测试平台设计宗旨

不重复造轮子，组合现有的优秀工具，完成各自的需求

HttpRunner简介

HttpRunner 是一款面向 HTTP(S) 协议的通用测试框架，用户通过编写YAML或JSON文档即可实现自动化测试、性能测试、线上监控、持续集成等多种测试需求。

HttpRunner设计理念是不重复造轮子，底层基于Request和Pytest实现接口自动化。

HttpRunner的特征是

支持以YAML或JSON 格式定义测试用例

支持响应验证

支持初始化清除机制

支持套件级别的用例管理

支持Pytest命令（hrun底层封装的pytest）h3新特性

支持allure生成测试报告　　h3新特性

支持性能测试（底层Locust)

HttpRunner版本对比

目前使用率较广的版本是HR2.x,和HR3.x

HttpRunner3.x是HttpRunner2.x重构之后的一个版本。在实现思路和命令上都具有显著的差异。这里我们先简要的描述下差异，希望大家能够选择合适自己项目的版本来学习和使用。

	2.x	3.x
推荐格式	yml	.py
命令行	项目实现	复用pytest命令
报告	独立实现（Jinja2）	复用pytest报告生成
分层	api、case、suit	RunTestCase、RunRequest
特点	代码和case分离	链式调用，简化结构

解释：
2.x里支持yml和json　3.x增加了pytest格式，并且推荐使用
2.x的命令行主要是hrun系列　3.x的命令行复用了pytest，也可以直接用pytest xxx
2.x报告使用独立模板　3.x报告使用pytest-html，pytest-allure
2.x使用3个层级来区分请求、case、参数化　3.x主要分为请求和引用case
2.x的特点是，编写case可以完全脱离代码基础。但是需要学习hrun数据规则
3.x的特点是，极大的精简了项目规模。写case的时候有自动补全，降低了学习成本。但是组合case的时候相对抽象

目前为止HR最新的版本是3.14，我们的课程以H3为主。

HttpRunner3.x的运行原理和特点

HR3设计理念

约定优先于配置

投资回报率很重要

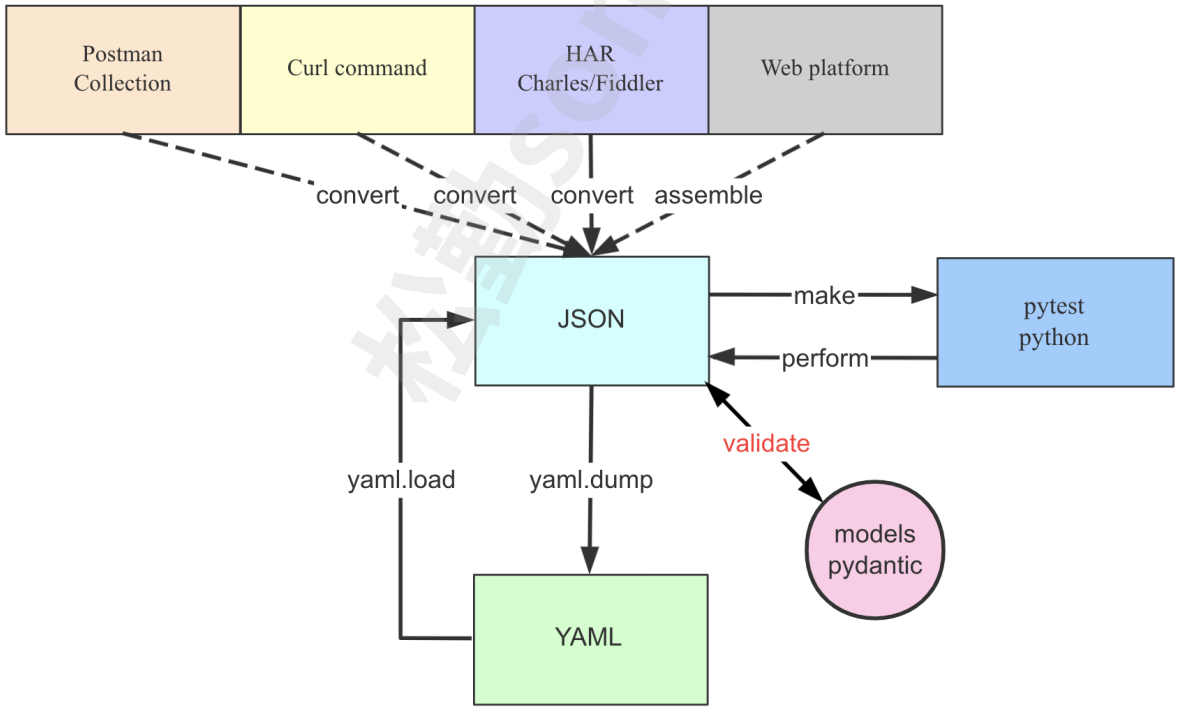
拥抱开源，底层利用requests，pytest，pydantic，allure和locust库

框架优点

继承 Requests 的全部特性，轻松实现 HTTP(S) 的各种测试需求
编写YAML或JSON格式的testcase，转译成pytestcase来运行测试
使用变量/提取/验证/钩子机制，创建case和复用case
重写pytest，可以利用pytest的各种插件
使用allure项目生成功能丰富的报告
使用jmespath，提取和验证json响应变得前所未有的简单。
基于 HAR 实现接口录制和用例生成功能（har2case）
结合 Locust 框架，无需额外的工作即可实现分布式性能测试

运行原理

如下图所示，实际上3.x会把json、yml格式的case转换成pytest用例，再使用重写的类来执行测试



HttpRunner3.x安装与使用

安装

HttpRunner存储于PyPI, 支持通过pip安装。

```
pip3 install httprunner
```

安装HttpRunner后，以下5个命令会写入系统环境变量配置。

- `httprunner`：主命令，用于所有功能。
- - `hrun`：指令`httprunner run`的别名，用于运行YAML/JSON/Pytest 测试用例。
  - `hmake`：指令`httprunner make`的别名，将YAML/JSON用例转换成pytest用例。
  - `har2case`：指令`httprunner har2case`的别名，将HAR文件转换成 YAML/JSON 用例。
  - `locust`：利用locust 运行性能测试。

查看`httprunner`版本：

```
$ httprunner -V  # hrun -V
3.1.4
```

查看可以使用的功能选项，运行:

```
$ httprunner -h
usage: httprunner [-h] [-V] {run,startproject,har2case,make} ...

一种HTTP(S)测试的解决方案。

选项说明:
  {run,startproject,har2case,make}
                        子命令说明
    run                 Make HttpRunner testcases and run with pytest.
    startproject        Create a new project with template structure.
    har2case            Convert HAR(HTTP Archive) to YAML/JSON testcases for
                        HttpRunner.
    make                Convert YAML/JSON testcases to pytest cases.

optional arguments:
  -h, --help            show this help message and exit
  -V, --version         show version
```

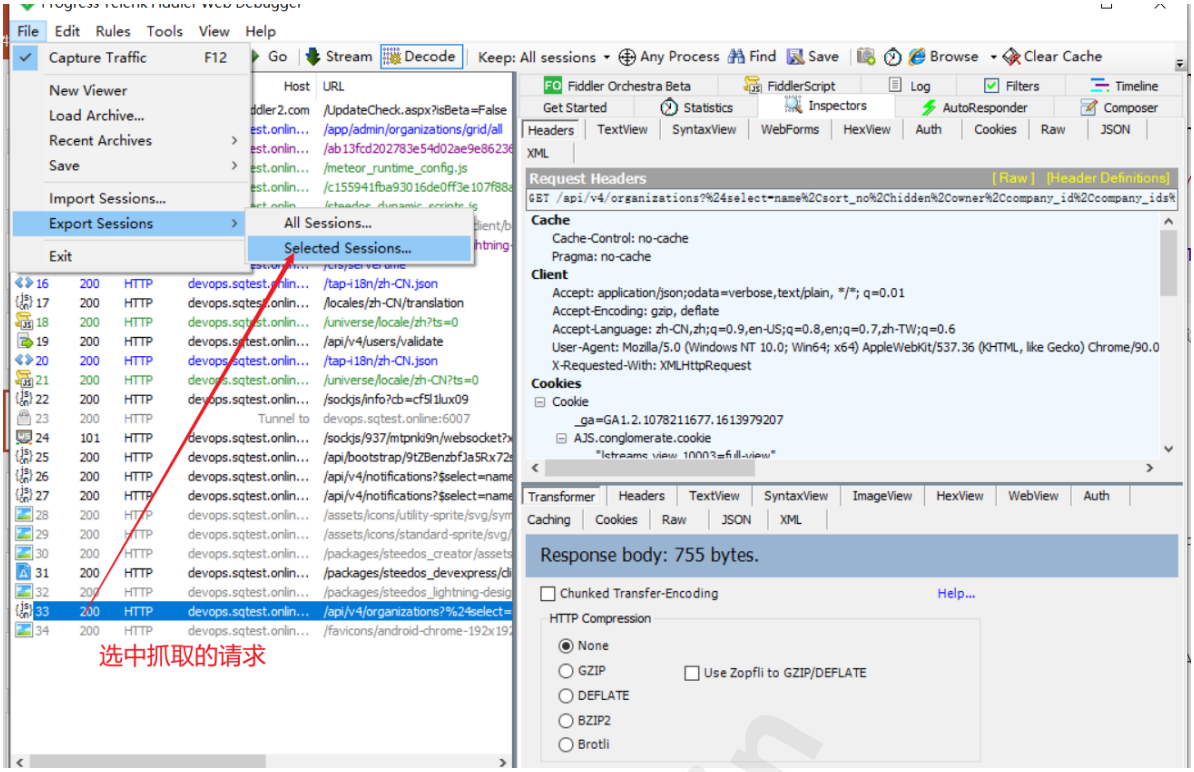
> 注意：`locust`是一个单独的命令，`locust`运行开始阶段时，monkey patch ssl避免递归错误

使用

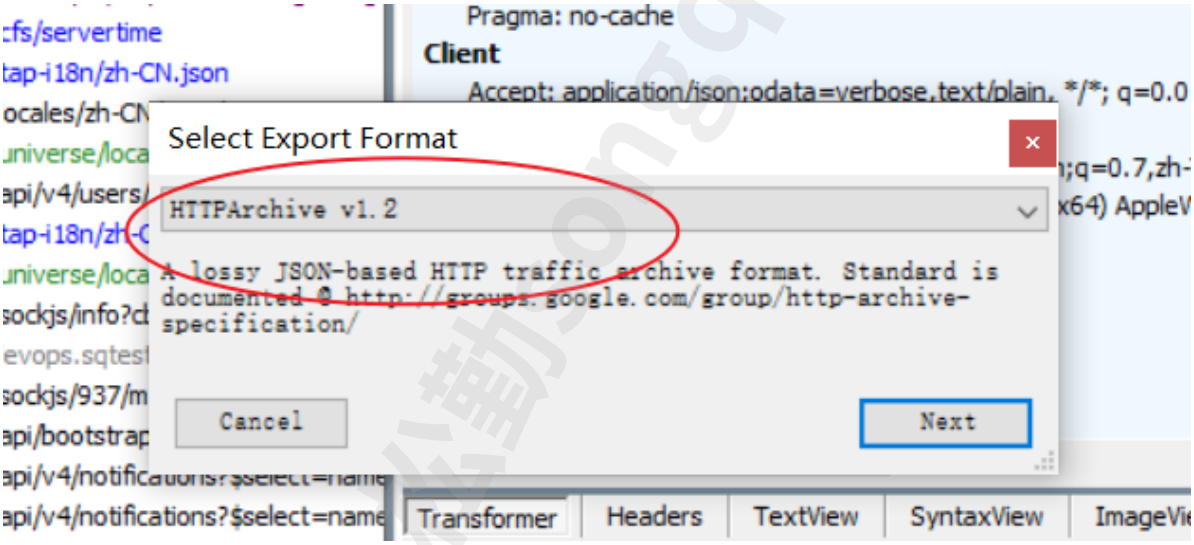
方式1.录制生成用例

http请求抓包

fiddler抓包，生成har文件，放到har目录中，File > Export Sessions > Selected Sessions > httparchive v1.2



选择该格式



生成pytest用例

HttpRunner `3.0.7`版本开始，`har2case`将HAR文件默认转换成pytest。

```
har2case  xxx.har
```

生成YAML文件

```
har2case -2y  xxx.har
```

生成Json文件

```
har2case -2j  xxx.har
```

方式2.编写测试用例

hr3支持 python代码，yaml，json 三种格式用例，以yml格式为例，只需要遵循一定规则，就可以写出符合hr3标准的用例

```
config:
  name: 登录测试用例
  parameters:
    username-password-code: ${get_login_data()}
  verify: false
  base_url: http://120.55.190.222:7080
  export:
     - cookie
teststeps:
-
  name: 用户名密码登录
  request:
    url: /api/mgr/loginReq
    data:
      username: $username
      password: $password
    method: POST
  extract:
    cookie: cookies.sessionid
  validate:
    -  eq:
         - status_code
         - 200
    -  eq:
         - body.retcode
         - $code
```

将该文件保存为case1.yml，进入该文件所在目录运行测试用例

```
hrun case1.yml
```


httprunner测试用例结构

HttpRunner `v3.x` 支持3种用例格式：`pytest`、`YAML`和`JSON`。

`pytest`、`YAML`和`JSON`格式的测试用例完全等价，包含的信息内容也完全相同。

- 对于熟悉 YAML 格式的人来说，编写维护 YAML 格式的测试用例会更简洁，但前提是要保证 YAML 格式没有语法错误。

在httprunner中，测试用例组织主要基于三个概念：

- 测试用例集(testsuite): 对应一个YAML/JSON/Python文件，包含单个或多个测试用例文件。
- 测试用例(testcase): 对应一个YAML/JSON/Python文件，包含单个或多个测试步骤。
- 测试步骤(teststep): 对应YAML/JSON/Python中 teststeps下的一个节点，描述单次接口测试的全部内容，包括发起接口请求、解析响应结果、检验结果等。

对于单个YAML文件来说，数据存储结构为`list of dict`的形式，其中可能包含一个全局配置项(config)和若干个测试步骤。

具体地： * config: 作为整个测试用例的全局配置项 * 测试步骤：对应单个测试步骤(teststep)，测试用例存在顺序关系，运行时将从前往后依次运行各个测试步骤。

testcase对应的YAML格式如下所示：

```
config:
    ...

teststeps:
-   # step 1
    ...

-   # step 2
    ...
```

实战案例：

访问教管系统：

http://120.55.190.222:7080/mgr/login/login.html

案例1：实现1个用例，完成登录操作

testcases\demo01.yml

```
config:
  name: 登录测试用例
  verify: false
  base_url: http://120.55.190.222:7080
  export:
     - cookie
teststeps:
-
  name: 用户名密码登录
  request:
    url: /api/mgr/loginReq
    data:
      username: auto
      password: sdfsdfsdf
    method: POST
  validate:
    -   eq:
          - status_code
          - 200
    -   eq:
          - body.retcode
          - 0
```

执行用例

```
hrun testcases\demo01.yml
```

案例2：继续编写该用例，在登录步骤的后面加上列出课程的功能

查询需要用到登录返回的cookies，因此需要从步骤1中返回cookies，这样步骤2可以引用

testcases\demo01.yml

```
config:
  name: 教管系统测试用例1
  verify: false
  base_url: http://120.55.190.222:7080
  export:  # 返回当前用例步骤中的变量
     - cookie
teststeps:
-
  name: 用户名密码登录
  request:
    url: /api/mgr/loginReq
    data:
      username: auto
      password: sdfsdfsdf
    method: POST
  extract:  #提取当前步骤变量
    cookie: cookies.sessionid
  validate:
    -  eq:
         - status_code
         - 200
    -  eq:
         - body.retcode
         - 0
-
  name: 列出课程
  request:
    cookies:
      sessionid: $cookie
    url: /api/mgr/sq_mgr/
    params:
      action: list_course
      pagenum: 1
      pagesize: 20
    method: GET
  extract:
    id: body.retlist[0].id
  validate:
      - eq:
          - status_code
          - 200
      - eq:
          - body.retcode
          - 0
```

案例3：测试步骤引用其他用例

如：实现登录后列出课程

testcases\demo01.yml

```
config:
  name: 列出课程
  verify: false
  base_url: http://120.55.190.222:7080
  export:
    - id
teststeps:
-
  name: 先登录
  variables:
    username: auto
    password: sdfsdfsdf
    code: 0
  testcase: testcases/login.yml
-
  name: 列出课程
  request:
    cookies:
      sessionid: $cookie
    url: /api/mgr/sq_mgr/
    params:
      action: list_course
      pagenum: 1
      pagesize: 20
    method: GET
  extract:
    id: body.retlist[0].id
  validate:
      - eq:
          - status_code
          - 200
      - eq:
          - body.retcode
          - 0
```

被引用的用例:

testcases\login.yml

```
config:
  name: 登录测试用例
  parameters:
    username-password-code: ${get_login_data()}
  verify: false
  base_url: http://120.55.190.222:7080
  export:
     - cookie
teststeps:
-
  name: 用户名密码登录
  request:
    url: /api/mgr/loginReq
    data:
      username: $username
      password: $password
    method: POST
  extract:
```

```
    cookie: cookies.sessionid
  validate:
    -  eq:
         - status_code
         - 200
    -  eq:
         - body.retcode
         - $code
```

课程增删改查操作。。。

测试用例集（测试套件）

测试用例集（testsuite）的格式如下所示：

```yaml
config:
  name: test suite demo
  variables:  # testsuite config variables
    foo1: config_bar1
    foo2: config_bar2
    expect_foo1: config_bar1
    expect_foo2: config_bar2
  base_url: "https://postman-echo.com"

testcases:
-
  name: test case 1
  variables:  # testcase variables
    uid: 1000
    var_c: ${gen_random_string(5)}
  parameters:
    uid: [101, 102, 103]
    var_c: ["demo1", "demo2", "demo3"]
  testcase: /path/to/testcase1
```


附录：变量空间(context)作用域

在测试用例内部，HttpRunner划分了两层空间作用域(context).

- config: 作为整个测试用例的全局配置项，作用域为整个测试用例，包含base_url, verify, variables, export。
- teststeps: 测试步骤的列表，每个步骤都对应一个API请求或另一个用例的引用，另外variables/extract/validate/hooks支持创建及其复杂的测试用例。
- 测试步骤的变量空间会继承或覆盖config中定义的内容
  - 若某变量在 config 中定义了，在某 test 中没有定义，则该 test 会继承该变量
  - 若某变量在 config 和某 test 中都定义了，则该 test 中使用自己定义的变量值
- 各个测试步骤的变量空间相互独立，互不影响；
- 如需在多个测试步骤中传递参数值，则需要使用 extract 关键字，并且只能从前往后传递

config

每个测试用例都必须有config部分，可以配置用例。

```
config:
    name: xxx
    variables:              # 配置变量(config variables)
        varA: "configA"
        varB: "configB"
        varC: "configC"
    parameters:             # 参数变量(parameter variables)
        varA: ["paramA1"]
        varB: ["paramB1"]
    base_url: "https://postman-echo.com"
    verify: False
    export: ["foo3"]

teststeps:
-
    name: step 1
    ...
-
    name: step 2
    ...
```

name(必须)

测试用例的名称，将在log和报告中展示。

base_url(可选)

测试用例中的通用Host，例如https://postman-echo.com。如果base_url被指定，测试步骤中的url只能写相对路径。当你要在不同环境下测试时，这个配置非常有用。

variables(可选的)

定义的全局变量，作用域为整个用例。每个测试步骤都可以引用config variables。也就是说，step variables 优先级高于 config variables.

parameters(可选的)

全局参数，用于实现数据化驱动，作用域为整个用例。

verify(可选的)

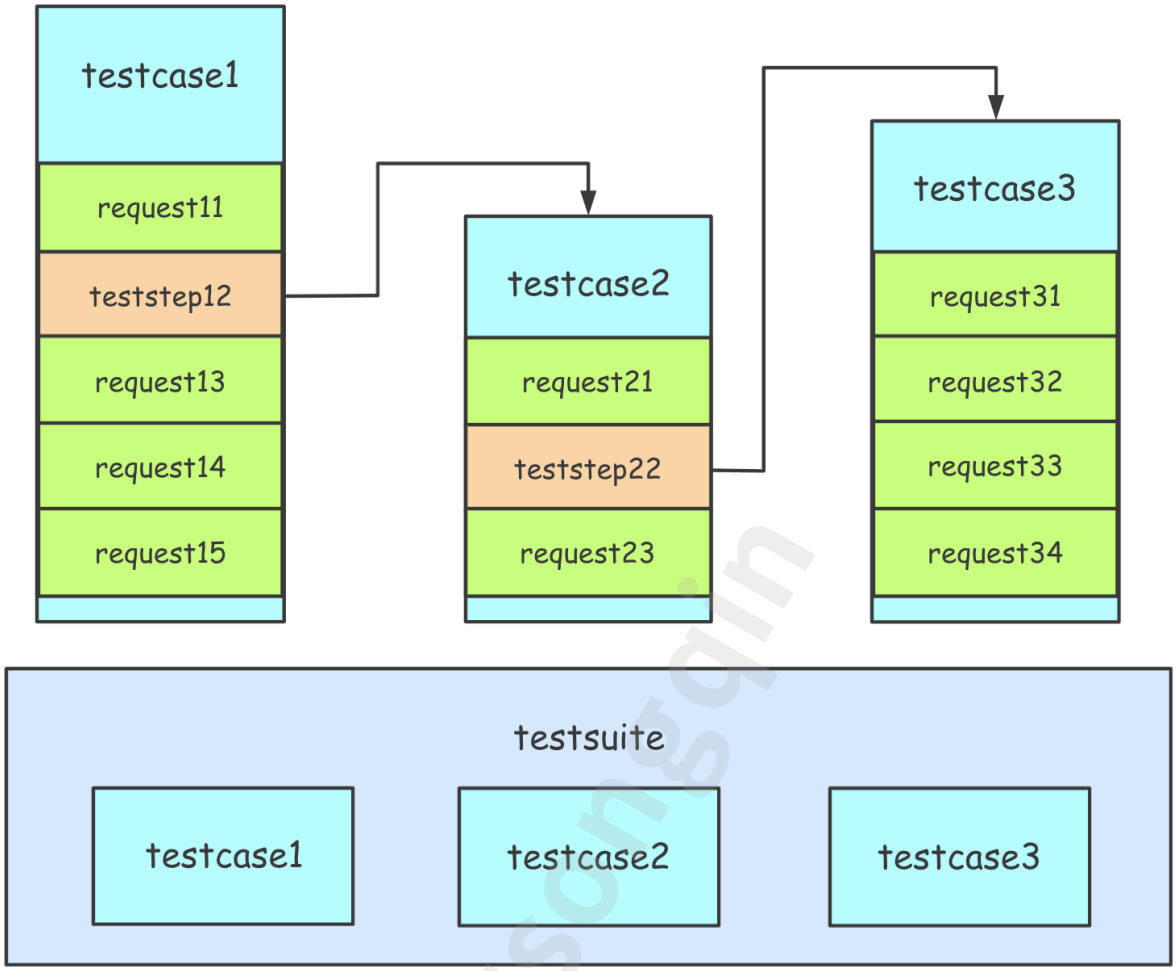
指定是否验证服务器的TLS证书。如果我们想记录测试用例执行的HTTP流量，这将特别有用，因为如果没有设置verify或将其设置为True，则会发生SSLError。

export(可选的)

指定输出的测试用例变量。将每个测试用例看作一个黑盒，config variables是输入变量，config export是输出变量。当一个测试用例在另一个测试用例的步骤中被引用时，config export将被提取并在随后的测试步骤中使用。

teststeps

每个测试用例都有1个或多个测试步骤（List[step]），每个测试步骤对应一个API请求或其他用例的引用。




```
teststeps:
-
    name: get with params
    variables:
        foo1: bar11
        foo2: bar21
        sum_v: "${sum_two(1, 2)}"
    request:
        method: GET
        url: /get
        params:
            foo1: $foo1
            foo2: $foo2
            sum_v: $sum_v
        headers:
            User-Agent: HttpRunner/${get_httprunner_version()}
    extract:
        foo3: "body.args.foo2"
    validate:
        - eq: ["status_code", 200]
        - eq: ["body.args.foo1", "bar11"]
        - eq: ["body.args.sum_v", "3"]
```

```
        - eq: ["body.args.foo2", "bar21"]

-
    name: post form data
    variables:
        foo2: bar23
    request:
        method: POST
        url: /post
        headers:
            User-Agent: HttpRunner/${get_httprunner_version()}
            Content-Type: "application/x-www-form-urlencoded"
        data: "foo1=$foo1&foo2=$foo2&foo3=$foo3"
    validate:
        - equal: ["status_code", 200, "失败原因"]
        - equal: ["body.form.foo1", "$expect_foo1"]
        - equal: ["body.form.foo2", "bar23"]
        - equal: ["body.form.foo3", "bar21"]
```

name(必须)

name用来定义测试步骤 name，将出现在log和测试报告中。

variables(可选的)

测试步骤中定义的变量，作用域为当前测试步骤。

如果想在多个测试步骤中共享变量，需要在config variables中定义。

测试步骤中的变量，会覆盖config variables中的同名变量。

request(必须)

METHOD(必须)

设置http方法，支持RestFul中的所有http方法（GET/POST/PUT/PATCH/DELETE/），相当于requests.request 中的method。

URL(必须)

设置Url,如果base_url在config中设置了，url只能是相对路径部分。相当于requests.request 中的url。

PARAMS(可选)

设置Url的query，相当于requests.request 中的params。

HEADERS(可选)

设置请求的headers，相当于requests.request 中的headers。

COOKIES(可选)

设置Http请求的cookies，相当于requests.request 中的cookies。

DATA(可选)

设置http请求的Body，相当于requests.request 中的data。

JSON(可选)

设置http请求json格式的body，相当于requests.request 中的json。

extract(可选)

从当前 HTTP 请求的响应结果中提取参数，并保存到参数变量中（例如token），后续测试用例可通过$token的形式进行引用。

原理：利用jmespath 提取Json response body的内容。

validate(可选)

测试用例中定义的结果校验项，作用域为当前测试用例，用于实现对当前测试用例运行结果的校验。

原理：用jmespath 提取Json response的内容，并进行断言校验。

格式：

```
- 运算符: [jmespath表达式, expected_value, message]
```

- 运算符包括:
- equal: 等于
- contained_by: 实际结果是否被包含在预期结果中
- contains: 预期结果是否被包含在实际结果中
- endswith: 以...结尾
- greater_or_equals: 大于等于
- greater_than: 大于
- length_equal: 长度等于
- length_greater_or_equals: 长度大于等于
- length_greater_than: 长度大于
- length_less_or_equals: 长度小于等于
- length_less_than: 长度小于
- less_or_equals: 小于等于
- less_than: 小于
- not_equal: 不等于
- regex_match: 字符串是否符合正则表达式匹配规则
- startswith: 以...开头
- string_equals: 字符串相等
- type_match: 类型是否匹配
- jmespath: jmespath表达式，详见JMESPath Tutorial
- expected_value: 指定期望值或变量，也可以调用方法
- message(optional): 用于描述断言失败原因

hooks(可选)

- setup_hooks函数放置于debugtalk.py中，并且必须包含三个参数:
- method: 请求方法，比如：GET,POST,PUT
- url: 请求URL
- kwargs: request的参数字典
- teardown_hooks函数放置于debugtalk.py中，并且必须包含一个参数:
- resp_obj: requests.Response实例