPlanstatusservicePlanid.py
Created file: E:\iworkswb\testcases\plan\test_webTaskPhotoPlanid.py
Successfully generate cases
```

看到 `Successfully generate` 表示接口生成成功，我们用 pycharm 打开 `iworkswb` 项目，生成后的样子如下



打开 `webPlanCalendar.py` 接口文件，看看生成的接口方法是什么样子，查看到 `setPlanCalendar` 方法如下

```

webPlanCalendar.py x
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # @TIME : 2021/09/18 14:55
4  # @Author : system
5  # @File : webPlanCalendar.py
6
7  import allure
8
9  class WebPlanCalendar:
10     """
11     进度计划对应日历配置rest接口
12     """
13
14     @allure.step("设置计划对应日历类型 ")
15     def setPlanCalendar(self, item_fixture, chooseCalendarType=None, planId=None, bodyKwargs=None):
16         """
17         设置计划对应日历类型
18         :param item_fixture: item fixture,
19         :param planId: 计划id
20         :param chooseCalendarType: 日历类型 * 0: 24小时日历(默认) * 1: 标准日历(周六周日休息) * 2: 自定义日历(复制24小时) * 3: 自定义日历(复制标准)
21         :param bodyKwargs: body if bodyKwargs is None else bodyKwargs
22         """
23         resource = f"/web/plan/calendar/calendar_type"
24         body = {'planId': planId, 'chooseCalendarType': chooseCalendarType}
25         payload = body if bodyKwargs is None else bodyKwargs
26         response = item_fixture.request("POST", resource, payload = payload)
27         return response
28

```

生成好的接口文件就可以直接在用例中调用了，调用方式和程序的类和方法调用方式一样没有区别。

打开 `test_webPlanCalendar.py` 文件，看看生成的测试用例是什么样子，查看到

`test_setPlanCalendar` 测试用例如下

```

test_webPlanCalendar.py x
3  # @TIME : 2021/09/18 14:55
4  # @Author : system
5  # @File : test_webPlanCalendar.py
6
7  import ...
11
12  @allure.feature("进度计划对应日历配置rest接口")
13  class Test_WebPlanCalendar:
14     """
15     进度计划对应日历配置rest接口
16     """
17
18     @allure.title("设置计划对应日历类型 ")
19     def test_setPlanCalendar(self, token):
20         """
21         设置计划对应日历类型
22         """
23         body = {'planId': 'integer', 'chooseCalendarType': 'integer'}
24         response = WebPlanCalendar().setPlanCalendar(token, bodyKwargs=body)
25         Assertions.assert_code(response, response.get("status_code"), 200)
26

```

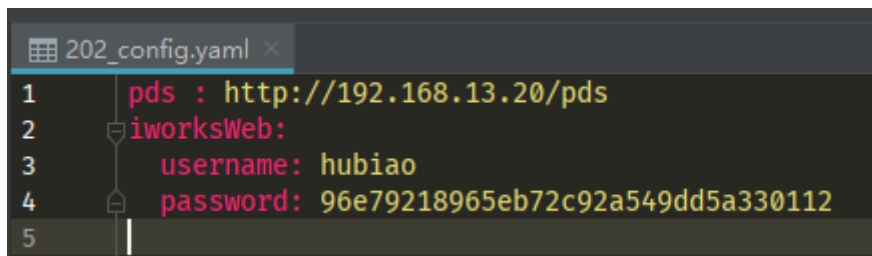
生成的测试用例中，默认会带有请求体（如果有），也可以不生成，只要在生成用例时添加-b 参数即可，可根据自己的实际情况确定是否生成

### 6.1.3 修改测试用例文件

根据情况修改和完善自动生成的用例，比如参数、参数上下文引用、断言添加等，默认会添加http状态断言

### 6.1.4 修改账号配置文件

进入 config 目录，由于现在演示的这个项目是企业部署项目，所以我们进入了 enterprise 目录，我复制了一个yaml 配置文件，命名为 202\_config.yaml，修改后的配置内容如下



```

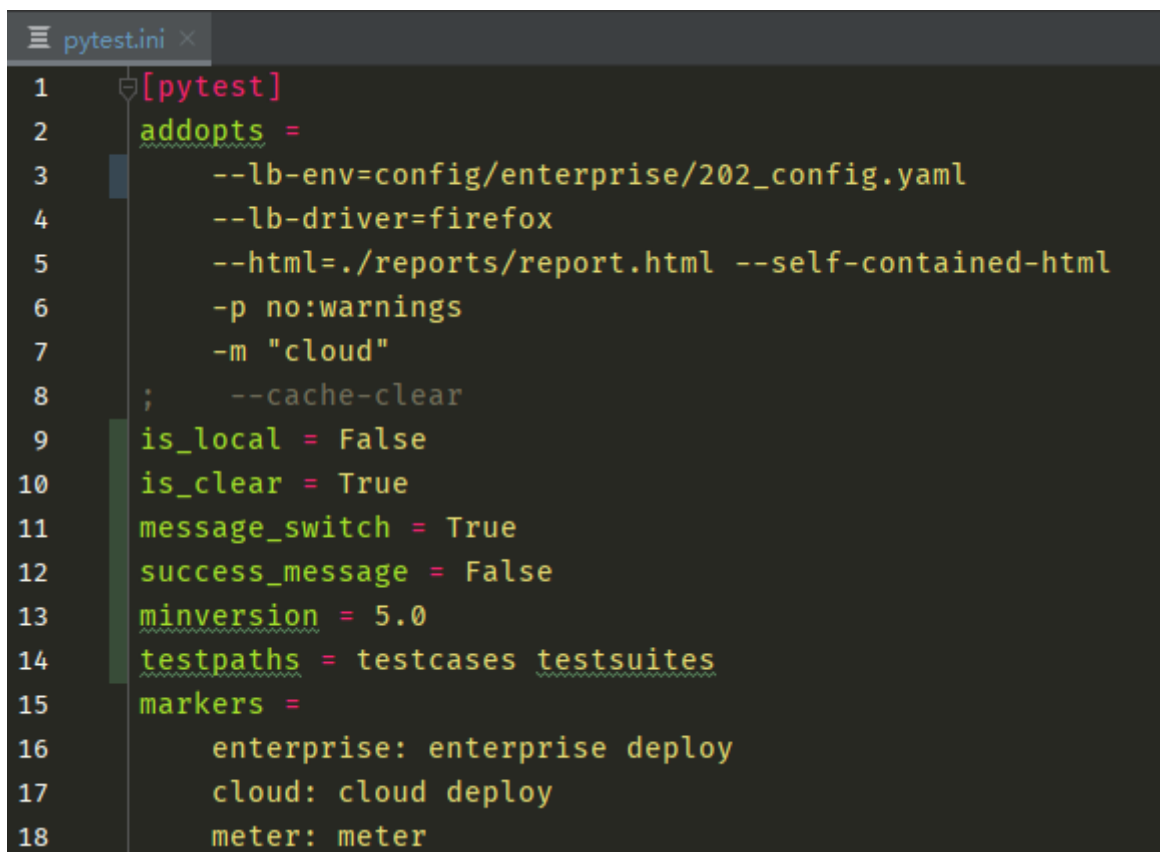
202_config.yaml x
1  pds : http://192.168.13.20/pds
2  iworksWeb:
3      username: hubiao
4      password: 96e79218965eb72c92a549dd5a330112
5

```

**注意：**账号和地址信息必须要按默认文件的方式，建议大家在不了解运行机制时，只修改登录地址、用户名、密码，不要调整格式，如果需要添加信息，按已有样式添加即可

### 6.1.5 修改pytest.ini配置文件

在 `iworkswb` 根目录找到 `pytest.ini` 文件，定位到 `--lb-env` 配置，把 `--lb-env` 配置修改为我们新建的 `Config/enterprise/202_config.yaml` 调整后的样子如下图



```

pytest.ini x
1  [pytest]
2  addopts =
3      --lb-env=config/enterprise/202_config.yaml
4      --lb-driver=firefox
5      --html=./reports/report.html --self-contained-html
6      -p no:warnings
7      -m "cloud"
8      ; --cache-clear
9  is_local = False
10 is_clear = True
11 message_switch = True
12 success_message = False
13 minversion = 5.0
14 testpaths = testcases testsuites
15 markers =
16     enterprise: enterprise deploy
17     cloud: cloud deploy
18     meter: meter

```

### 6.1.6 运行测试

进入 `testcase` 目录，执行 `pytest` 就可以运行case了

## 6.2 未封装产品

### 6.2.1 调整全局配置文件

一般情况不需要调整全局配置文件，由于之前没有把 iworkswb 的产品ID加进来，所以我们要加一下，进入 config 目录，找到 globalConf.yaml 配置文件，这是一个全局配置文件，把不变的配置都放在这里，比如产品ID，请求头信息等，加了一个 `iworksWebProductId: 192` 的配置，修改后的配置内容如下

```
globalConf.yaml x
1  centerProductid : 100
2  iworksAppProductId : 94
3  iworksWebProductId: 192
4  headers:
5      multipart_header : '{"User-Agent": "Mozilla/5.0 (Windows NT 10
6      json_header : '{"User-Agent": "Mozilla/5.0 (Windows NT 10.0; V
7      urlencoded_header : '{"Accept": "text/html,application/xhtml+xml
8      plain_header : '{"Accept": "text/plain", "User-Agent": "Mozilla
9      soap_header : '{"Content-Type": "text/xml; charset=utf-8", "Acce
```

**注意：**建议大家在不了解运行机制时，不要调整格式，如果要添加产品ID，按已有样式添加即可