

三日精通：自动化必备之Pytest测试框架训练营

第一天：pytest测试框架快速上手

主讲：北凡老师

1. 测试框架

测试框架：**抽象**出来的**工具集合**，提供大量组件、工具、功能

- 用例发现
- 用例管理
- 环境管理
- 用例执行
- 测试报告

大部分变成语言，都有测试框架：

- java: junit, testng
- **python: unittest, pytest**

	unittest	pyetst
安装、卸载	无需安装	手动安装
升级、降级	无法改变版本	可以指定版本
代码风格	Java语言	Python语言
插件生态	只有几个插件	1400+插件涵盖各方面
备注	由python官方维护	完全兼容unittest

2. 快速上手

安装很简单

```
pip install pytest # 安装
pip install pytest -U # 升级到最新版
```

pytest有三种启动方式：

1. 命令： `pytest`
2. 代码：

```
import pytest

pytest.main()
```

3. 鼠标【不推荐】

1. 是由pycharm提供的，不是pytest
2. 行为前两种方式，不一致，不适合负债项目

pytest在简单的基础上，对断言进行高级封装（AST），对python数据结构断言，非常友好

1. pytest遵循了python简单的学习方式
2. pytest实现了很多高级特性
3. 鼓励（课程要求）大家积极使用断言

3. 看懂结果

```
===== test session starts =====
platform win32 -- Python 3.12.0, pytest-8.3.3, pluggy-1.5.0
rootdir: E:\PyProject\my_pytest
plugins: result-log-1.2.2
collected 4 items

test_data.py FFFF [100%]

===== FAILURES =====
```

```
test_number

def test_number():
>     assert 1 == 2
E     assert 1 == 2

test_data.py:3: AssertionError
test_data.py:24: AssertionError
===== short test summary info =====
FAILED test_data.py::test_number - assert 1 == 2
===== 4 failed in 0.07s =====
```

- 1. 执行环境：版本、根目录、用例数量
- 2. 执行过程：文件名称、**用例结果**、执行进度
- 3. 失败详情：用例内容、断言提示
- 4. 整体摘要：结果情况、结果数量、花费时间

用例结果缩写

缩写	单词	含义
.	passed	通过
F	failed	失败（用例执行时报错）
E	error	出错（fixture执行报错）
s	skipped	跳过
X	xpassed	预期外的通过（不符合预期）
x	xfailed	预期内的失败（符合预期）

4. 用例规则

1. 用例发现规则

测试框架在识别、加载用例的过程，称之为：**用例发现**

pytest的用例发现步骤：

- 1. 遍历所有的目录，例外：`venv`，`.`开头的目录
- 2. 打开python文件，`test_`开头 或者 `_test` 结尾
- 3. 遍历所有的 `Test` 开头类

4. 收集所有的 `test_` 开头的函数 或者 方法

2. 用例内容规则

pytest 8.4 增加了一个强制要求

pytest对用例的要求：

1. 可调用的（函数、方法、类、对象）
2. 名字 `test_` 开头
3. 没有参数（参数有另外含义）
4. 没有返回值（默认为None）

3. 练习

有函数 `add` 接收两个参数，并返回它们相加的结果

请为此编写测试用例

```
def add(a, b):  
    return a+b  
  
class TestAdd:  
  
    def test_int(self):  
        res = add(1,3)  
        assert res == 4  
  
    def test_str(self):  
        res = add("1","3")  
        assert res == "13"  
  
    def test_list(self):  
        res = add([1],[2,3,4])  
        assert res == [1,2,3,4]
```

在Python中，类是函数的进一步抽象，由需求驱动而来

5. 配置框架

配置 可以改变pytest 默认的规则：

1. 命令参数
2. ini配置文件

所有的配置方式，可以一键获取

```
pytest -h
```

- 有哪些配置
- 分别是什么方式
 - `-` 开头：参数
 - 小写字母开头：ini配置
 - 大写字母开头：环境遍历
- 配置文件: `pytest.ini`

常用参数：

- `-v` ： 增加详细程度
- `-s`： 在用例中正常的使用**输入输出**
- `-x`： 快速退出，当遇到失败的用例停止执行
- `-m`： 用例筛选

6. 标记mark

标记 可以让用例与众不同，进而可以让用被区别对待

1. 用户自定义标记

用户自定义标记 只能实现用例筛选

步骤：

1. 先注册
2. 再标记
3. 后筛选

```
class TestAdd:

    @pytest.mark.api
    def test_int(self):
        res = add(1,3)
        assert res == 4

    @pytest.mark.ui
    def test_str(self):
        res = add("1", "3")
        assert res == "13"

    @pytest.mark.pay
    def test_list(self):
        res = add([1], [2,3,4])
        assert res == [1,2,3,4]

#
```

```
pytest -m web
```

2. 框架内置标记

框架内置标记 为用例增加特殊执行效果

和用户自定义标记区别：

1. 不需注册，可以直接使用
2. 不仅可以筛选，还可以增加特殊效果
3. 不同的标记，增加不同的特殊效果
 - **skip**：无条件跳过
 - **skipif**：有条件跳过
 - **xfail**：预期失败
 - **parametrize**：参数化
 - **usefixtures**：使用fixtures

数据驱动测试 = 参数化测试 + 数据文件

根据数据文件的内容，动态决定用例的数量、内容

7. 数据驱动测试参数

数据文件，驱动用例执行数量、内容

```
a,b,c  
1,1,2  
2,3,5  
3,3,6  
4,4,7
```

```
@pytest.mark.ddt  
@pytest.mark.parametrize(  
    "a,b,c",  
    read_csv("data.csv")  
)  
def test_ddt(self,a,b,c):  
    res = add(int(a),int(b))  
    assert res == int(c)
```

三日精通：自动化必备之Pytest测试框架训练营

第二天：pytest测试框架核心技巧

主讲：北凡老师

上节回顾

8. 夹具fixture

夹具：在用例**执行之前**、**执行之后**，自动运行代码

场景：

- 之前：加密参数 / 之后：解密结果
- 之前：启动浏览器 / 之后：关闭浏览器
- 之前：注册、登录账号 / 之后：删除账号

1. 创建fixture

```
@pytest.fixture
def f():

    # 前置操作
    yield
    # 后置操作
```

1. 创建函数
2. 添加装饰器
3. 添加yield关键字

2. 使用fixture

1. 在用例的参数列表中，加入 fixture名字即可
2. 给用例加上 `usefixtures` 标记

```
def test_1(f):  
    pass  
  
@pytest.mark.usefixtures("f")  
def test_2():  
    pass
```

3. 高级用法

1. 自动使用
2. 依赖使用
 1. linux: 使用linux进行编译
 2. git: 使用git进行版本控制
 3. fixture: 使用fixture进行前后置自动操作
3. 返回内容: 接口自动化封装: 接口关联
4. 范围共享:
 - 默认范围: function
 - 全局范围: session
 - 使用 `conftest.py`

命名空间 -> 第三空间

9. 插件管理

pytest插件生态是pytest特别的优势之处。

插件分成两类:

- 不需要安装: 内置插件

- 需要安装：第三方插件

插件的启用管理：

- 启用：`-p abc`
- 禁用：`-p no:abc`

插件使用方式：

1. 参数
2. 配置文件
3. fixture
4. mark

10. 常用第三方插件

pytest有1400+ 插件： https://docs.pytest.org/en/stable/reference/plugin_list.html

1. pytest-html

用途：生成HTML测试报告

安装：

```
pip install pytest-html
```

使用：

```
--html=report.html --self-contained-html
```

report.html

Report generated on 14-Nov-2024 at 21:08:13 by [pytest-html](#) v4.1.1

Environment


Python	3.12.0
Platform	Windows-11-10.0.22631-SP0
Packages	<ul style="list-style-type: none">• pytest: 8.3.3• pluggy: 1.5.0
Plugins	<ul style="list-style-type: none">• html: 4.1.1• metadata: 3.1.1• order: 1.3.0• rerunfailures: 14.0• result-log: 1.2.2• xdist: 3.6.1

Summary

3 tests took 5 ms.

(Un)check the boxes to filter the results.

☐ 0 Failed, ☒ 3 Passed, ☐ 0 Skipped, ☐ 0 Expected failures, ☐ 0 Unexpected passes, ☒ 0

Result 	Test
Passed	tests/test_baili.py::test_3
Passed	tests/test_beifan.py::test_1
Passed	tests/test_beifan.py::test_2

2. pytest-xdist

用途：分布式执行

安装：

```
pip install pytest-xdist
```

使用：

```
-n N
```

- 只有在任务本身耗时较长，超出调用成本很多的时候，才有意义
- 分布式执行，有并发问题：资源竞争、乱序

3. pytest-rerunfailures

用途：用例失败之后，重新执行

安装：

```
pip install pytest-rerunfailures
```

使用：

```
--reruns 5 --reruns-delay 1
```

4. pytest-result-log

用途：把用例的执行结果记录到日志文件中

安装：

```
pip install pytest-result-log
```

使用：

```
log_file = ./logs/pytest.log
log_file_level = info
log_file_format = %(levelname)-8s %(asctime)s [%(name)s:%(lineno)s] : %(message)s
log_file_date_format = %Y-%m-%d %H:%M:%S

; 记录用例执行结果
result_log_enable = 1
; 记录用例分割线
result_log_separator = 1
; 分割线等级
result_log_level_separator = warning
; 异常信息等级
result_log_level_verbose = info
```

11. 企业级测试报告

allure 是一个测试报告框架

安装

```
pip install allure-pytest
```

配置

```
--alluredir=temps --clean-alluredir
```

生成报告

```
allure generate -o report -c temps
```

allure支持对用例进行**分组和关联**（敏捷开发术语）

@allure.epic	史诗	项目
@allure.feature	主题	模块
@allure.story	故事	功能
@allure.title	标题	用例

使用相同装饰器的用例，自动并入一组

```
@allure.epic('码尚教育自动化测试')
@allure.feature('pytest框架训练营')
@allure.story('mark标记和筛选')
@allure.title("实现筛选的用例")
@pytest.mark.ut
def test_a():
    pass

@allure.epic('码尚教育自动化测试')
@allure.feature('pytest框架训练营')
@allure.story('fixture前置和后置')
@allure.title("使用fixture的用例")
@pytest.mark.ut
def test_b():
    pass
```

12. Web自动化测试实战

pytest仅进行用例管理，**不会控制浏览器**，需要借助新的工具：**selenium**

1. 只了解selenium

2. 搜索关于selenium的pytest插件

```
@pytest.fixture()
def selenium():
    d = webdriver.Chrome()
    yield d

    d.quit()
```

三日精通：自动化必备之Pytest测试框架训练营

第三天：接口自动化框架封装实战

主讲：北凡老师

13. 测试框架要封装什么

封装：

- 隐藏细节
- 增加功能
- 优化功能

接口自动化封装：

- 使用yaml作为用例，降低自动化门槛
- 自动请求接口、断言接口
- 自动在日志记录HTTP报文
- 自动生成allure测试报告

14. YAML文件格式

一句话：YAML完全兼容JSON格式，并且支持Python相似写法

重点：

1. YAML完全兼容JSON
2. 是数据格式，不是变成语言
3. 像Python一样容易编辑和阅读

1. 安装yaml模块

```
pip install pyyaml
```

2. 编写yaml文件

1. `#` 作为注释符号
2. 缩进：使用2个空格
3. 成员表示
 - `-` 表示列表成员
 - `:` 表示字典成员
4. 兜底：完全兼容JSON

```
# 这是我的第一个yaml文件
```

```
数字：
```

```
- 1  
- -1  
- 1.1
```

```
字符串：
```

```
- '1231231231'  
- "ajklasdjas"  
- asdasda
```

```
空值：null # JSON写法
```

```
列表：[ 1, 2, 3, ] # JSON写法
```

```
字典：{ "a": 1, "b": 2 } # JSON写法
```

3. 加载yaml文件


```
import yaml

def load_yaml(path):
    f = open(path, encoding="utf-8") # 打开文件
    s = f.read() # 读取文件内容

    data = yaml.safe_load(s)

    return data
```

15. 接口测试用例

1. 设计用例内容

1. 名字：请求首页数据接口
2. 标记【可选】
3. 步骤
 1. 请求接口：GET https://www.baidu.com
 2. 响应断言：status_code == 200
 3. 提取变量：json()['code']

2. YAML表示用例

```
name: 登陆成功用例
steps:
  - request: # 发送请求
    method: POST
    url: http://116.62.63.211/shop/api.php?
application=app&application_client_type=weixin
    params:
      s: user/login
    json: {
      "accounts": "beifan_1105",
      "pwd": "beifan_1105",
      "type": "username"
    }

  - response: # 断言响应
    status_code: 200
```

```
    json:
      code: 0
      msg: 登录成功
      data:
        username: beifan_1105

- extract: # 提取变量
  token: [ json, $.data.token]
```

16. 封装接口自动化框架

1. 请求接口

外部工具: requests

从HTTP协议抓包角度, 请求由三部分组成:

- 行: 方法+地址 (必填)
- 头: 请求头 (键值对)
- 体: 参数内容

```
import requests

### 1. 方法
url = 'http://116.62.63.211/shop/api.php'

# GET方法
requests.get(url)
# POST方法
requests.post(url)

# 任意方法
requests.request('MOVE', url)

### 2. 头
method = 'post'
url = 'http://116.62.63.211/shop/api.php'

requests.request(
    method, url,
    headers={ # 字符串字典
        "1": '1',
        "2": "b",
    }
)
```

```

)

### 3. 参数
method = 'post'
url = 'http://116.62.63.211/shop/api.php'

requests.request(
    method, url,
    json={ # 任意内容字典
        "a": 1,
        "b": [1,2,3],
        "c": {},
    }
)

```

2. 断言响应

1. 响应里有什么
2. 响应如何断言

从HTTP协议抓包角度，响应由三部分组成：

- 行：状态码
- 头：响应头（键值对）
- 体：响应内容

```

# resp 就是响应
# 获取响应中的内容
print(resp.status_code) # 状态码
print(resp.headers) # 响应头
print(resp.text) # 响应正文
print(resp.json()) # 响应正文转成JSON

# 断言单个内容是否正确
assert resp.status_code == 200
assert '美酒' in resp.text
assert resp.json()['data']['banner_list'][0]['name'] == "美酒"

# 断言全部的内容
from responses_validator import validator

validator(
    resp,

```

```
status_code=200,
text='*美酒*',
json={
    "data":{
        "banner_list":[{"name":"美酒"}]
    }
}
)
```

3. 变量提取

基本原则：

- JSON: JSONPATH
- HTML: XPATH
- 字符串: RE

RE 可以兜底

```
from extract_utils import extract

var = extract(resp, 'json', '$..banner_list[0].name')

print(var)
```

4. 框架落地封装

```
runner_utils.py x  pytest.log x
my_pytest E:\PyProject\my_pytest
> .venv library root
> commons
> data
> logs
> report
> temps
> tests
> conftest.py
> main.py
> pytest.ini
External Libraries
Scratches and Consoles

INFO 2024-11-15 21:37:09 [pytest_result_log:122] : -----Start:
INFO 2024-11-15 21:37:09 [beifan:14] : 1. 正在发送请求...
INFO 2024-11-15 21:37:09 [beifan:15] : {'method': 'POST', 'url': 'http://116.6:
INFO 2024-11-15 21:37:09 [beifan:18] : 2. 正在断言响应...
INFO 2024-11-15 21:37:09 [beifan:19] : {'status_code': 200, 'json': {'code': 0,
INFO 2024-11-15 21:37:09 [responses_validator:20] : resp=<Response [200]>, rais
INFO 2024-11-15 21:37:09 [responses_validator:20] : is_valid = True, with []
INFO 2024-11-15 21:37:09 [beifan:22] : 3. 正在提取变量...
INFO 2024-11-15 21:37:09 [beifan:25] : token = 0be55e73bfb0e81a2948493b67c
INFO 2024-11-15 21:37:09 [pytest_result_log:190] : test status is PASSED (test
INFO 2024-11-15 21:37:09 [pytest_result_log:128] : -----End:
```

进一步完善：

1. YAML用例测试文件上传？
2. YAML用例进行数据去掉测试？
3. YAML用例进行自定义的断言
4. YAML用例进行数据库查询？

两个方向：

1. 自动化工程师
2. 测试开发工程师

直播：自动化测试 4个月左右

快速就业跳槽：15-30天

学习：

- 找班主任
- 找技术北凡

工作：

- 技术难题
- 跳槽