# Tavern框架

## 一、框架格式

test\_name: Get some fake data from the JSON placeholder API

stages:

- name: Make sure we have the right ID

request:

url: https://jsonplaceholder.typicode.com/posts/1

method: GET

response:

status\_code: 200

json:

id: 1

save:

json:

returned\_id: id

**test\_name**：测试用例名称

测试tavern\_test\_x.yaml，tavern\_test\_y.yaml文件

使用tavern-file-path-regex在配置文件或命令行选项

如，py.test --tavern-file-path-regex "tavern\_test\_.\*.yaml"

**stages**：阶段（一个或多个）

name：阶段名称

**request**：请求

url： 请求url

method：GET、POST、PUT、DELETE、PATCH、OPTIONS 或 HEAD 之一。

如果未定义，则默认为 GET。

json： 键：值对/列表的映射，将转换为 JSON 并作为请求正文发送。

params：查询参数的键值对的映射。

data： 作为参数进入正文的键值对的映射，或自行发送的字符串（传参）

headers：网页头部。默认为添加content-type: application/json

**response**：回复

status\_code：状态码。如果没定义，默认为200

json：如果响应的是json，根据给定的值检查主体，期望是键值对的形式

headers：将根据报头检查的值对

redirect\_query\_params：检查在报文头部中传递的重定向url的查询参数

save: 保存。可以从正文、标题或重定向查询参数中保存内容

## 二、变量格式

**请求变量 request\_vars**

**环境变量 env\_vars**

email=tong@tong.com

password=Aa123456

json:

email: "{tavern.env\_vars.email}"

password: "{tavern.env\_vars.password}"

## 三、调用外部函数

$ PYTHONPATH=$PYTHONPATH:

verify\_response\_with响应块

verify\_response\_with:

function: testing\_utils:函数名

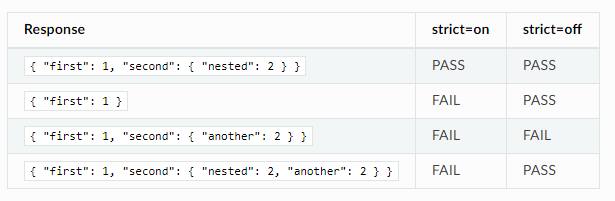
**将外部数据注入请求**

外部函数必须返回一个字典，其中每个键指向单个值或使用点符号可访问的对象。最简单的方法是返回一个Box对象。

**保存响应的数据**

$ext在save块中使用时有特殊作用：返回对象中的每个键都将被保存

## 四、Strict 关键字检查



strict关键字打开后 希望**完全匹配**

关闭 'strict' 还意味着只要存在测试响应中指定的项目，列表中的额外项目就会被忽略，如果来自服务器的响应是严格开启的

可以通过 3 种不同方式控制此设置，**优先级**顺序为：

1.在测试/阶段本身

2.在命令行上传递

3.从 pytest 配置中读取

**打开/关闭**

json:off headers:on- 关闭正文，但打开标题。

redirect\_query\_params将保持默认关闭。

json:off headers:off - 关闭正文和标题严格检查

redirect\_query\_params:on json:on 重定向参数打开并且 json 保持打开状态（默认情况下它是打开的），头部严格匹配保持关闭状态（默认情况下）。

**命令行：**

--tavern-strict

py.test --tavern-strict json:on headers:on redirect\_query\_params:off -- my\_test\_folder/

**在 Pytest 配置文件中：**

[pytest]

tavern-strict=json:off headers:on

**在测试用例中：**

strict:

- headers:on

- json:off

## 五、重用请求和 YAML 片段

定义**锚点** &name\_of\_anchor

first: &top\_anchor

a: b

c: d

second: \*top\_anchor

third:

<<: \*top\_anchor

c: overwritten

e: f

## 六、包括外部文件

使用**!include**指令在每个测试中包含一个文件

includes:

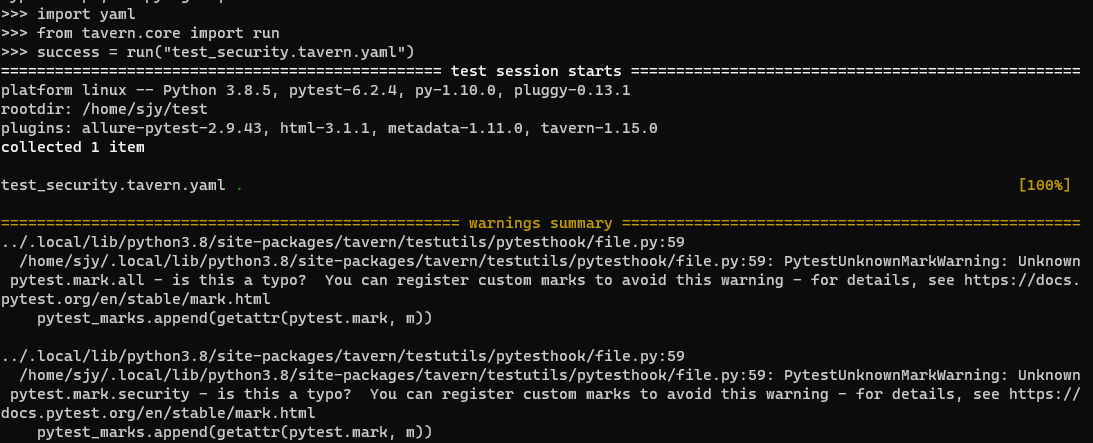
- !include includes.yaml

## 七、使用 run() 函数

import yaml

from tavern.core import run

success = run("test\_security.tavern.yaml")



## 八、匹配响应中的任意返回值

只想确保返回一个值，但不知道它是什么。这可以通过使用!anything作为响应块中匹配的值来实现

response:

json:

returned\_block: !anything

## 九、类型转换

整数→字符串

request:

json:

an\_integer: !!str 1234567890

处理格式化的值

request:

json:

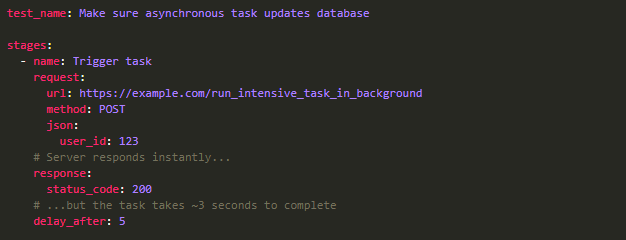
# an\_integer: !!int "{my\_integer:d}" # Error

an\_integer: !int "{my\_integer:d}" # Works

不会执行字符串格式化的标签：!raw

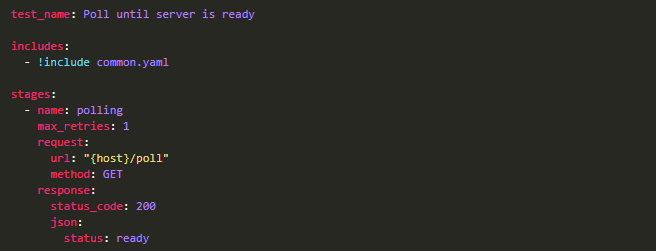
## 十、在测试之间添加延迟

可以使用**delay\_before**和**delay\_after**键



## 十一、重试测试

重试某个阶段一定次数**max\_retries**



## 十二、标记测试

标记可用于：

1.从命令行选择要运行的标记测试的子集

2.根据条件跳过某些测试

3.将测试标记为暂时预期失败，以便稍后修复

test\_name: Get server info from slow endpoint

**marks:**

**- slow**

stages:

test\_name: Get server info from fast endpoint

**marks:**

**- fast**

stages:

$ py.test -m "not slow"

$ py.test -m "fast"

**特殊标记：**

**跳过**skip**：**

marks:

- skip

stages:

skip: True

**失败**xfail**：**

marks:

- xfail

**参数化**parametrize**：**

marks:

- parametrize:

key: metadata

vals:

**使用固定装置**usefixtures：

marks:

- usefixtures:

# Allure测试报告

