yamlapi

发布 *1.4.1*

yangjianliang

2021 年 09 月 08 日

Contents

CHAPTER 1

yamlapi

yamlapi 接口测试框架

CHAPTER 2

QQ 群

529063263

CHAPTER 3

工程主页

readthedocs：https://yamlapi-docs.readthedocs.io/zh_CN/latest/pypi：https://pypi.org/project/yamlapi/github：https://github.com/yjlch1016/yamlapi

yamlapi 即为 yaml 文件 +api 测试的缩写可看作是一个脚手架工具可快速生成项目的各个目录与文件支持 unittest 与 pytest 两种运行模式支持 MySQL、PgSQL、MongoDB、Redis 等数据库的增删改查支持 Jenkins、GitLab Runner 等 CI/CD 工具支持飞书、钉钉、企业微信等机器人只需维护一份或者多份 yaml（或者 json）文件即可

`pip install yamlapi` 安装

`yamlapi -h`（或 `yamlapi --help`）查看参数信息

`yamlapi -v`（或 `yamlapi --version`）查看版本号

`pip install -U yamlapi` 安装最新版

`yamlapi create --p=` 项目名称创建项目例如在某个路径下执行命令：`yamlapi create --p=demo_project`

`yamlapi run --c=` 环境缩写运行项目例如在项目的根目录下面执行命令：`yamlapi run --c=test`

`yamlapi clean` 清理测试报告与日志目录下的所有文件类似于 `mvn clean`

`pip uninstall yamlapi` 卸载

CHAPTER 4

一、思路

1、采用 requests+unittest+ddt+PyMySQL+DBUtils+psycopg2-binary+pymongo+redis+influxdb+BeautifulReport+demjson+loguru+PyYAML+ruamel.yaml+pytest+pytest-html+allure-pytest+pytest-reportlog+pytest-assume+pytest-rerunfailures+pytest-instafail+pytest-sugar+pytest-timeout+pytest-parallel+tablib2、requests 是发起 HTTP 请求的第三方库 3、unittest 是 Python 自带的单元测试工具 4、ddt 是数据驱动的第三方库 5、PyMySQL 是连接 MySQL 的第三方库 6、DBUtils 是数据库连接池的第三方库 7、psycopg2-binary 是连接 PgSQL 的第三方库 8、pymongo 是连接 Mongo 的第三方库 9、redis 是连接 Redis 的第三方库 10、influxdb 是连接 influxDB 的第三方库 11、BeautifulReport 是生成 html 测试报告的第三方库 12、demjson 是解析非标格式 json 的第三方库 13、loguru 是记录日志的第三方库 14、PyYAML 与 ruamel.yaml 是读写 yaml 文件的第三方库 15、pytest 是单元测试的第三方库 16、pytest-html 是生成 html 测试报告的插件 17、allure-pytest 是生成 allure 测试报告的插件 18、pytest-reportlog 是替换–resultlog 选项的插件 19、pytest-assume 是多重断言的插件 20、pytest-rerunfailures 是失败重跑的插件 21、pytest-instafail 是实时显示错误信息的插件 22、pytest-sugar 是显示进度的插件 23、pytest-timeout 是设置超时时间的插件 24、pytest-parallel 是多线程的插件 25、tablib 是导出多种格式数据的第三方库

CHAPTER 5

二、目录结构

1、case 是测试用例包 2、report_log 是测试报告和日志的目录 3、resource 是 yaml 文件的目录 4、setting 是工程的配置文件包 5、tool 是常用方法的封装包 6、.dockerignore 是在传递给 docker 引擎时需要忽略掉的文件 7、.gitignore 是.ignore 插件需要排除的文件 8、conftest.py 是全局钩子文件 9、Dockerfile 是构建镜像的文件 10、Jenkinsfile 是 Jenkins Pipeline 文件 11、pytest.ini 是 pytest 的配置文件 12、requirements.txt 是第三方依赖库

CHAPTER 6

三、yaml、json 文件说明

yaml 文件

```yaml
- case_name: 用例名称
  step:
    - step_name: 步骤名称
      mysql:
        -
        -
        -
      pgsql:
        -
        -
        -
      mongo:
        -
        -
        -
      redis:
        -
        -
        -
      request_mode: POST
      api: /api/test
      file:
```

(下页继续)

(续上页)

```
        -
        -
        -
      body:
        { "key_1": "value_1","key_2": "value_2" }
      headers:
        { "Content-Type": "application/json" }
      query_string:
        { "key_3": "value_3","key_4": "value_4" }
      expected_time: 3
      expected_code: 200
      expected_result:
        { "code": 1,"message": "成功" }
      regular:
        variable:
          - name_1
          - name_2
        expression:
          - '"response_1":"(.+?)"'
          - '"response_2":"(.+?)"'
```

json 文件

```
[
  {
    "case_name": "用例名称",
    "step": [
      {
        "step_name": "步骤名称",
        "mysql": [],
        "pgsql": [],
        "mongo": [],
        "redis": [],
        "request_mode": "POST",
        "api": "/api/test",
        "file": [],
        "body": "{\"key_1\":\"value_1\",\"key_2\":\"value_2\"}",
        "headers": "{'Content-Type': 'application/json'}",
        "query_string": "{'key_3':'value_3','key_4':'value_4'}",
        "expected_time": 3,
        "expected_code": 200,
        "expected_result": "{\"code\":1,\"message\":\"成功\"}",
        "regular": {
          "variable": [
```

(下页继续)

(续上页)

```
            "name_1",
            "name_2"
          ],
          "expression": [
            "\"response_1\":\"(.+?)\"",
            "\"response_2\":\"(.+?)\""
          ]
        }
      }
    ]
  }
]
```

1、外层有 2 个字段，内层有 17 个字段命名和格式不可修改，顺序可以修改

2、mysql 字段说明 mysql：MySQL 语句，-列表格式，顺序不可修改，选填

第一行：mysql[0] 第二行：mysql[1] 第三行：mysql[2] 第一行为增、删、改语句，第二行为查语句（动态传参），第三行为查语句（数据库双重断言）第一行是发起请求之前的动作，没有返回结果第二行是发起请求之前的动作，有返回结果，是为了动态传参第三行是发起请求之后的动作，有返回结果，但是不可用于动态传参，是为了断言实际的响应结果当不需要增删改查和双重断言时，可以不写 mysql 字段，或者三行都为空当只需要增删改时，第一行为增删改语句，第二行为空，第三行为空当只需要查时，第一行为空，第二行为查语句，第三行为空当只需要双重断言时，第一行为空，第二行为空，第三行为查语句

3、pgsql 字段说明 pgsql：PgSQL 语句，-列表格式，顺序不可修改，选填

第一行：pgsql[0] 第二行：pgsql[1] 第三行：pgsql[2] 第一行为增、删、改语句，第二行为查语句（动态传参），第三行为查语句（数据库双重断言）第一行是发起请求之前的动作，没有返回结果第二行是发起请求之前的动作，有返回结果，是为了动态传参第三行是发起请求之后的动作，有返回结果，但是不可用于动态传参，是为了断言实际的响应结果当不需要增删改查和双重断言时，可以不写 pgsql 字段，或者三行都为空当只需要增删改时，第一行为增删改语句，第二行为空，第三行为空当只需要查时，第一行为空，第二行为查语句，第三行为空当只需要双重断言时，第一行为空，第二行为空，第三行为查语句

4、mongo 字段说明（参考 mysql 字段）mongo：Mongo 语句，-列表格式，顺序不可修改，选填

第一行：mongo[0] 第二行：mongo[1] 第三行：mongo[2] 第一行为增、删、改，第二行为查（动态传参），第三行为查（数据库双重断言）第一行是发起请求之前的动作，没有返回结果第二行是发起请求之前的动作，有返回结果，是为了动态传参第三行是发起请求之后的动作，有返回结果，但是不可用于动态传参，是为了断言实际的响应结果当不需要增删改查和双重断言时，可以不写 mongo 字段，或者三行都为空当只需要增删改时，第一行为增、删、改，第二行为空，第三行为空当只需要查时，第一行为空，第二行为查，第三行为空当只需要双重断言时，第一行为空，第二行为空，第三行为查

5、redis 字段说明（参考 mysql 字段）redis：Redis 语句，-列表格式，顺序不可修改，选填

第一行：redis[0] 第二行：redis[1] 第三行：redis[2] 第一行为增、删、改，第二行为查（动态传参），第三行为查（数据库双重断言）第一行是发起请求之前的动作，没有返回结果第二行是发起请求之前的动作，有返回

结果，是为了动态传参第三行是发起请求之后的动作，有返回结果，但是不可用于动态传参，是为了断言实际的响应结果当不需要增删改查和双重断言时，可以不写 redis 字段，或者三行都为空当只需要增删改时，第一行为增、删、改，第二行为空，第三行为空当只需要查时，第一行为空，第二行为查，第三行为空当只需要双重断言时，第一行为空，第二行为空，第三行为查

6、file 字段说明 file：文件参数，-列表格式，顺序不可修改，选填

7、函数助手

正则表达式提取的结果用 ${变量名} 匹配，一条用例里面可以有多个 MySQL 查询语句返回的结果，即第二行 mysql[1] 返回的结果，用 {__SQL 索引} 匹配即 {__SQL0}、{__SQL1}、{__SQL2}、{__SQL3}。。。。。一条用例里面可以有多个 PgSQL 查询语句返回的结果，即第二行 pgsql[1] 返回的结果，用 {__PGSQL 索引} 匹配即 {__PGSQL0}、{__PGSQL1}、{__PGSQL2}、{__PGSQL3}。。。。。一条用例里面可以有多个 Mongo 查询语句返回的结果，即第二行 mongo[1] 返回的结果，用 {__MONGO 索引} 匹配即 {__MONGO0}、{__MONGO1}、{__MONGO2}、{__MONGO3}。。。。。一条用例里面可以有多个 Redis 查询语句返回的结果，即第二行 redis[1] 返回的结果，用 {__MONGO 索引} 匹配即 {__REDIS0}、{__REDIS1}、{__REDIS2}、{__REDIS3}。。。。。一条用例里面可以有多个随机数字用 {__RN 位数}，如 {__RN15}，一条用例里面可以有多个随机英文字母用 {__RL 位数}，如 {__RL10}，一条用例里面可以有多个随机手机号码用 {__MP}，一条用例里面可以有多个随机日期时间字符串用 {__RD 开始年份, 结束年份}，如 {__RD2019,2020}，一条用例里面可以有多个以上 9 种类型在一条用例里面可以混合使用 ${变量名} 的作用域是全局的，其它 8 种的作用域仅限该条用例

CHAPTER 7

四、用例 demo

1、包含 mysql 语句的 demo：

```
- case_name: 【进项发票列表高级搜索】根据主键id查询
  step:
    - step_name: 根据主键id查询
      mysql:
        -
        - SELECT id FROM invoice_purchaser_main WHERE purchaser_tenant_id=${tenantId}␣
↪AND purchaser_org_id=${orgId} AND purchaser_company_id=${companyId} AND paper_drew_
↪date BETWEEN '2020-01-01 00:00:00' AND '2020-08-01 00:00:00' ORDER BY id DESC LIMIT␣
↪1;
        - SELECT id FROM invoice_purchaser_main WHERE purchaser_tenant_id=${tenantId}␣
↪AND purchaser_org_id=${orgId} AND purchaser_company_id=${companyId} AND paper_drew_
↪date BETWEEN '2020-01-01 00:00:00' AND '2020-08-01 00:00:00' ORDER BY id DESC LIMIT␣
↪1;
      request_mode: POST
      api: /${tenantId}/invoice/v1/pool/input/invoices/advance-query
      body:
        { "and": [ { "fieldName": "id","fieldValue": "{__SQL0}","operationType":
↪"EQUAL" } ],"sorts": [ { "fieldName": "id","sortNo": 0,"sortType": "DESC" } ],
↪"orgIds": [ "${orgId}" ],"companyIds": [ "${companyId}" ] }
      headers:
        Content-Type: application/json;charset=UTF-8
        xforce-saas-token: ${xforce-saas-token}
```

(下页继续)

(续上页)

```
      userId: ${userId}
    query_string:
      appId: ${appId}
      startPaperDrewDate: 2020/01/01 00:00:00
      endPaperDrewDate: 2020/08/01 00:00:00
      businessFlag: 'true'
      pageNo: 1
      pageSize: 20
    expected_code: 200
    expected_result:
      { "code": "INVOICE0200","message": "请求成功","id": "{__SQL0}" }
```

2、包含 pgsql 语句的 demo：

```
- case_name: 【查询物品列表】精确查询的用例
  step:
    - step_name: 步骤一
      pgsql:
        -
        - SELECT goods_name, salesbill_no, invoice_code, invoice_no, receiver_company_
↪name, receiver_name, receiver_tel, receiver_addr, sender_company_name FROM goods␣
↪WHERE tenant_id = '${tenantId}' ORDER BY create_time DESC LIMIT 1;
        -
      request_mode: GET
      api: /${tenantId}/enterprise/v1/logistics/goods
      headers:
        Content-Type: application/json;charset=UTF-8
        xforce-saas-token: ${xforce-saas-token}
      query_string:
        {
          "appId": "${tenantId}",
          "orderNum": "",
          "goodsName": "{__SQL0}",
          "businessNo": "{__SQL1}",
          "invoiceCode": "{__SQL2}",
          "invoiceNo": "{__SQL3}",
          "receiverCompanyName": "{__SQL4}",
          "receiverName": "{__SQL5}",
          "receiverTel": "{__SQL6}",
          "receiverAddress": "{__SQL7}",
          "senderCompanyName": "{__SQL8}",
          "page": "1",
          "size": "20"
        }
```

(下页继续)

(续上页)

```
    expected_code: 200
    expected_result:
      { "code": "LGSTZZ0200","message": "请求成功" }
```

3、包含 mongo 语句的 demo：

```
- case_name: 熊猫清理-OPPO-牛数-广告激活次留的用例
  step:
    - step_name: 第一步：广告
      mongo:
        -
        -
        - - ad_qili
          - - { "imei": "{__FA0}" }
            - { "product_name": 1,"_id": 0 }
      request_mode: GET
      api: /oppo/common/apis
      query_string:
        {
          "adid": "__ADID__",
          "cid": "__CID__",
          "imei": "{__RN38}",
          "mac": "__MAC__",
          "android_id": "{__RN39}",
          "os": "__OS__",
          "timestamp": "__TS__",
          "campaign_id": "__CAMPAIGN_ID__",
          "aid_name": "__AID_NAME__",
          "csite": "__CSITE__",
          "ctype": "__CTYPE__",
          "product_name": "1812",
          "market_name": "oppo_01",
          "pkg": "com.geek.jk.weather"
        }
      expected_code: 200
      expected_result: { "ret": 0,"msg": "success" }
      mongo_result: [ "1812" ]
    - step_name: 第二步：埋点
      mongo:
        -
        -
        - - data_qili
          - - { "unique_id": "{__FA1}" }
            - { "unique_id": 1,"active_match_status": 1,"active_source": 1,"active_
↪status": 1,"ad_market": 1,"active_day": 1,"active_error_msg": 1,"is_actual": 1,"_id
↪": 0 }
```

(下页继续)

(续上页)

```
request_mode: POST
api: /calculate
body:
  {
    "common": {
      "upload_time": "2020-09-08 10:39:59.250",
      "mpc": "中国联通",
      "screen_width": "720",
      "screen_height": "1440",
      "os_system": "1",
      "os_version": "7.1.1",
      "imei": "{__FA0}",
      "idfa": "",
      "model": "OPPO A83",
      "message_id": "1",
      "sdk": 0,
      "android_id": "{__FA1}",
      "uuid": "{__RN40}",
      "product_name": "1812"
    },
    "events": [ {
      "gender": "未知",
      "app_version": "0.0.9",
      "page_type": "android",
      "latitude": "1111.2222",
      "ip": "172.16.90.44",
      "receive_time": "{__RD2022,2025}",
      "market_name": "oppo-01",
      "event_type": "custom",
      "cv": "1.2.3",
      "user_id": "60066122",
      "event_code": "imei",
      "event_name": "激活",
      "phone_num": "13653363989",
      "network_type": "WIFI",
      "age": "未知",
      "oaid": "",
      "longitude": "1111.1111",
      "province": "provincetest",
      "city": "test",
      "ts": "2020-09-08 10:39:59.250"
    } ]
  }
```

(下页继续)

(续上页)

```
    headers: { "Content-Type": "application/json;charset=UTF-8" }
    expected_code: 200
    expected_result: { "ret": 0,"msg": "success" }
    mongo_result: [ "{__FA1}", "1", "oppo", "1", "oppo_01","{__TODAY}", "上报成功",
↪"1" ]
  - step_name: 第三步：次留
    mongo:
      - - data_qili
        - update
        - - { "unique_id": "{__FA1}" }
          - { "$set": { "active_day": "{__YESTERDAY}" } }
      -
      - - data_qili
        - - { "unique_id": "{__FA1}" }
          - { "unique_id": 1,"active_match_status": 1,"active_source": 1,"active_
↪status": 1,"ad_market": 1,"active_day": 1,"active_error_msg": 1,"is_actual": 1,
↪"next_day_retain_status": 1,"next_day_error_msg": 1,"next_day_retain_day": 1,"_id":␣
↪0 }
    request_mode: POST
    api: /calculate
    body:
      {
        "common": {
          "upload_time": "2020-09-08 10:39:59.250",
          "mpc": "中国联通",
          "screen_width": "720",
          "screen_height": "1440",
          "os_system": "1",
          "os_version": "7.1.1",
          "imei": "{__FA0}",
          "idfa": "",
          "model": "OPPO A83",
          "message_id": "1",
          "sdk": 0,
          "android_id": "{__FA1}",
          "uuid": "{__FA2}",
          "product_name": "1812"
        },
        "events": [ {
          "gender": "未知",
          "app_version": "0.0.9",
          "page_type": "android",
          "latitude": "1111.2222",
```

(下页继续)

(续上页)

```
            "ip": "172.16.90.44",
            "receive_time": "{__RD2022,2025}",
            "market_name": "oppo-01",
            "event_type": "cold_start",
            "cv": "1.2.3",
            "user_id": "60066122",
            "event_code": "cold_start",
            "event_name": "激活",
            "phone_num": "13653363989",
            "network_type": "WIFI",
            "age": "未知",
            "oaid": "",
            "longitude": "1111.1111",
            "province": "provincetest",
            "city": "test",
            "ts": "2020-09-08 10:39:59.250"
          } ]
        }
      headers: { "Content-Type": "application/json;charset=UTF-8" }
      expected_code: 200
      expected_result: { "ret": 0,"msg": "success" }
      mongo_result: [ "{__FA1}", "1", "oppo", "1", "oppo_01","{__YESTERDAY}",
↪"上报成功", "1", "1", "上报成功", "{__TODAY}" ]
```

CHAPTER 8

五、Apollo 配置文件 demo（选用，非必须）

```
test_scenario=测试场景：XXX接口测试
test_story=测试故事：XXX接口测试
test_case_priority=critical
test_case_address=http://www.testcase.com
test_case_address_title=XXX接口测试用例地址
# allure配置
beautiful_filename=xxx_report
beautiful_description=XXX接口测试报告
# BeautifulReport配置
html_report_title=XXX接口测试报告
project_name=XXX接口自动化测试
swagger_address=http://www.swagger.com/swagger-ui.html
test_department=测试部门：
tester=测试人员：
# conftest配置
test_case_format=yaml
# 测试用例的格式：yaml或者json
# 不可混用，只能选取一种格式
first_test_case_file=demo_one.yaml
# 第一个测试用例文件
robot=feishu
# 机器人：feishu、dingtalk或者为空
# 不可混用，只能选取一种
feishu_webhook=https://open.feishu.cn/open-apis/bot/v2/hook/XXXXXX
```

(下页继续)

(续上页)

```
# 飞书机器人webhook
feishu_secret=abcdefghij1234567890
# 飞书机器人密钥
card_header_title_content=飞书消息卡片标题
# 飞书消息卡片标题
card_elements_actions_text_content=飞书消息卡片跳转链接文字
# 飞书消息卡片跳转链接文字
card_elements_actions_url=https://demo.fesihu.com
# 飞书消息卡片跳转链接
dingtalk_webhook=https://oapi.dingtalk.com/robot/send?access_token=XXXXXX
# 钉钉机器人webhook
dingtalk_secret=1234567890abcdefghij
# 钉钉机器人密钥
wechat_webhook=https://qyapi.weixin.qq.com/cgi-bin/webhook/send?key=XXXXXX
# 企业微信机器人webhook
influxdb_switch=true
# 是否插入到influxDB, true或者为空
# 不可混用，只能选取一种
influxdb_host=www.influxdb.com
influxdb_port=8086
influxdb_database=influxdb_database
influxdb_user=root
influxdb_password=123456
influxdb_measurement=influxdb_measurement
# InfluxDB数据库配置
db_host=mysql.test.com
db_port=3306
db_user=root
db_password=123456
db_database=
# MySQL数据库配置
pgsql_host=pgsql.test.com
pgsql_port=5432
pgsql_user=root
pgsql_password=123456
pgsql_database=pgsql_db_1
# PgSQL数据库配置
mongo_host=mongo.test.com
mongo_port=27017
mongo_database=mongo_db_1
mongo_user=root
mongo_password=123456
# Mongo数据库配置
```

(下页继续)

(续上页)

```
redis_host=redis.test.com
redis_port=6379
redis_password=123456
redis_database=0
# Redis数据库配置
```

CHAPTER 9

六、运行

1、unittest 模式：python+ 测试文件名 + 环境缩写 `python case/demo_test.py dev` 开发环境 `python case/demo_test.py test` 测试环境 `python case/demo_test.py pre` 预生产环境 `python case/demo_test.py formal` 生产环境

2、pytest 模式：pytest+ cmd= 环境缩写 `pytest --cmd=dev` 开发环境 `pytest --cmd=test` 测试环境 `pytest --cmd=pre` 预生产环境 `pytest --cmd=formal` 生产环境

3、yamlapi 模式：yamlapi+run+ c= 环境缩写 `yamlapi run --c=dev` 开发环境 `yamlapi run --c=test` 测试环境 `yamlapi run --c=pre` 预生产环境 `yamlapi run --c=formal` 生产环境

4、运行结果：会在 report_log 目录下生成以下文件 allure-reportlog 年月日.logreport.htmlreport.xmltest_case.csvtest_case.htmltest_case.jsontest_case.xlsxtest_case.yaml

CHAPTER 10

七、打包镜像，运行容器

`docker pull registry.cn-hangzhou.aliyuncs.com/yangjianliang/yamlapi:0.0.8` 从阿里云镜像仓库拉取 yamlapi 镜像

`docker build -t demo_image .` docker build -t 镜像名称. 本地打包，demo_image 为镜像名称，随便取

docker run -e cmd=”环境缩写”镜像名称:latest`docker run -e cmd="dev" demo_image:latest` 开发环境 `docker run -e cmd="test" demo_image:latest` 测试环境 `docker run -e cmd="pre" demo_image:latest` 预生产环境 `docker run -e cmd="formal" demo_image:latest` 生产环境

CHAPTER 11

索引和表格

- genindex
- modindex
- search