《接口自动化测试框架》

使用说明书

北京乐步教育科技有限公司

Beijing Lebu Education Technology Co., Ltd

修订记录

日期	作者	版本	备注
2022-07-06	郭志国	1.0	无
2022-08-09	郭志国	1.1	增加了模板示例、一些注意事项等
2022-08-16	郭志国	1.2	增加了(批量)上传文件、content-type 说明等
2023-3-16	郭志国	1.3	增加 jsonschema 校验
2023-3-21	郭志国	1.4	补充完善余下内容并调整结构
2023-4-3	郭志国	1.5	增加 jsonschema 知识内容(截图)
2023-6-14	郭志国	1.6	增加 run 的最新写法

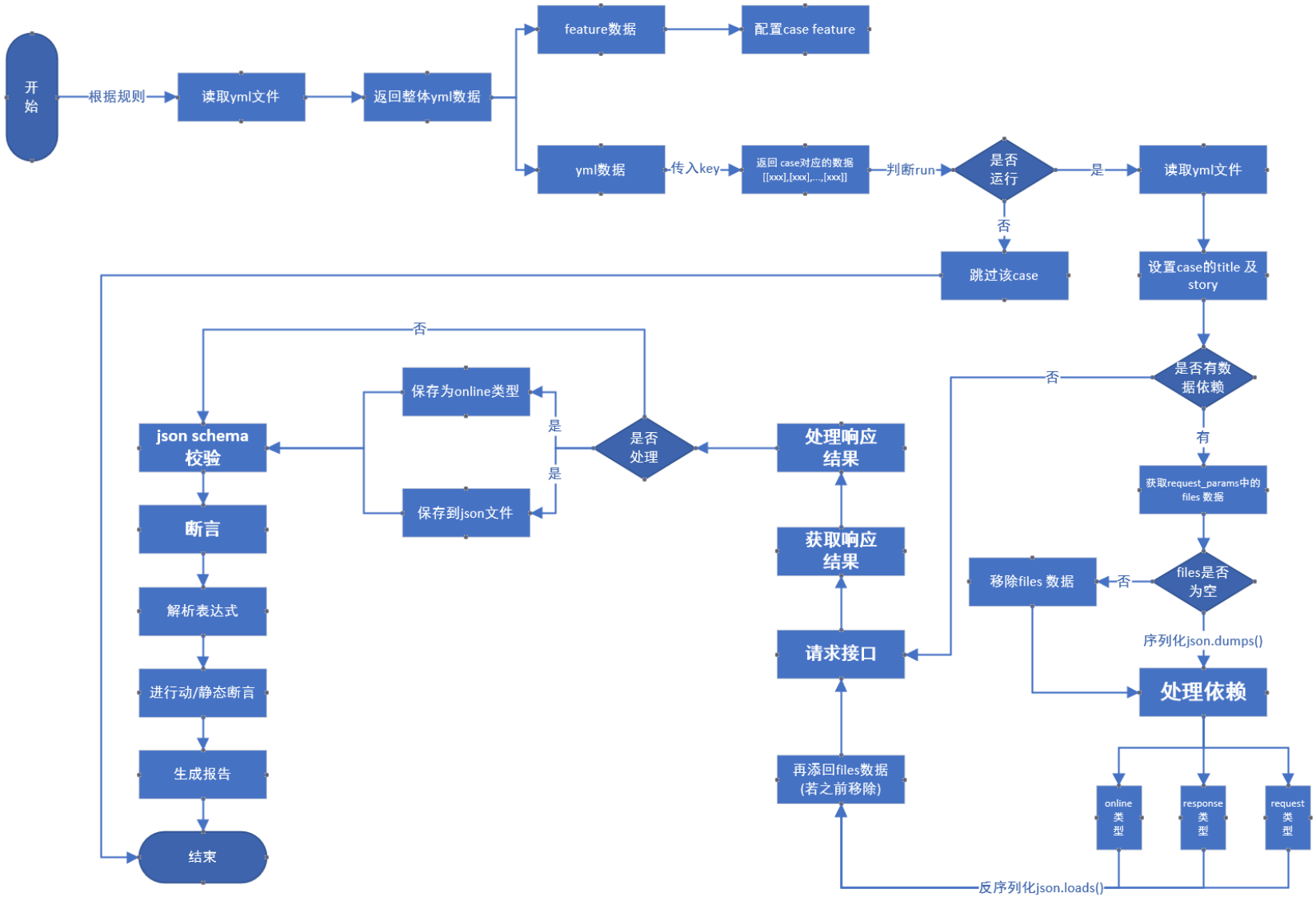
目 录

1 概述

1.1 编写目的

为方便组内成员快速上手接口自动化测试框架，同时也为了统一使用规则，有效减少使用问题，提升工作效率。

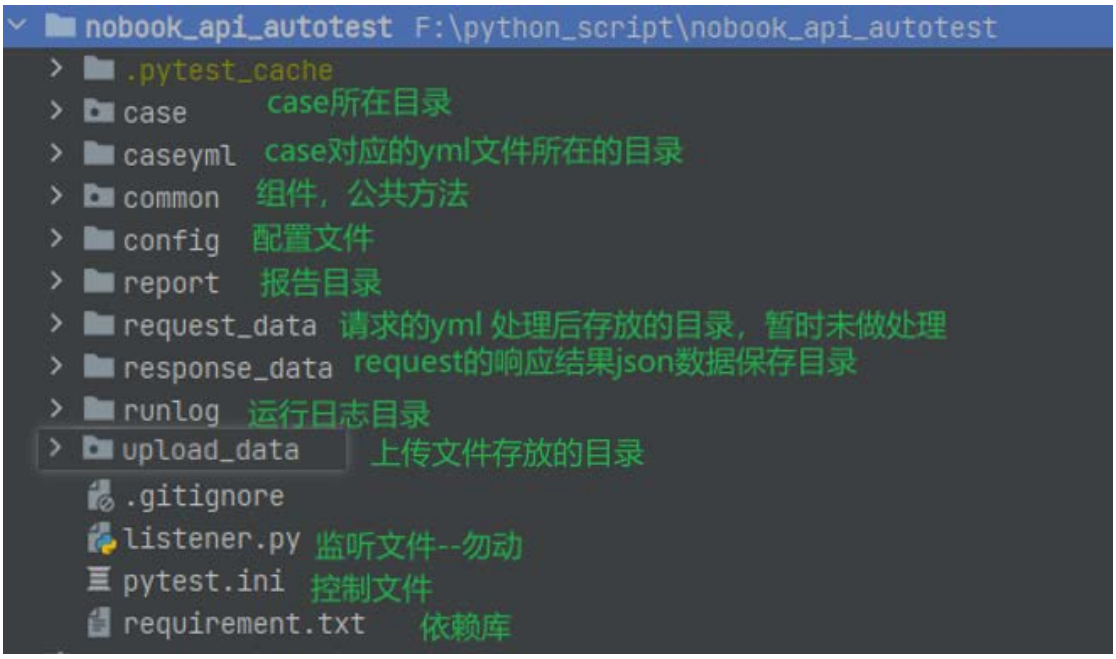
1.2 整体逻辑处理流程图



2 目录介绍

2.1 目录结构

目录结构如下图所示：



2.2 case 目录

所有的 case 的存在的目录

2.3 caseyml

所有的 case 存在的 py 文件对应的 yml 文件所在的目录

2.4 common

所有的组件及公共方法所在的目录

2.5 config

config.yml 配置文件所在的目录

2.6 report

报告所在的目录

2.7 request_data

暂定：case 对应的 yml 文件中的数据，经过处理后的数据，然后转存的 yml 文件所在的目录。目前框架中未启用该目录

2.8 response_data

1. 那些需要保存 request 响应结果的 json 数据的 json 文件所在的目录
2. Jsonschema 文件所在的目录

2.9 runlog

运行日志所在的目录

2.10 upload_data

要上传的文件所存放的目录

2.11 listener.py

Allure 库的 listener.py 文件，经过了修改，在自己本地使用时，请将该文件覆盖本地的 python 虚拟环境中的 allure_pytest 库中的 listener.py 文件。

2.12 pytest.ini

配置文件

2.13 requirement.txt

虚拟环境所依赖的第三方库信息

2.14 .gitignore

该文件里配置了提交代码的一些忽略项

2.15 yaml 文件模板示例

注：该 yaml 文件中不含 jschema_validate 部分，该部分内容请参考 jschema_validate

```yaml
author: guozhiguo
# 需要注意：yml 文件的名称要与 case 所在的 py 文件的文件名称保持一致。
procotol: &procotol https
useragent: &agent Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.77
Safari/537.36
# 用于 class 的 title
feature: 登录功能模块的相关验证
# 功能模块标识
v5login_moduleflag: &v5login_moduleflag v5login
v6console_moduleflag: &v6console_moduleflag v6console
# v5 功能模块的域名
# 测试环境域名
v5login_testhost: &v5login_test storage.noteach.com.cn
# 灰度环境域名
v5login_relhost: &v5login_rel storage-backend-dev.nobook.com
# 生产环境域名
v5login_prodhost: &v5login_prod storage-backend.nobook.com
# v6 功能模块的域名
# 测试环境域名
v6console_testhost: &v6console_test console-v6.noteach.com.cn
# 灰度环境域名
v6console_relhost: &v6console_rel console-v6.nobook.com
# 生产环境域名
v6console_prodhost: &v6console_prod console-v6.nobook.com
```

```yaml
login:
  run:
  story: 登录
#  所属的功能模块组，不同的模块组对应的域名不一样
#  module_flag 的值会作为对应的 host 域名值所对应的 key 的前缀
#  如当前的 module_flag 为 v5login,那么必然存在
#  v5login_testhost、v5login_relhost、v5login_prodhost 这 3 个 key
#  且这 3 个 key 的值对应着测试环境、灰度环境、生产环境的 host 域名。
  module_flag: *v5login_moduleflag
#  case 的描述/目的
  description:
    - 验证正常登录
    - 验证错误的账号密码能否登录成功
#  设置 request_param 下的 params 是 params 还是 data
  params_type: params
#  域名
  host: *v5login_test
  path: /passport/v5/login/username
  request_param: &req_param
    url: $url
    method: post
    headers:
      Content-Type: application/x-www-form-urlencoded
      User-Agent: *agent
    params: &loginparams
      username: 13718620015
      password: admin123
      platform: web
```

```
      clientinfo: ~
      pid: &pid CZWlTE4lVgz9
#  请求参数
  request_params:
    - <<: *req_param
    - <<: *req_param
      params:
        <<: *loginparams
        username: 13718620016
#  响应结果的处理
  response:
#    要保存的文件路径(含文件名称),当值为 online 时表示使用 setattr 和 getattr 来获取相关的数据。
#    当不使用在线保存响应结果时，filepath 要录入具体的 json 文件在 response_data 目录下的路径名称
#    例：0018.json 而不是/0018.json 或 response_data/0018.json 或 /response_data/0018.json
    - filepath: online
      # 要提取响应结果的哪些值，这些要使用 jsonpath 表达式。resp_keys 的值必须为 list
      resp_keys: [ $.auth_token ]
      # 响应结果提示的字段值在保存时对应的 key
      keys: [ auth_token_0015 ]
    - filepath: 0018.json
      resp_keys: [$.auth_token]
      keys: [0015_token]
#  依赖的数据
  depends_on: ~
#  断言
  assert:
#    一个 case 里的多重断言示例
#    jsonpath 表示式的开头必须为 jsonpath,不能是 jsonpath.jsonpath
#    需要注意的是:当对 response 的 json 结果进行数据提取时，此时 jsonpath
```

```
#    表达式里的变量为 req_json，而不是 req ; 若是获取状态码,表达式为: req.status_code
#    所有的断言必须统一使用 pytest.assume(),不支持其他断言,如自带的 assert()断言
    - [pytest.assume(200==req.status_code),
       "pytest.assume('奥林匹克化学学校' in (jsonpath(req_json,'$.schoolname')))"]
#    - pytest.assume('奥林匹克化学学校' in (jsonpath(req_json,'$.schoolname')))
    - pytest.assume(401==req.status_code)

# 检查登录状态
checklogin:
  run:
  story: 登录
  module_flag: *v5login_moduleflag
  description: ["校验登录状态"]
  params_type: params
  host: *v5login_test
  path: /passport/v5/login/check
  request_param: &req_param1
    url: $url
    method: get
    headers:
      User-Agent: *agent
      Authorization: Bearer $token
    params:
      pid: *pid
  request_params:
    - <<: *req_param1
  response: ~
  depends_on:
#    online: 默认为 true，当炎 False 时，表示要读取文件。online 使用 getattr 来获取相应的值。
```

```
#     当使用在线模式时，case_id 为空，此时的 depend_key 表示是哪个 key
#     当 online 为 False 时，depned_key 的值为 jsonpath 表达式。
#     当 type 为 response 时，此时的 case_id 的值为 json 文件在 response_data 目录下的相对路径名称,如 0018.json
#     当 type 为 request 时，case_id 的值为具体的为所依赖的 case 所在的 yml 文件,如: test_demo.yml::login
#     目前不考虑 request 类型的依赖,因为在同个 yml 文件中可以进行数据引用。
    - online: true
      type: response
      case_id:
      depend_key: [ auth_token_0015 ]
      replace_key: [token]
  assert:
    - pytest.assume(200==req.status_code)

# 创建班级
classcreate:
  run:
  story: 创建班级
  module_flag: *v6console_moduleflag
  description: ["创建班级"]
  params_type: json
  host: *v6console_test
  path: /v1/class/create
  request_param: &req_param5
    url: $url
    method: post
    headers:
      User-Agent: *agent
      Authorization: Bearer $token
      Content-Type: application/json
```

```
    json:
      name: create_class_01
      subject_id: 1
      grade_phase_id: 2

  request_params:
    - <<: *req_param5
  response:
    - filepath: create_class_id.json
      resp_keys: [$.data.id]
      keys: [ classid_resp ]
  depends_on:
    - online: false
      type: response
      case_id: 0016.json
      depend_key: [ token_0016 ]
      replace_key: [token]
  assert:
    - pytest.assume(200==req.status_code)

# 更新班级
classupdate:
  run:
  story: 更新班级
  module_flag: *v6console_moduleflag
  description: ["更新班级"]
  params_type: json
  host: *v6console_test
  path: /v1/class/update
```

```yaml
request_param: &req_param6
  url: $url
  method: post
  headers:
    User-Agent: *agent
    Authorization: Bearer $token
  json:
    name: class8081
    # id 的值定义一个变量,依赖于 classcreate 的响应结果的 data.id 的值。
    id: $classid_req

request_params:
  - <<: *req_param6
response: ~
depends_on:
  # 当一个接口同时有多个数据依赖时,则为二维 list,否则就为一维 list
  # 如下所示:当前接口依赖于 token 和 classcreate 创建时返回的班级 id
  - - online: false
      type: response
      case_id: 0016.json
      # 想要获取的数据保存在哪个 key 下
      depend_key: [ token_0016 ]
      # 想要替换的请求参数中，哪个$变量 ,与 $后面的变量名保持一致
      replace_key: [token]
    - online: false
      type: response
      case_id: create_class_id.json
      depend_key: [ classid_resp ]
      # 值要与上面的 json 下的 id 的值中所定义的变量一致,即$后面的值。
```

```
      replace_key: [ classid_req ]
assert:
  - pytest.assume(200==req.status_code)
```

3 注意事项

3.1 yml 文件命名规则

1. yml 文件的名称要与对应的 case 所在的 py 文件名保持一致。如：testdemo.py 对应的 yml 文件就应该为 testdemo.yml
2. 所以的 yml 文件名的后缀为.yml , 不建议使用 .yaml
3. caseyml 目录下的 CaseTemplate.yml 为模板 yml，请勿变动。

3.2 case yml 文件中 key 及变量规则

主要介绍 yml 中的一些需要注意点及容易犯错的地方。

3.2.1 feature

该 yml 文件中的 case 所属于哪个功能版块，统筹该 yml 下的所有 case。

3.2.2 run

1. 该值是用来控制该用例是否运行。默认是运行的。该 key 对应的 item 值可以为空，也支持为 list。当为 list 时，表示下面的多个 case 中有不需要执行的 case。
2. 当其值的 list 时，list 中的元素个数可以与下面的要执行的 case 的个数不一致。如 case 有 23 个，而 list 中的元素个数为 18 个，那么没有写的 5 个则表示全部执行。因为只在当值为 False 时，才会不执行，其他的任何值均表示执行。
3. 之前是不支持 run 的值为 list，之所以改为支持，是因为有些功能下线时，我们要注释很多 case，极有可能注释错。所以为了更加方便的标记不需要执行的 case，所以直接将不需要执行的 case 对应的 run 的标识标记为 False 即可。
4. 当多个 case 中有需要不执行的 case 的写法，见下方的贴图

```
# 写作业
submit_homework:
#  一共有23个case,其中7-14(8个case),17,18 这些case不需要运行,其他的case都需要执行
#  下面的run的值只有18个,最后的那5个case对应的run值虽然没有写,但是仍然会被执行.
  run: [True,True,True,True,True,True,
        False,False,False,False,False,False,False,False,
        True,True,False,False,]
  story: 15-提交导学案作业
  module_flag: *v6console_moduleflag
  description:
    - 【作业流程1-2】验证提交初中化学作业是否成功
    - 【作业流程2-2】验证提交高中化学作业是否成功
    - 【作业流程3-2】验证提交初中物理作业是否成功
    - 【作业流程4-2】验证提交高中物理作业是否成功
    - 【作业流程5-2】验证提交初中生物作业是否成功
    - 【作业流程6-2】验证提交高中生物作业是否成功
```

3.2.3 story

小功能模块，如登录，为了方便报告更新清晰，同个 feature 下，stroy 的命名规则如下：序号-story 名称，如：1-登录模块、2-检查登录模块，最后的报告展示样例如下：

3.2.4 module_flag

功能模块标记，和域名所对应的 key 有一定的关联，具体规则如下：

1. 所属的功能模块组，不同的模块组对应的域名不一样
2. module_flag 的值会作为 对应的 host 域名值所对应的 key 的值的前缀，如下所示：
   a) 如当前的 module_flag 为 v5login，那么必然存在着 v5login_testhost、v5login_relhost、v5login_prodhost
   b) 这 3 个 key 的值分别对应着测试环境、灰度环境及生产环境的 host 域名。

module_flag 与 host 的逻辑规则如下：

一、case 自己的 module_flag 有值时，则 根据 当前的 环境标记(cur_host_flag)进行如下的逻辑判断：

1. 优先通过 从 config.yml 中获取对应的域名
    i. 当 config.yml 中不存在时，则再 case 所在的 yml 文件中获取
        1. 若 case 所在的 yml 文件中 仍然不存在，则使用 case 自己的 host 值(后续有可能将逻辑变更为:若是 case 所在的 yaml 文件中仍然不存在则直接将当前的 case 置为 fail,不在运行该 case)

二、case 自己的 module_flag 无值时

1. 使用 case 自己的 host(后续有可能将逻辑变更为:直接将 case 置为 fail,不在运行)

module_flag 的值 与 host 环境 key 的对应关系如下：

1. module_flag: xxxx 对应的 host 环境的 key 则为 xxxx_testhost,xxxx_devhost,xxxx_prodhost
2. 如：module_flag: v5login 则对应的 host 各个环境的 key 如下：
   v5login_testhost: xxxxx
   v5login_relhost: xxxx
   v5login_prodhost: xxxxx

各个 host 环境的值 优先级：

1. jenkins > config.yml > case 所在的 yaml 文件中的 域名环境 > case 自己的 host 值

若遇到亲的 module_flag 值(如 v7demo)时，要进行如下操作

1. 假设 module_flag: v7demo
2. 先在 case 所在的 yml 文件中 添加 v7demo_testhost: xxxxxx(测试环境域名)、v7demo_relhost: xxxxxx(灰度环境域名)、v7demo_prodhost: xxxxxx(生产环境域名)
3. 然后再在 config.yml 中的 http_conf 下，将刚添加的 v7demo_testhost: xxxxxx(测试环境域名)、v7demo_relhost: xxxxxx(灰度环境域名)、v7demo_prodhost: xxxxxx(生产环境域名)再添加一次。

如何切换当前环境的 host，请参考《cur_host_flag》

3.2.5 descritption

该 case 的目的，用于生成报告图表中的 case 的 title。

3.2.6 params_type

指的是 request 请求是参数是 params 还是 data 或者 json。其值与 rquest_param 下的 parms 的 key 一致。如 params_type 的值为 data 时，则 request_param 下的 params 就要变更为 data

3.2.7 response

Request 的响应结果是否需要怎样处理，以及怎样处理。通常用于数据依赖和 token 依赖。若是不需要处理响应结果，其值直接为空即可。

3.2.7.1 filepath

其值为两种。第一种值为 online，此时表示使用在线的方式，来处理数据依赖；第二种为 xxx.json，指的是将 request 的响应结果的 json 数据保存到 response_data 目录下的 xxx.json 文件中。如 0015.json 表示的是将 response json 数据保存到 response_data 目录下折 0015.json 文件中。如果写的是 response_data/0015.json ，则表示的是将结果保存到 response_data 目录下的 子目录 response_data 下的 0015.json 文件中.

3.2.7.2 resp_keys

表示的是要将响应结果的哪些 key 的值 提取出来。需要注意的是 resp_keys 的值为 list，且均为 jsonpath 表达式。如：[$.auth_token]
若需要对当前接口的响应结果中的多个值进行保存，如返回的结果示例数据为

```
{
    "count":10,
    "data":{
        "id":102,
        "name":"zhangsan",
        "age":10,
        "class":3
    }
}
```

现需要保存 id、name 和 class 三个字段的值，则写法如下：
resp_keys: [$.data.id , $.data.name , $data.class]

3.2.7.3 keys

将提取的结果保存到 json 文件中的哪个 key 下。其值为 list。 [0015_token]
如上图所示，需要对 id、name 和 class 三个字段的值进行保存，正确的写法如下：
keys: [dataid , dataname , dataclass]
需要注意的是 keys 中的 list 的值的顺序要与要 resp_keys 中 list 中的值 一一对应。表示的是将 data 下的 id、name、class 三个 key 对应的值，分别存入到一个 json 文件中的 dataid、dataname、dataclass 下。即最终的 json 文件展示的效果如下：

```
{
    "dataid":102,
    "dataname":"zhangsan",
    "dataclass":3
}
```

3.2.8 depends_on

数据依赖，指的是当前 case 在发送请求前，需要将请求参数中哪些字段的值进行替换。
若是没有数据依赖，直接为空即可。
需要注意的是：当一个接口同时有多个依赖时，则该 case 对应的 depends_on 的值是一个二维 list；若只有一个依赖，则该 case 对应的 depends_on 的值是一个一维 list。示例可参考 2.15 yaml 文件模板示例

```yaml
depends_on:
  # 当一个接口同时有多个数据依赖时,则为二维 list,否则就为一维 list
  # 如下所示:当前接口依赖于 token 和 classcreate 创建时返回的班级 id
  - - online: false
      type: response
      case_id: 0016.json
      # 想要获取的数据保存在哪个 key 下
      depend_key: [ token_0016 ]
      # 想要替换的请求参数中，哪个$变量 ,与 $后面的变量名保持一致
      replace_key: [token]
    - online: false
      type: response
      case_id: create_class_id.json
      depend_key: [ classid_resp ]
      # 值要与上面的 json 下的 id 的值中所定义的变量一致,即$后面的值。
      replace_key: [ classid_req ]
```

3.2.8.1 online

指的是数据的方式，是读取的在线的数据变量，还是读取的 json 或 yml 文件。当 online 的值为 true 时，表示读取的在线 online_data 对象的属性值；当 online 的值为 false 时，表示读取的是 json 或 yml 文件。
online 的默认为 true，即使为空 也表示 online 的值为 true。当 online 的值为 true 时，无需填写 type 和 case_id 的值。

3.2.8.2 type

当 online 为 false 时，需要填写该值。其值为固定的两个值，reponse 和 request。目前框架中未涉及到 request，所以目前只需要写 response 即可。
reponse 表示的是数据依赖为某个接口请求的响应结果
request 表示的是数据依赖为某个接口请求的请求参数值。

3.2.8.3 case_id

当 online 的值为 false 时，需要填写该值。当 type 的值为 response 时，需要填写 json 文件在 response_data 目录下相对于 response_data 目录的 相对路径(含文件名)；当 type 的值为 request 时，需要填写 yml 文件，规则为 case 对应的 yml 文件::case 对应的 yml 文件中的 key，如：testdemo.py::login

3.2.8.4 depend_key

表示想要替换的值在源数据中存于哪个 key 下，其值为 list。具体可参阅 2.15 yaml 文件模板示例

3.2.8.5 replace_key

表示想要替换当前 case 请求参数的哪个参数的值，其值为 list。具体可参阅 2.15 yaml 文件模板示例

3.2.9 assert

断言，于 20220825 支持了动态断言，即支持参数化断言的前半部分(后半部分为 jsonpath 表达式，框架已支持)。

动态断言目前只支持 3 种，分别如下：

1. 取当前请求参数中的某个值
2. 读取某个 json 文件中的某个 key 的值
3. 读取 online_data 某个 属性的值

具体的样例如下：

```
'''
表达式如下：
针对请求参数的写法：
注：“::” 中间的为 jsonpath 表达式，若需要将运行的 jsonpath 表达后提取的结果进行提取时，需要再加上下标(下标从 0 开始,0 表示取第一个值)，如:: 后面的[0] 或 0 均可。
若是不需要提取返回的 list 的具体值，直接使用返回的 list 进行对比时，则不需要加::及后面的[0] 或 0,具体可见下面第 3 个表达式。
$$req_param::$.params.username::[0]$$  或 $$req_param::$.params.username::0$$ 或
$$req_param::$.params.username$$

针对获取 online_data 的固定类 OnlineData 的某个属性写法如下：
$$online_data::token$$

针对读取 json file 某个字段的值如下：
注：json 文件直接写 response_data 目录下的 json 文件相对于 response_data 目录的相对路径(含文件名称)。
例：要读取 response_data 目录下的 0016.json 文件，则如下方所示，直接写 0016.json 即可。不能写 response_data/0016.json，这是一种错误的写法，也不能写./0016.json,这两种写法均是错误的。
$$json_file::0016.json::token$$
'''
针对参数化的动态断言还需要注意以下内容：
```

1. 由于是使用$$进行切割,而在 yml 文件中$$ 会转化为$,所以就写 4 个$,
   如:$$$$req_param::$.params.username::[0]$$$$
2. 在写表达式，需要注意$$$$xxx::xxxx$$$$的外层是否有引号。当$$$$xxx::xxxx$$$$ 返回的值为数字时，若是外面有引号，如”$$$$xxx::xxx$$$$”，则此时就会变为字符串数字，此时它的类型变更为 str，而不是 int；若是$$$$xxx::xxxx$$$$的返回值为字符串时，则需要在外层加上引号，如：” $$$$xxx::xxxx$$$$”
3. 若是需要读取 json 文件时，则在写 json 文件时，应该写相对于 response_data 目录的相对路径，具体可参考上面的注释
4. req_param 指的是当前的请求参数，不支持读取其他 case 的请求参数；online_data 指的是读取 Online_Data 类中的某个属性值；json_file 指是读取某个 json 文件中的某个 key 的值。

常规的断言如下：

- 当一个 case 有多重断言时，其该 case 的断言为 list，如下图所示。

```
- [pytest.assume(200==req.status_code),
   "pytest.assume('奥林匹克化学学校' in (jsonpath(req_json,'$.schoolname')))"]
```

- 断言只支持 pytest.assume()，不支持其他的断言
- jsonpath 表达式的开头必须为 jsonpath,不能为 jsonpath.jsonpath
- 需要想要获取状态码，固定写法: req.status_code
- 如果想要断言响应 json 结果的某个字段，变量名必须为 req_json，如：想要断言成功登录接口的响应结果的学校名称，则提取学校的 jsonpath 表达式为：jsonpath(req_json,'$.shcoolname') ，而完事的断言表达式为: pytest.assume('实验小学' in jsonpath(req_json,'$.shoolname'))

注：错误提示信息 最好每人维护一份自己负责的接口的提示信息 json 文件,将原有的写死的提示信息变更为从 json 文件里读取。这样能很大程度上防止开发修改了提示信息后，我们的脚本能最小幅度最高效率的进行调整。

3.2.10 request_params 下的 files

当接口中需要进行文件上传时，request_params 中有 files 字段，其对应的值为相对于项目的根目录(nb_api_autotest)的相对路径，如 upload_data/1.log,个体如下所示：

```yaml
# v6 文控学生上传文案接口
uploadfile:
  run: false
  story: 上传图片
  #  case 的描述/目的
  description:
    - 上传图片
  #  设置 request_param 下的 params 是 params 还是 data
  params_type: files
  host: *host
  path: /v1/attachment/upload
  request_param: &file_param
    url: $url
    method: post
    headers:
      User-Agent: *agent
      authorization: Bearer $token
    files:
    # 上传单个文件
      files:
        open("upload_data/1.jpg","rb")
---
# v6 文控学生上传文案接口
uploadfile:
  run: false
  story: 上传图片
  #  case 的描述/目的
  description:
    - 上传图片
```

```yaml
#  设置 request_param 下的 params 是 params 还是 data
params_type: files
host: *host
path: /v1/attachment/upload
request_param: &file_param
  url: $url
  method: post
  headers:
    User-Agent: *agent
    authorization: Bearer $token
  files:
  # 批量上传
    - ['files[]','open("upload_data/1.log","rb")']
    - ['files[]','open("upload_data/2.png","rb")','image/png']
    - ['files[]','open("upload_data/1.jpg","rb")','image/jpeg',refer: localhost ]
    - open("1.log","rb")
```

3.2.11 jschema_validate

jsonschema validate 相关配置项，其值为 None 或 list。而 list 中的值的类型要么为 dict 要么为 None，即[{xxxx},{xxxxx},None,{xxxx}]或[{xxxx},{xxxxx},None,None]或[{xxxx},{xxxxx}]。于 2023-03 月份添加此校验规则。

```yaml
# 需要注意的是 jschema_validate 的值为 list，其中 list 中的值要与 assert、request_params 中的 list 中的顺序一致。即：
#  第一组请求参数对应着第一组相关的 assert 断言和 jsonschema 校验。但：若是对应的 jsonschema 不需要校验：
#    若此接口的该组(假设为当前为第二组数据,一共有四组请求数据)请求不需要进行 jsonschema 校验，但是该组后面的请求数据(如第三组)仍然有需要进行 jsonschema 校验，
#    则当前不需要 jsonschema 校验的数据对应的 jschema_validate 必须要有,其值不空(即不填写任何值),如下方的第三组 jsonschema
#    若该组及该组后面的数据均不需要进行 jsonschema 校验,那么从该组开始均可以不写，或者可以写,但值为空. 如下方的第四组 jsonschema(未写)
```

```yaml
  jschema_validate:
    - &demo1
#       是否进行校验，默认值为 true(含空)
      validate: true
#       要校验的 jsonschema 所在位置,默认值为 local(loc),其值为 local、mysql、redis 该 key 为扩展字段,暂不启用
      filelocation: local
#       当为 local 时，则填写 json 文件在 response_data 目录下的文件名称;若为 mysql 或 redis，则写该 jsonschema 对应的 json 字符串值所对应的 key
#       暂不考虑非 local 值的情况，mysql 和 redis 留到日后扩展。
      filepath: 0015_jschema.json
#       是否为第一次。值默认为 false(含空),其逻辑规则如下:
#       当 validate 为 true 或空时且 first 为 true 时,表示首先运行该接口，当不存在对应的 json 文件或有 json 文件，但是读取的内容为空，则表示需要存入
#       jsonscheam 文件。若是本地有 json 文件且读取内容不为空(不进行内容校验),则不会进行任何的 jsonschema 内容变更,此时会直接进行校验。
#       当 first 为 false 时,若没有相对的 jsonschema 内容，则直接报错，不再进行校验。
      first: true
#       是否更新，默认值为 false(含为空时)，表示是否需要对已保存的 jsonschema 的内容进行更新。该字段仅对 filelocation 的值为 mysql 或 redis 开放
      update: false
    - <<: *demo1
      first: false
    -
```

3.2.11.1 逻辑规则

jsonschema 校验，校验的整体规则如下(一期,针对将 jsonschema 存放在本地的 json 文件中)：

1. 当 jschema_validate 的值为 None 或 validate 的值为 False 时，表示不需要进行校验，返回 False
2. 当需要校验时，会优先检测 filepath 的值是否有录入，若没有录入时，会返回 file_error
3. 当 filepath 有录入值时：

a) 会优先判断 filelocation 的值是 mysql、redis 还是 local（非 mysql、redis，其他的值均被认为为 local).当为 mysql 或 redis 时，会返回 False

b) 当值为 local 时，会进入如下的判断：

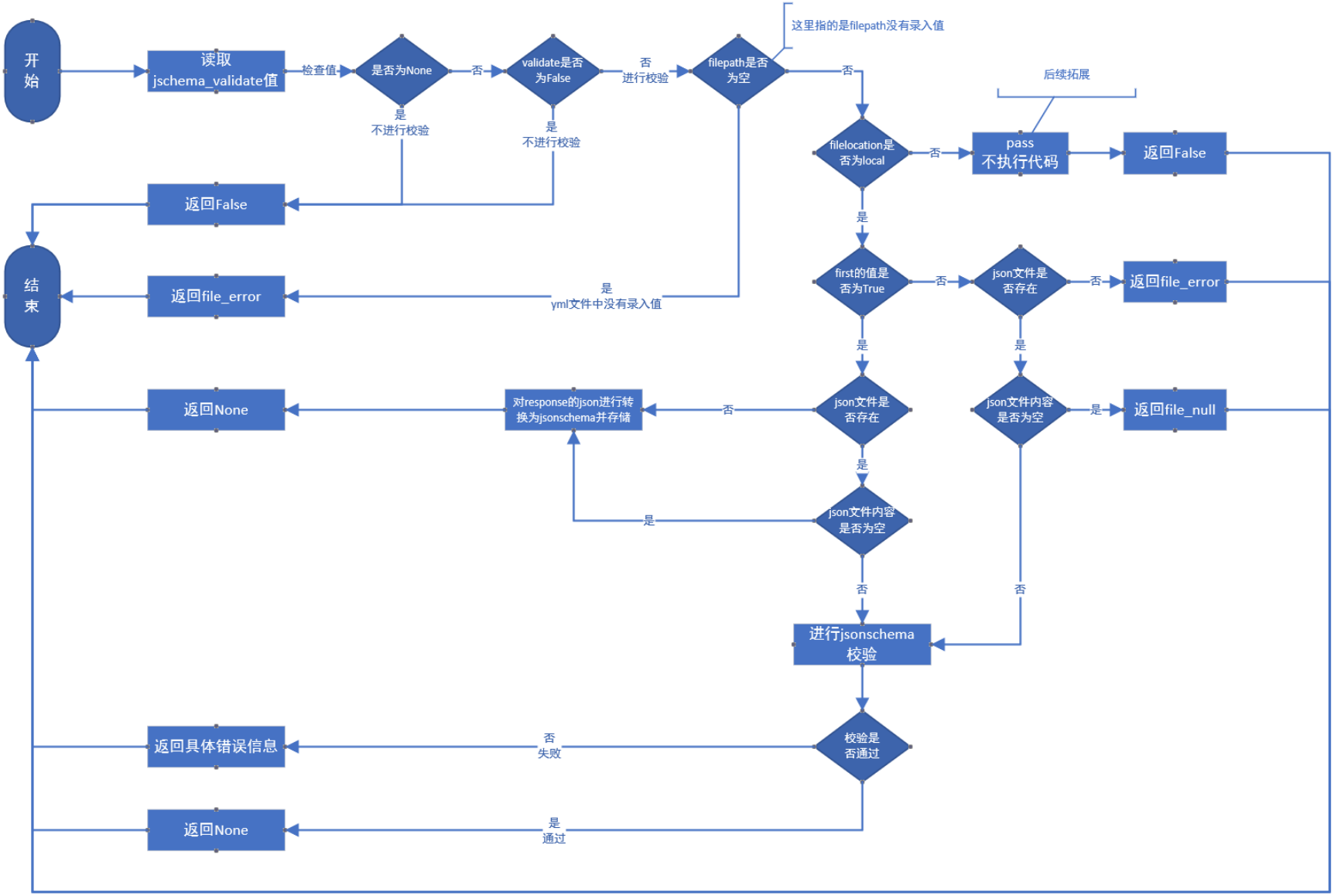
i. first 的值是否为 true，当为 true 时，表示是第一次运行该接口，要将返回的 json 数据进行转换为 jsonschema 并存入到 json 文件中。具体逻辑如下：

1. 当 filepath 里指定的 json 文件不存在时，则会将返回的 json 数据进行转换为 jsonschema，并存入到 filepath 里指定的 json 文件中，并返回 None
2. 当 filepath 里指定的 json 文件存在时，则会读取 json 文件的内容，进行如下的判断：
   a) 当内容为空时，逻辑同 2.1.1，即会将 response 的 json 文件进行转换为 jsonschema，并存入到 filepath 里指定的 json 文件，并返回 None
   b) 当内容不为空时，会直接进行 jsonschema 的校验。校验通过，则返回 None；校验不通过时，则会返回匹配失败的具体信息

ii. 当 first 的值为 false 时：

1. 当 filepath 指定的 json 文件不存在时，则返回 file_error
2. 当 filepath 指定的 json 文件存在时：
   a) 当内容为空时，会返回 file_null
   b) 当内容不为空时，会进行 jsonschema 校验。当校验通过时，返回 None；当校验未通过时，则返回具体的匹配失败的信息。

3.2.11.2 逻辑规则处理流程图



3.2.11.3 validate

是否进行验证，其值有 ture、false 及空，**默认值为 true(**含空)。当值不为 false 时，**其他的任何值均认为为 true。**

3.2.11.4 filelocation

要校验的 jsonschema 所在的位置，其值有 local、loc、mysql、redis 和空，**默认值为 local。当值不为 redis 和 mysql 时，均默认为 local。**
而本期只考虑 local 的情况，即当值为 mysql 和 redis 时，会直接将返回校验结果 False。

3.2.11.5 filepath

jsonschema 文件(json 文件)的名称，当 filelocation 的值为 local 时，此处应该填写 json 文件在 response_data 目录下的文件名称，如：0015_schema.json。
注意点：不要重名，要保证目录下唯一。建议文件名称里有 schema 字符串，和 response 的保存 json 文件区分开。

3.2.11.6 first

是否为第一次，其值为 true 和 false，默认值为 false。当不会 true 时，均被认为为 false。
注意点：

1. 当 validate 的值为 true 时，且 first 的值也为 true 时，若文件不存在或文件存在但是内容为空时，则会将 response 的 json 文件进行转化为 jsonschema，然后存入到 filepath 指定的 json 文件中。若是文件存在，但内容不为空，则会直接进行 jsonschema 校验。

3.2.11.7 update

是否更新 jsonschema 的内容，其值为 true 和 false，默认值为 false，当值不为 true 时，均被认为为 false。该字段仅对 filelocation 的值为 mysql 或 redis 时有效，后续拓展用，本期不考虑。

3.3 config.yml 解读

3.3.1 config.yml 内容

```yaml
#目录配置
dir_conf:
  # case 目录
  case: case
  # case 对应的 yaml 文件所在的目录
  caseyml: caseyml
  # common 目录名称
  common: common
  #config 目录名称
  config: config
#  report 临时目录名称
  report_tmp: report/tmp
#  report 的最终目录
  report: report/report
#  case 的 yaml 文件进行处理后保存的目录名称
  requestdata: request_data
#  保存 request 请求的响应结果的目录名称
  responsedata: response_data
#  运行时日志保存所在的目录名称。
  runlog: runlog
#  上传文件所在的目录名称
  uploaddata: upload_data
```

```yaml
# http 请求相关的配置
http_conf:
#  协议
  procotol: https
##  测试环境的域名
#  host_test: &host_test storage.noteach.com.cn
##  灰度环境的域名
#  host_rel: &host_rel storage.noteach.com.cn
##  生产环境的域名
#  host_prod: &host_prod storage.noteach.com.cn
##  访问的域名
#  host: *host_test

  # 功能模块标识
  v5login_moduleflag: v5login
  v6console_moduleflag: v6console
  # v5 功能模块的域名
  # 测试环境域名
  v5login_testhost: storage.noteach.com.cn
  # 灰度环境域名
  v5login_relhost: storage-backend-dev.nobook.com
  # 生产环境域名
  v5login_prodhost: storage-backend.nobook.com
  # v6 功能模块的域名
  # 测试环境域名
  v6console_testhost: console-v6.noteach.com.cn
  # 灰度环境域名
  v6console_relhost: console-v6.nobook.com
  # 生产环境域名
```

```
  v6console_prodhost: console-v6.nobook.com

#  当前的 host 环境标识，值为 test,rel(release),prod(product)
  cur_host_flag: test
#  接口请求中的 User-Agent 的配置
  useragent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.77
Safari/537.36
```

3.3.2 config.yml 文件中的 key 及注意事项

在 config.yml 里，本文档只介绍 cur_host_flag，因为其他项直接看注释就行。

3.3.2.1 cur_host_flag

cur_host_flag 控制着当前的域名环境，值只有：test、rel、prod 这 3 个值，分别对应着测试环境、灰度环境、生产环境

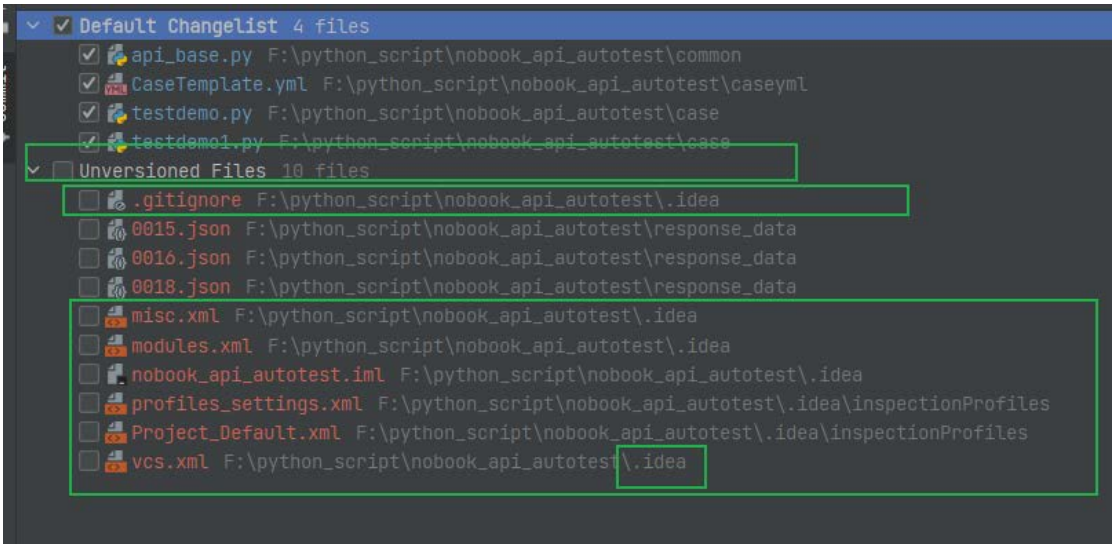
当 cur_host_flag 的值为 test 时，此时会去读取 case 所在的 yml 文件中的 module_flag 的值；然后再查找 config.yml 文件中 http_conf 下的以 module_flag 为开头的，以_testhost 为后缀的 key 所对应的域名。当 cur_host_flag 为 rel 或 prod 时，规则同 test。

示例：

cur_host_flag: test , 而 case 所在的 yml 中的 module_flag 的值为 v6console,那么此时 case 的 host 值规则如下：

1. 优先查找 config.yml 文件中的 http_conf 下的 以 v6console 开头，以_testhost 为结尾的 key，即 v6console_testhost 是否存在。若存在，则读取该 key 的值，作为该 case 的 host。
2. 若是不存在名为 v6console_testhost 的 key，则查找 case 所在的 yml 文件中是否存在该 key。若是存在，则读取该 key 的值，做为该 case 的 host
3. 若 case 所在的 yml 文件中仍然不存在该 key，则此时才会读取 case 的 host key 的值，作为 case 的 host。
4. 当 cur_host_flag 的值为 rel 或 prod 时，规则请参考 test，会优先查找 http_conf 下是否存在名为 v6console_relhost 或 v6console_prodhost 的 key……

3.4 代码提交注意事项



如上图所示，在进行提交代码时，不得提交根目录下的.idea 文件夹下的任何内容

3.5 python 版本及虚拟环境

3.5.1 python 版本

python 的版本区间[3.8,3.10)，即必须为 3.8.x 和 3.9.x 这两个系列的版本。

3.5.2 虚拟环境

3.5.2.1 名称

虚拟环境的名称必须为 python_api_venv

3.5.2.2 包

虚拟环境中的第三方依赖包，必须安装 requirement.txt 中的内容。安装方法如下：
pip install -r requirement.txt
不得安装 requirement.txt 中以外的包。

3.6 删除数据

凡是有新增数据动作的 case，最后都要对数据进行删除，以防垃圾数据堆积。

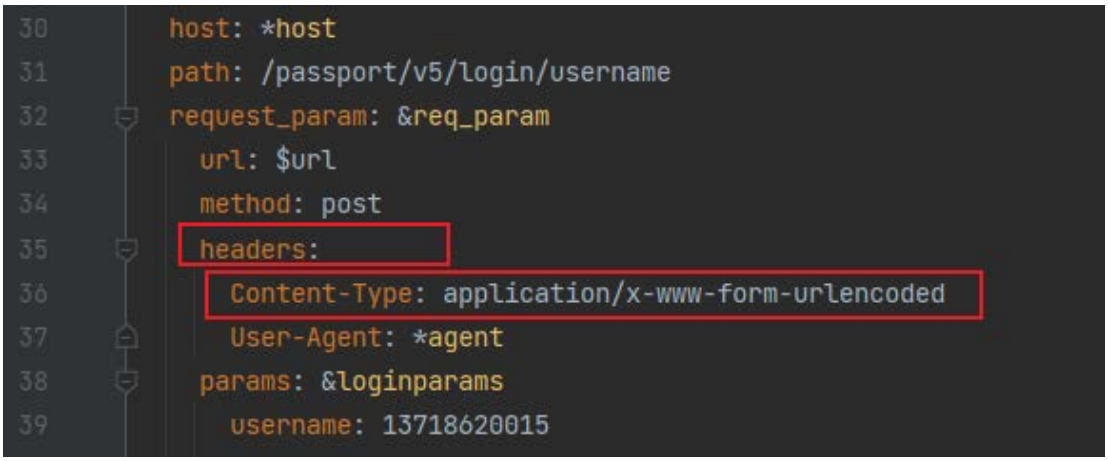
3.7 断言

所有的断言要尽可能的详细、字段尽可能的多，非特殊情况，不得仅仅断言 status_code。尽可的详细、字段尽可能的多：是指不仅仅要断言 status_code，还要对返回的值进行断言。

如：一个班级新增接口，当新增成功后，除了断言 status_code 外，还要断言返回的值中的 名称、创建时间、创建者、类型……等，若是返回的值中，有新增时的字段，则都需要进行校验。

3.8 content-type

在 yaml 文件中 content-type 的位置如下：



在 HTTP 协议消息头中，使用 Content-Type 来表示请求和响应中的媒体类型信息。它用来告诉服务端如何处理请求的数据，以及告诉客户端（一般是浏览器）如何解析响应的数据，比如显示图片，解析并展示 html 等等。

下面主要说明一些最常见的类型,注：如下图所示，**下面所讲述的 params，指的是下图所圈的 request_param 下的 params。**

```yaml
  params_type: params
# 域名
  host: *host
  path: /passport/v5/login/username
  request_param: &req_param
    url: $url
    method: post
    headers:
      Content-Type: application/x-www-form-urlencoded
      User-Agent: *agent
    params: &loginparams
      username: 13718620015
      password: admin123
      platform: web
      clientinfo: ~
      pid: &pid CZWlTE4lVgz9
# 请求参数
request_params:
  - <<: *req_param
  - <<: *req_param
    params:
      <<: *loginparams
```

3.8.1 application/x-www-form-urlencoded

当为此类型时，如果是 get 请求，则 params 的值名称不变；若是 post 请求，则建议将 params 变更为 data，同时还要将 params_type 的值，由默认的 params 变更为 data(参考上图)。

注：若是为 post 请求时，params 为 params 或 data 均可以，建议使用 data。

3.8.2 application/json

当为此类型时，需要将 params 变更为 json，同时 params_type 的值也需要由 params 变更为 json。

3.8.3 multipart/formdata

当为此类型时，若是只有上传文件的参数，则需要将 params_type 的值变更为 files，params 变更为 files；若是还有其他的参数(不仅仅是上传文件，还要传入上传文件之外的其他的字段)时，则 request_param 下必须要有 files，且 files 为上传文件的写法，至于其他的参数，可以放在 params 下，若是放在 params 下 请求失败，则再将 params 变更为 data，再将其他的参数放在 data 下，直至调试成功。或者也可咨询开发。

注：当 params 的名称变更时，一定要变更 params_type 的值，两者必须要保持一致。

情况 1：

```yaml
params_type: files
path: /v1/attachment/upload
request_param: &file_param
  url: $url
  method: post
  headers:
    authorization: Bearer $token
    Content-Type: multipart/form-data
  files:
  # 批量上传
    - ['files[]','open("1.log","rb")']
    - ['files[]','open("2.png","rb")','image/png']
    - ['files[]','open("1.jpg","rb")','image/jpeg' ,refer: localhost ]
    - open("1.log","rb")
```

情况 2：

```yaml
params_type: data
path: /v1/attachment/upload
request_param: &file_param
  url: $url
  method: post
  headers:
    authorization: Bearer $token
    Content-Type: multipart/form-data
  data:
    key1: xxxxxx
    key2: xxxxxx
  files:
  # 批量上传
    - ['files[]','open("1.log","rb")']
    - ['files[]','open("2.png","rb")','image/png']
    - ['files[]','open("1.jpg","rb")','image/jpeg' ,refer:
    - open("1.log","rb")
```

4 附件

4.1 jsonschema 校验流程图

jsonschema validate.vsdx

4.2 整体框架逻辑处理流程图

接口自动化测试框架逻辑处理流程图.v

4.3 Jsonschema 校验知识点

参考链接：

Json Schema 简介 - 苦力笨笨 - 博客园 (cnblogs.com)

在 jsonschema 的 json 文件中，如果一个字段的 type 可能有多个值时，那么该 type 的值为 list，如：type: [“string”,”integer”,null]

```
35          "updated_by": {
36              "type": ["integer","string",null]
37          },
```

Json Schema简介

1. 引言

什么是Json Schema? 以一个例子来说明

假设有一个web api，接受一个json请求，返回某个用户在某个城市关系最近的若干个好友。一个请求的例子如下：

```
{
    "city" : "chicago",
    "number": 20,
    "user" : {
        "name":"Alex",
        "age":20
        }
}
```

在上面的例子中，web api要求提供city，number，user三个成员，其中city是字符串，number是数值，user是一个对象，又包含了name和age两个成员。

对于api来说，需要定义什么样的请求合法，即什么样的Json对于api来说是合法的输入。这个规范可以通过Json Schema来描述，对应的Json Schema如下。

```
{
    "type": "object",
    "properties": {
        "city": { "type": "string" },
        "number": { "type": "number" },
        "user": {
            "type": "object",
            "properties": {
                "name" : {"type": "string"},
                "age" : {"type": "number"}
            }
        }
    }
}
```

例子可以通过Json Schema Validator来验证。

什么是Json Schema?

> **Json Schema定义了一套词汇和规则，这套词汇和规则用来定义Json元数据，且元数据也是通过Json数据形式表达的。Json元数据定义了Json数据需要满足的规范，规范包括成员、结构、类型、约束等。**

本文后面的部分是简要介绍Json Schema定义的这些规则，以及如何用这些规则描述规范。

Json Schema定义了一系列关键字，元数据通过这些关键字来描述Json数据的规范。其中有些关键字是通用的；有些关键字是针对特定类型的；还有些关键字是描述型的，不影响合法性校验。本文的主要内容就是介绍这些关键字的应用。

2. 类型关键字

首先需要了解的是"type"关键字，这个关键字定义了Json数据需要满足的类型要求。"type"关键字的用法如下面几个例子：

1. {"type":"string"}。规定了Json数据必须是一个字符串，符合要求的数据可以是

> "Today is a good day."
> "I love you"

2. {"type" : "object"}。规定了Json数据必须是一个对象，符合要求的数据可以是

> {"name" : "Alexander", "age" : 98}
> {}

3. {"type" : "number"}。规定了Json数据必须是一个数值，符合要求的数据可以是。Java Script不区分整数、浮点数，但是Json Schema可以区分。

> 2
> 0.5

4. {"type": "integer"}。要求数据必须是整数。

> 2

5. {"type" : "array"}。规定了Json数据必须是一个数组，符合要求的数据可以是

> ["abc", "cdf"]
> [1, 2, 3]
> ["abc", 25, {"name": "Alexander"}]
> []

6. {"type" : "boolean"}。这个Json Schema规定了Json数据必须是一个布尔，只有两个合法值

> true
> false

7. {"type" : "null"}。null类型只有一个合法值

> null

3. 简单类型

这部分介绍类型特定的关键，包括字符串、数值、布尔、空值几种基本类型。

3.1 字符串

Json合法的字符串

> "Today is a good day."

对应的Json Schema

>

可以进一步对字符串做规范要求。**字符串长度、匹配正则表达式、字符串格式。**

3.1.1 字符串长度

> 关键字： **minLength, maxLength**

可以对字符串的最小长度、最大长度做规范。

```
{
    "type" : "string",
    "minLength" : 2,
    "maxLength" : 3,
}
```

3.1.2 正则表达式

> 关键字： **pattern**

可以对字符串应满足的Pattern做规范，Pattern通过正则表达式描述。

```
{
    "type" : "string",
    "pattern" : "^(\\([0-9]{3}\\))?[0-9]{3}-[0-9]{4}$",
}
```

3.1.3 字符串Format

> 关键字： **format**

可以通过Json Schema内建的一些类型，对字符串的格式做规范，例如电子邮件、日期、域名等。

```
{
   "type" : "string",
   "format" : "date",
}
```

Json Schema支持的format包括"date", "time", "date-time", "email", "hostname"等。具体可以参考文档。

3.2 数值

Json Schema数值类型包括"number"和"integer"。number合法的数值可以是

> 2
> 0.1

对应的Json Schema为

> {"type": "number"}

如果是integer则只能是整数。"number"和"integer"的类型特定参数相同，可以限制**倍数**、**范围**。

3.2.1 数值满足倍数

> 关键字： **multipleOf**

可以要求数值必须某个特定值的整数倍。例如要求数值必须是10的整数倍。

```
{
    "type" : "number",
    "multipleOf" : 10,
}
```

3.2.2 数值范围

> 关键字： **minimum, maximum, exclusiveMinimum, exclusiveMaximum**

可以限制数值的方位，包括最大值、最小值、开区间最大值、开区间最小值。

要求数值在[0, 100)范围内。

```
{
    "type" : "number",
    "minimum": 0,
    "exclusiveMaximum": 100
}
```

3.3 布尔

布尔类型没有额外的类型特定参数。

3.4 空值

null类型没有额外的类型特定参数。

4. 复合类型

复合类型可以通过Nest的方式构建复杂的数据结构。包括数组、对象。

4.1 数组

Json数组合法数据的例子

> ```
> [1, 2, 3]
> [1, "abc", {"name" : "alex"}]
> []
> ```

Json Schema为

> ```
> {"type": "array"}
> ```

数组的类型特定参数，可以用来限制**成员类型、是否允许额外成员、最小元素个数、最大元素个数、是否允许元素重复。**

4.1.1 数组成员类型

> 关键字： **items**

可以要求数组内每个成员都是某种类型，通过关键字items实现。下面的Schema要求数组内所有元素都是数值，这时关键字"items"对应一个嵌套的Json Schema，这个Json Schema定义了每个元素应该满足的规范。

```
{
    "type": "array",
    "items": {
        "type": "number"
    }
}
```

> ```
> [1, 2, 3]
> ```

关键字items还可以对应一个数组，这时Json数组内的元素必须与Json Schema内items数组内的每个Schema按位置一一匹配。

```
{
    "type": "array",
    "items": [
    {
        "type": "number"
    },
    {
        "type": "string"
    }]
}
```

> `[1, "abc"]`

4.1.2 数组是否允许额外成员

> 关键字： **additionalItems**

当使用了items关键字，并且items关键字对应的是Schema数组，这个限制才起作用。关键字additionalItems规定Json数组内的元素，除了一一匹配items数组内的Schema外，是否还允许多余的元组。当additionalItems为true时，允许额外元素。

```
{
    "type": "array",
    "items": [
    {
        "type": "number"
    },
    {
        "type": "string"
    }],
    "additionalItems" : true
}
```

> `[1, "abc", "x"]`

4.1.3 数组元素个数

> 关键字： **minItems, maxItems**

可以限制数组内元素的个数。

```
{
    "type": "array",
    "items": {
        "type": "number"
    },
    "minItems" : 5,
    "maxItems" : 10
}
```

> `[1,2,3,4,5,6]`

4.1.4 数组内元素是否必须唯一

> 关键字： **uniqueItems**

规定数组内的元素是否必须唯一。

```
{
    "type": "array",
    "items": {
        "type": "number"
    },
    "uniqueItems" : true
}
```

> `[1,2,3,4,5]`

4.2 对象

Json对象是最常见的Json数据类型，合法的数据可以是

```
{
    "name": "Froid",
    "age" : 26,
    "address" : {
        "city" : "New York",
        "country" : "USA"
    }
}
```

就对象类型而言，最基本的类型限制Schema是

> `{"type" : "object"}`

然而，除了类型外，我们通常需要对其成员做进一步约定。对象的类型特定关键字，大多是为此目的服务的。

4.2.1 成员的Schema

> 关键字：**properties**

规定对象各成原所应遵循的Schema。

```
{
    "type": "object",
    "properties": {
        "name": {"type" : "string"},
        "age" : {"type" : "integer"},
        "address" : {
            "type" : "object",
            "properties" : {
                "city" : {"type" : "string"},
                "country" : {"type" : "string"}
            }
        }
    }
}
```

对于上例中的Schema，合法的data是

```
{
    "name": "Froid",
    "age" : 26,
    "address" : {
        "city" : "New York",
        "country" : "USA"
    }
}
```

properties关键字的内容是一个key/value结构的字典，其key对应Json数据中的key，其value是一个嵌套的Json Schema。表示Json数据中key对应的值所应遵守的Json Schema。在上面的例子中，"name"对应的Schema是 `{"type" : "string"}` ，表示"name"的值必须是一个字符串。在Json数据中，对象可以嵌套，同样在Json Schema中也可以嵌套。如"address"字段，在Json Schema中它的内容是一个嵌套的object类型的Json Schema。

4.2.2 批量定义成员Schema

> 关键字：**patternProperties**

与properties一样，但是key通过正则表达式匹配属性名。

```
{
    "type": "object",
    "patternProperties": {
        "^S_": { "type": "string" },
        "^I_": { "type": "integer" }
    }
}
```

> `{"S_1" : "abc"}`
> `{"S_1" : "abc", "I_3" : 1}`

4.2.3 必须出现的成员

> 关键字：**required**

规定哪些对象成员是必须的。

```
{
    "type": "object",
    "properties": {
        "name": {"type" : "string"},
        "age" : {"type" : "integer"},
    },
    "required" : ["name"]
}
```

上例中"name"成员是必须的，因此合法的数据可以是

> `{"name" : "mary", "age" : 26}`
> `{"name" :  "mary"}`

但缺少"name"则是非法的

> `{"age" : 26}`

4.2.4 成员依赖关系

> 关键字：**dependencies**

规定某些成员的依赖成员，不能在依赖成员缺席的情况下单独出现，属于数据完整性方面的约束。

```
{
    "type": "object",
    "dependencies": {
        "credit_card": ["billing_address"]
    }
}
```

dependencies也是一个字典结构，key是Json数据的属性名，value是一个数组，数组内也是Json数据的属性名，表示key必须依赖的其他属性。

上面Json Schema合法的数据可以是

> `{}`
> `{"billing_address" : "abc"}`

但如果有"credit_card"属性，则"billing_address" 属性不能缺席。下面的数据是非法的

> `{"credit_card": "7389301761239089"}`

4.2.5 是否允许额外属性

> 关键字：**additionaProperties**

规定object类型是否允许出现不在properties中规定的属性，只能取true/false。

```
{
    "type": "object",
    "properties": {
        "name": {"type" : "string"},
        "age" : {"type" : "integer"},
    },
    "required" : ["name"],
    "additionalProperties" : false
}
```

上例中规定对象不能有"name"和"age"之外的成员。合法的数据

> `{"name" : "mary"}`
> `{"name" : "mary", "age" : 26}`

非法的数据

> `{"name" : "mary", "phone" : ""84893948}`

4.2.6 属性个数的限制

> 关键字：**minProperties, maxProperties**

规定最少、最多有几个属性成员。

```
{
    "type": "object",
    "minProperties": 2,
    "maxProperties": 3
}
```

>

5. 逻辑组合

> 关键字：**allOf, anyOf, oneOf, not**

从关键字名字可以看出其含义，满足所有、满足任意、满足一个。前三个关键字的使用形式是一致的，以allOf为例说明其形式。

```
{
    "allOf" : [
        Schema1,
        Schema2,
        ...
    ]
}
```

其中，"allOf"的内容是一个数组，数组内的成员都是内嵌的Json Schema。上例Schema1、Schema2都是内嵌的Json Schema。整个Schema表示当前Json数据，需要同时满足Schema1、Schema2,。

5.1 allOf

满足allOf数组中的所有Json Schema。

```
{
    "allOf" : [
        Schema1,
        Schema2,
        ...
    ]
}
```

需要注意，不论在内嵌的Schema里还是外部的Schema里，都不应该使"additionalProperties"为false。否则可能会生成任何数据都无法满足的矛盾Schema。

可以用来实现类似“继承”的关系，例如我们定义了一个Schema_base，如果想要对其进行进一步修饰，可以这样来实现。

```
{
    "allOf" : [
        Schema_base
    ]
    "properties" : {
        "other_pro1" : {"type" : "string"},
        "other_pro2" : {"type" : "string"}
    },
    "required" : ["other_pro1", "other_pro2"]
}
```

Json数据既需要满足Schema_base，又要具备属性"other_pro1"、"other_pro2"。

5.2 anyOf

满足anyOf数组中的任意个Schema。

```
{
    "anyOf" : [
        Schema1,
        Schema2,
        ...
    ]
}
```

Json数据需要满足Schema1、Schema2中的一个或多个。

5.3 oneOf

满足且进满足oneOf数组中的一个Schema，这也是与anyOf的区别。

```
{
    "oneOf" : [
        Schema1,
        Schema2,
        ...
    ]
}
```

5.4 not

这个关键字不严格规定Json数据应满足什么要求，它告诉Json不能满足not所对应的Schema。

```
{
    "not" : {"type" : "string"}
}
```

只要不是string类型的都Json数据都可以。

6. 复杂结构

对复杂结构的支持包括定义和引用。可以将相同的结构定义成一个“类型”，需要使用该“类型”时，可以通过其路径或id来引用。

6.1 定义

> 关键字：无

定义一个类型，并不需要特殊的关键字。通常的习惯是在root节点的definations下面，定义需要多次引用的schema。definations是一个json对象，key是想要定义的“类型”的名称，value是一个json schema。

```
{
    "definitions": {
        "address": {
            "type": "object",
            "properties": {
                "street_address": { "type": "string" },
                "city":           { "type": "string" },
                "state":          { "type": "string" }
            },
            "required": ["street_address", "city", "state"]
        }
    },
    "type": "object",
    "properties": {
        "billing_address": { "$ref": "#/definitions/address" },
        "shipping_address": { "$ref": "#/definitions/address" }
    }
}
```

上例中定义了一个address的schema，并且在两个地方引用了它，`#/definitions/address`表示从根节点开始的路径。

6.2 $id

> 关键字：$id

可以在上面的定义中加入id属性，这样可以通过id属性的值对该schema进行引用，而不需要完整的路径。

```
...
    "address": {
            "type": "object",
            "$id" : "address",
            "properties": {
                "street_address": { "type": "string" },
                "city":           { "type": "string" },
                "state":          { "type": "string" }
            },
            "required": ["street_address", "city", "state"]
        }
...
```

6.3 引用

> 关键字：**$ref**

关键字 `$ref` 可以用在任何需要使用json schema的地方。如上例中，billing_address的value应该是一个json schema，通过一个 `$ref` 替代了。

`$ref` 的value，是该schema的定义在json中的路径，以#开头代表根节点。

```
{
    "properties": {
        "billing_address": { "$ref": "#/definitions/address" },
        "shipping_address": { "$ref": "#/definitions/address" }
    }
}
```

如果schema定义了$id属性，也可以通过该属性的值进行引用。

```
{
    "properties": {
        "billing_address": { "$ref": "#address" },
        "shipping_address": { "$ref": "#address" }
    }
}
```

7. 通用关键字

通用关键字可以在任何json schema中出现，有些影响合法性校验，有些只是描述作用，不影响合法性校验。

7.1 enum

> 关键字：**enum**

可以在任何json schema中出现，其value是一个list，表示json数据的取值只能是list中的某个。

```
{
    "type": "string",
    "enum": ["red", "amber", "green"]
}
```

上例的schema规定数据只能是一个string，且只能是"red"、"amber"、"green"之一。

7. 通用关键字

通用关键字可以在任何json schema中出现，有些影响合法性校验，有些只是描述作用，不影响合法性校验。

7.1 enum

> 关键字：**enum**

可以在任何json schema中出现，其value是一个list，表示json数据的取值只能是list中的某个。

```
{
    "type": "string",
    "enum": ["red", "amber", "green"]
}
```

上例的schema规定数据只能是一个string，且只能是"red"、"amber"、"green"之一。

7.2 metadata

> 关键字：**title，description，default，example**

```
{
    "title" : "Match anything",
    "description" : "This is a schema that matches anything.",
    "default" : "Default value",
    "examples" : [
        "Anything",
        4035
    ]
}
```

只作为描述作用，不影响对数据的校验。