全程66期接口自动化第二次

回顾：

**0 剧情+目标 无人值守**

**1 requests 库提供API（Application programming interface 接口 程序requests和用户对接）表**

**2 例子 requests 能发上下游接口**

### 内容1: python 语法加强 （降低代码风险）

**需求：**

****

**解决方式：**

**方式 1 [] 根据键获取值=====》getXX(键，默认值) 如果获取不到值 不会报错 而是用默认值代替**

**方式2 try 也能起到效果**

|  |
| --- |
| **#1 接口文档要素**  **import requests**  **# 运维服务器 + 接口地址**  **login\_url="http://localhost:6088/api/User2222"**  **login\_headers={'Content-Type':'application/json'}**  **login\_method='post'**  **login\_body={**  **"username": "admin" ,**  **"password": "123"}**  **# = 左边API表**  **# = 右边实际参数 来自于接口文档**  **shiji=requests.post(url=login\_url,headers=login\_headers,json=login\_body)**  **print(shiji.status\_code)**  **print(shiji.headers)**  **# 下游代码---》一个值 响应头～**  **#shouquanma=shiji.headers['Authorization'] # 存放**  **shouquanma=shiji.headers.get('Authorization','没有授权码') #默认值降低代码风险～**  **print(shouquanma) #引用**  **body=None**  **try:**  **# 提取json响应体**  **body=shiji.json()**  **except: # try 中报错执行本分支+ 断言 失败 执行本分支～**  **body="无结果～"**  **print("获取结果失败有bug～")**  **else: # try 中不报错执行本分支 +断言成功 执行本分支**    **print("获取结果成功无bug～")**  **finally: # 不管try 是否有异常 都要运行～**  **print(body)**  **print("获取结果结束～")**  **# # 发出下游接口**  **#**  **# # 运维服务器 + 接口地址**  **# user\_url='http://localhost:6088/api/User?IsEnabled=true&Page=1'**  **# url\_headers={'Authorization':shouquanma}**  **# user\_method='get'**  **#**  **#**  **# # = 左边API表**  **# # = 右边实际参数 来自于接口文档**  **#**  **# shiji=requests.get(url=user\_url,headers=url\_headers)**  **#**  **# print(shiji.status\_code)**  **# print(shiji.headers)**  **# print(shiji.json())** |

### 内容2： pytest灵魂三问：

**什么是Pytest？**

和unittest 一致 都是 单元测试框架 （测试开发写的代码）

共同点 组织test打头用例 提供断言

unittest 优点 ： 基础 官方 缺点 弱 ---》自测+简单单元测试 Selenium

**为什么pytest？**

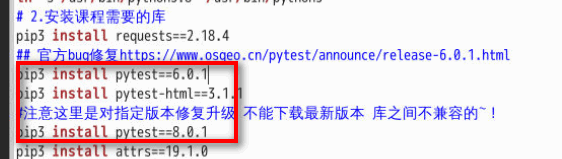
**ddt 数据驱动用例 用例调试 驱动 python执行用例的次数**

**优点： 1精简 2 pytest 更方便 ddt 3自带测试报告 4可以和其他工具 allure插件整合**

**缺点： 注意版本选型 pytest6.0.1===>pytest 8**

**如何掌握pytest？**

**环境**

****

**1 和unittest 通用（组织用例+断言）+被测代码**

**2 重点 数据驱动**

**3 产出自带报告**

### 内容3：快速入门

例子： 3条用例~

|  |
| --- |
| import unittest  import pytest  class Test\_login():  def setup\_class(self):  print("类中所有用例准备工作")  def teardown\_class(self):  print("类中所有用例结束工作")  def setup\_method(self): # 注意 u 和d小写  print("我是每条用例准备工作～")  def teardown\_method(self):  print("我是每条用例结束工作～")  def test\_11(self):  print("我是第一条用例～")  def test\_22(self):  print("我是第二条用例～")  def test\_33(self):  print("我是第三条用例～")  if \_\_name\_\_ == '\_\_main\_\_':  # pytest默认 精简报告--方便查看  # -s 参数 支持输出调试日志信息～  # -v 参数 支持显示用例执行过程  pytest.main(['-v','-s','jw33.py']) |

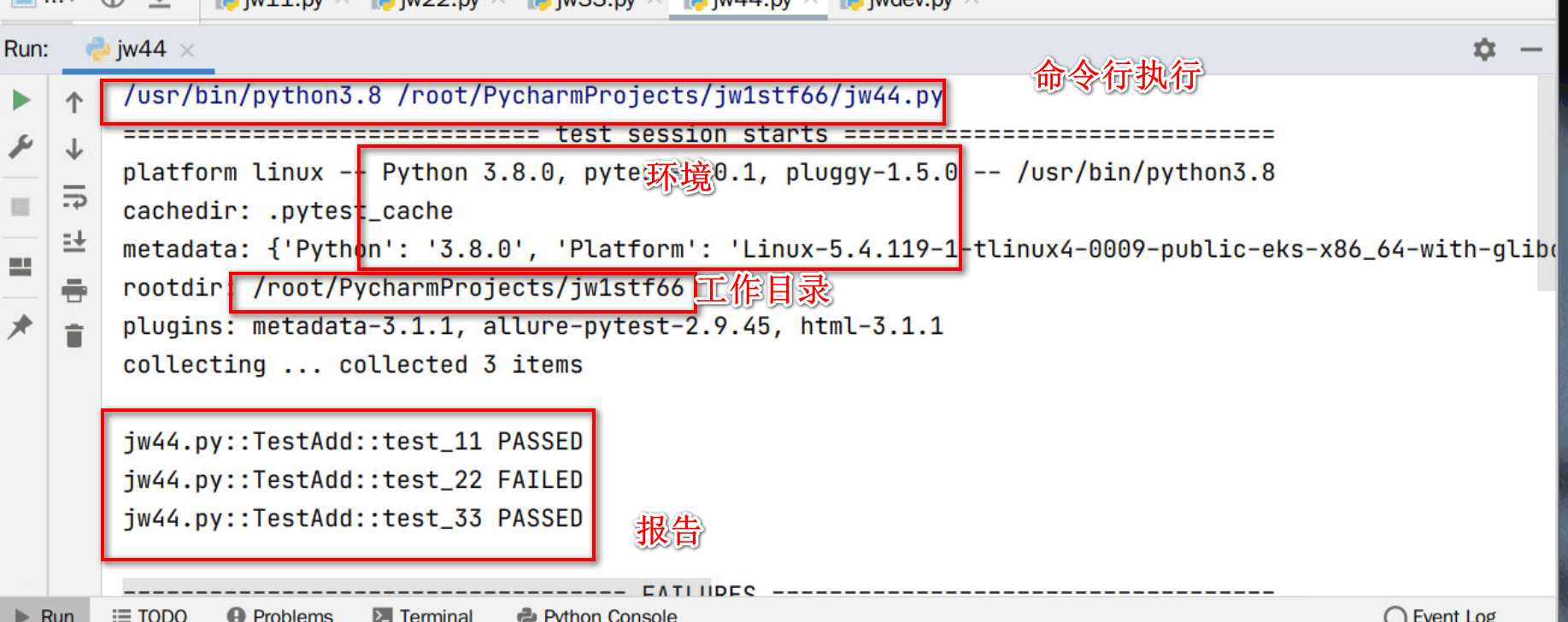
被测代码：

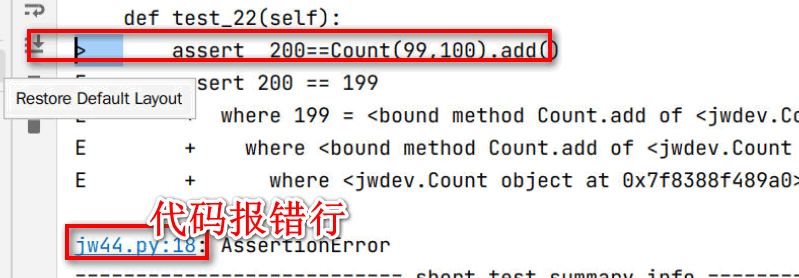
|  |
| --- |
| class Count:  def \_\_init\_\_(self,a,b):  self.a=int(a)  self.b=int(b)  def add(self):  return self.a+self.b  def sub(self):  return self.a-self.b |

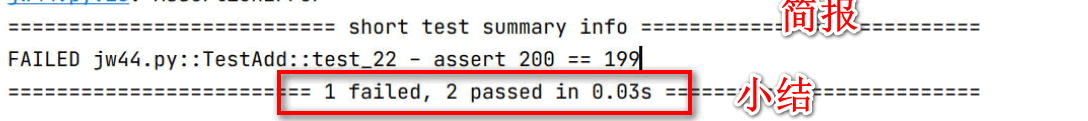
测试代码：

|  |
| --- |
| import pytest  from jwdev import Count  class TestAdd():  def test\_11(self):  # 1设计用例～===》预期结果  a=3  b=5  yuqi=8  # 2 调用开发写的代码===》实际结果～  c1=Count(a,b)  shiji=c1.add()  assert yuqi==shiji  def test\_22(self):  assert 200==Count(100,100).add()  def test\_33(self):  assert 202==Count(101,101).add()  if \_\_name\_\_ == '\_\_main\_\_':  pytest.main(['-v', '-s', 'jw44.py']) # 不要忘了改成你的文件名~ |

小结： 查看 测试结果~







小结2： pytest 和unittest 断言：

**2.1测试固件**

**setup\_class() 和teardown\_class() 所有用例准备工作和结束工作**

类用例函数界别（注意 和unittest不一样 u和d 不大写哈）

**pytest 8以前 如下写法**

**setup() 和teardown() 每条用例准备和结束**

**pytest 8（包含8）**

**setup\_method() 和teardown\_method() 每条用例准备和结束**

**2.2 测试断言**

2.2.1 是否相等断言

assert x==y 和 assert x !=y

2.2.2 是否真假断言

assert x 真True 和 assert not x

2.2.3 是否 包含断言

assert x in y

**练习题： 测试开发写素数（除了1个本身不能被整除的数字）功能：**

被测代码：

|  |
| --- |
| def is\_prime(n):  if n<=1:  return False  for i in range(2,n): # 除了一个本身 2~n-1  if n%i==0:  return False  return True |

请使用pytest 编写测试用例 断言功能是否正确~

|  |
| --- |
| import pytest  from jwdev2 import is\_prime  class Test\_prime():  def test\_prime1(self):  assert is\_prime(19)  def test\_prime2(self):  assert not is\_prime(9)  if \_\_name\_\_ == '\_\_main\_\_':  pytest.main(['-s','-v','jw55.py']) |

小结 组织用例+断言~~

### 内容4： pytest ddt（参数化 数据作为用例的参数）

**需求：每条用例思想一致 （都是 调用被测代码比对预期）==》过程代码一致 ，数据用例数量 决定用例的次数‘’**

**解决**

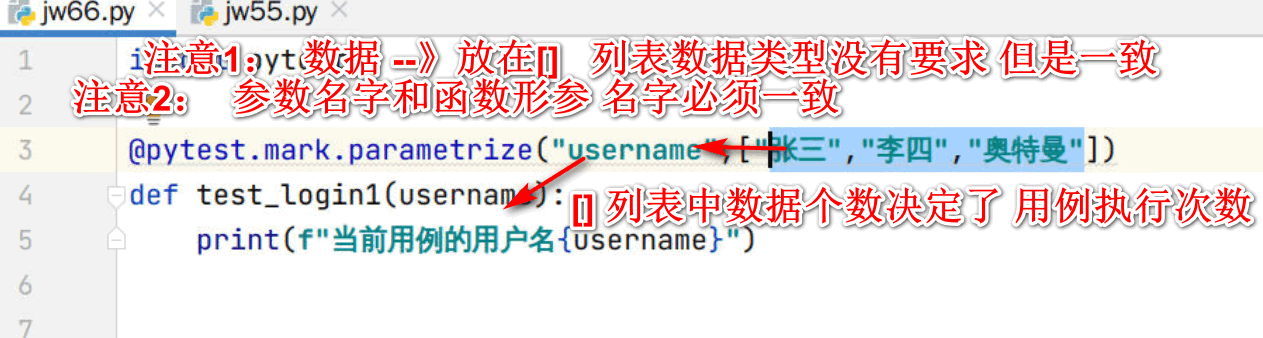
**pytest 装饰器**

**@pytest.mark.parametrize（参数名字，[用例数据]）**

**def test\_用例（参数）:**

**pass**

**语法：注意点**

****

|  |
| --- |
| **import pytest**  **@pytest.mark.parametrize("username",["张三","李四","奥特曼"])**  **def test\_login1(username):**  **print(f"当前用例的用户名{username}")**  **if \_\_name\_\_ == '\_\_main\_\_':**  **pytest.main(['-s', '-v', 'jw66.py'])** |

**例子 升级 加法用例 ddt 方式实现**

|  |
| --- |
| **import pytest**  **from jwdev import Count**  **class TestAdd():**  **@pytest.mark.parametrize('data',[(3,5,8),(100,100,9),(20,21,41)])**  **def test\_add(self,data):**  **print("当前用例数据",data)**  **# data[0] 元组下标**  **assert data[2]==Count(data[0],data[1]).add()**  **if \_\_name\_\_ == '\_\_main\_\_':**  **pytest.main(['-v', '-s', 'jw44.py'])** |

### 内容5： 单参数和多参数

**例子： 登录接口测试 一条用例三个数据 包含用户名 密码 和预期结果~**

**方式1： 单参数 参数名data 用例格式[{"username":"admin","pwd":"123","yuqi":True},**

**{"username":"志玲","pwd":"666","yuqi":True},**

**{"username":"奥特曼","pwd":"999","yuqi":False}]**

**方式 2： 多参数**

**参数1 username ["admin","志玲",“奥特曼”]**

**参数2 pwd ["123“,"666","999"]**

**参数3 yuqi [True,True,False]**

**例子1 问 选择 单参数还是多参数？**

**单参数：**

|  |
| --- |
| **import pytest**  **# 以后主要用单参数**  **# 缺点丑 优点 预期明确 功能没问题～**  **# 虽然丑 ---大量的用例 ---》mysql数据库中的表格---》替换成pymysql代码～**  **@pytest.mark.parametrize("data",[{"username":"admin","pwd":"123","yuqi":True},**  **{"username":"志玲","pwd":"666","yuqi":True},**  **{"username":"奥特曼","pwd":"999","yuqi":False}**  **])**  **def test\_login1(data):**  **print(f'当前用例的用户名{data["username"]},'**  **f'密码{data["pwd"]},'**  **f'预期结果{data["yuqi"]}')**  **if \_\_name\_\_ == '\_\_main\_\_':**  **pytest.main(['-s', '-v', 'jw777.py'])** |

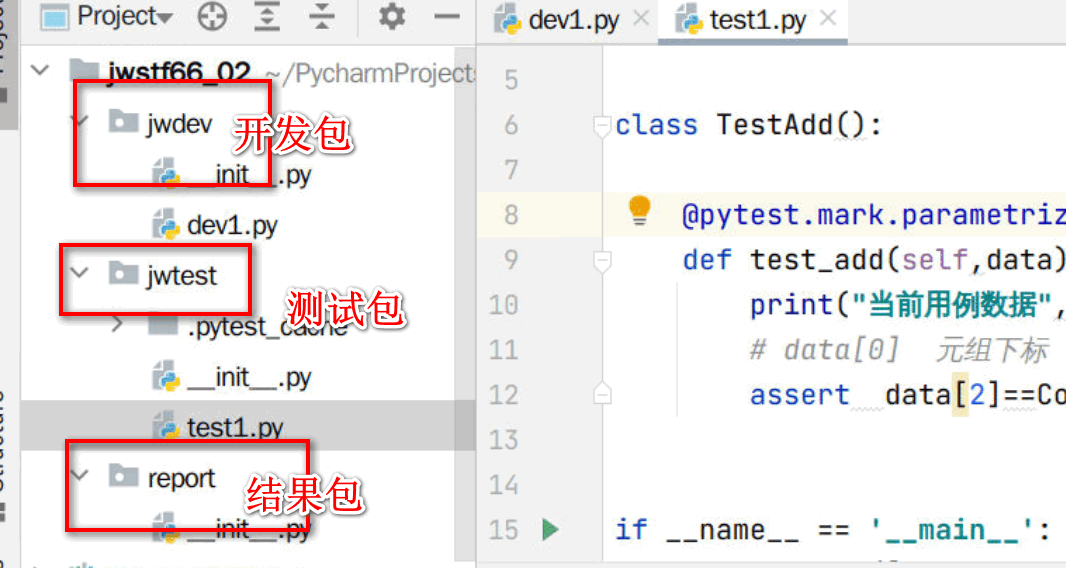
**多参数:**

|  |
| --- |
| **import pytest**  **#优点简洁 但是功能问题--->不好确定预期结果的～**  **# pytest 少量数据 产生大量的测试用例~**  **# 场景： 被测系统长时间稳定运行 大量用例**  **@pytest.mark.parametrize("username",["admin","志玲","奥特曼"])**  **@pytest.mark.parametrize("password",["123","666","999"])**  **@pytest.mark.parametrize("yuqi",[True,True,False])**  **def test\_login(username,password,yuqi):**  **print(f"当前用例用户名{username},密码{password},预期{yuqi}")**  **if \_\_name\_\_ == '\_\_main\_\_':**  **pytest.main(["-sv","jw888.py"])** |

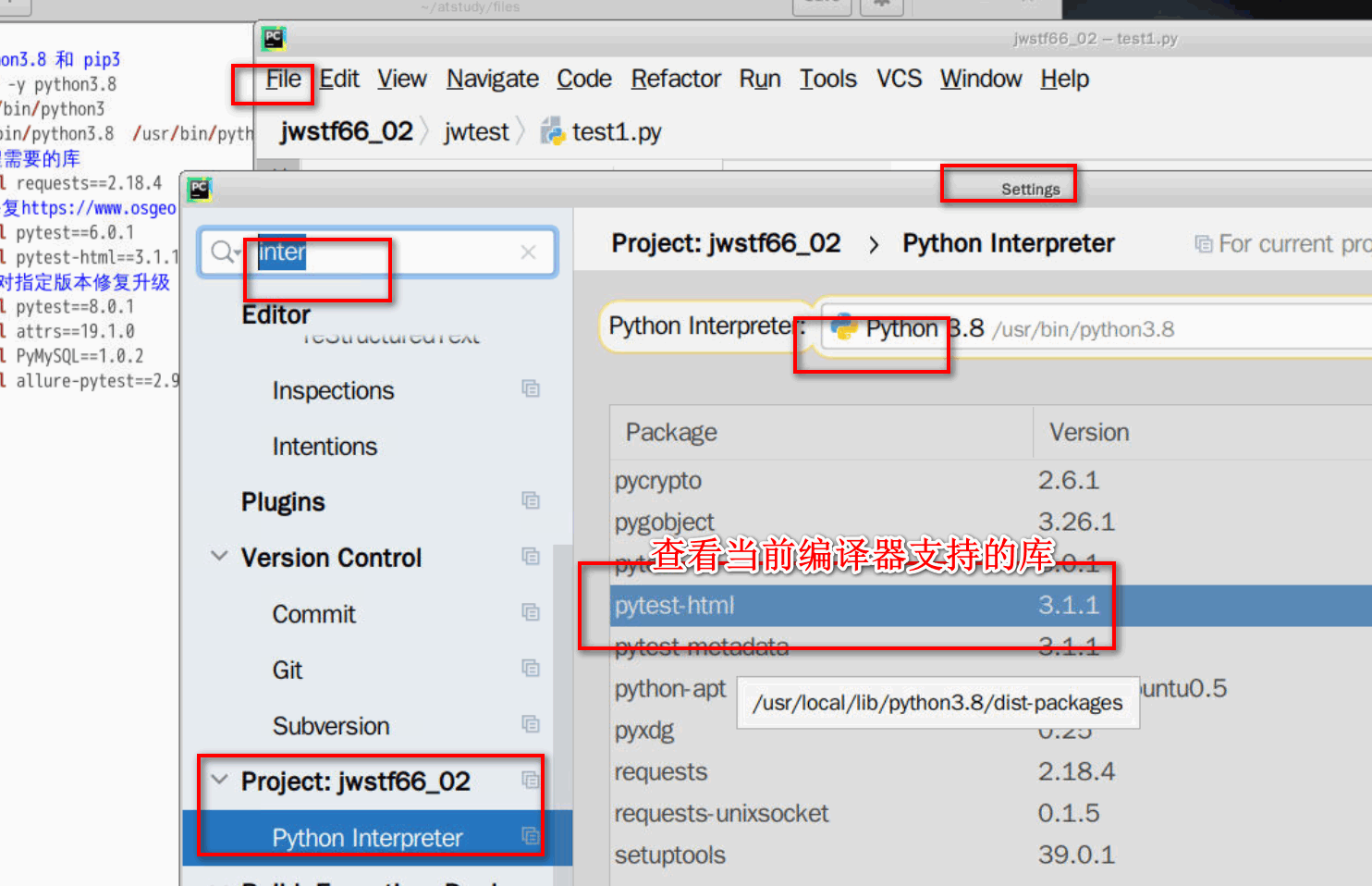
**小结： 主要以单参数为主**

### 内容6：pytest 测试报告

需求：更改项目规范 不同包负责不同职责~

****

pytest 自带了测试报告 ---安装库~

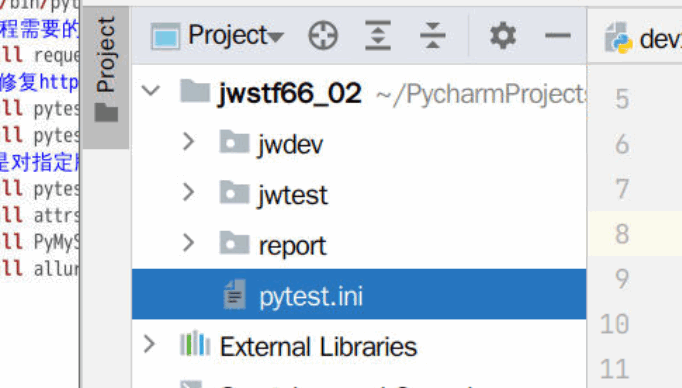


产出报告步骤

**pytest 要求 需要一个 文件 声明 运行 XXXX用例 产出 XXX测试报告（存放在 Report目录~）**

文件语法要求：

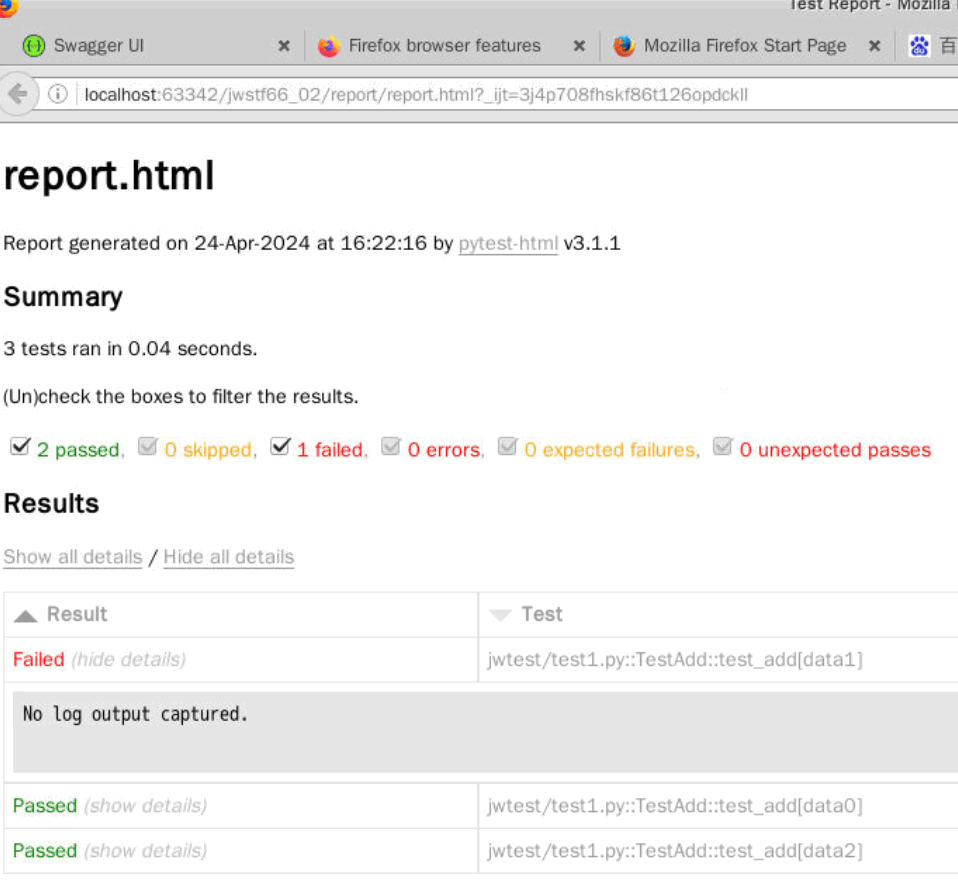
|  |
| --- |
| 1 文件名字必须 pytest.ini  2 文件路径 必须在项目路径下 不能在包下  **3**  pytest.ini 项目配置  **配置如何编写---》声明 用例满足如下条件产生报告**  **testpaths = 用例所在的包**  **python\_files=用例所在文件形式 [jwtest\*.py](https://jwtest*.py) jwtest打头.py 结尾文件都算**  **python\_classes=用例 类 Test\***  **python\_functions=用例方法名字 test\***  **addopts = 指定报告的位置 比如放到report目录下**  **4 还是运行用例 （pytest 查看发现用例满足pytest.ini 文件声明）==》就能产出报告** |



模版

|  |
| --- |
| [pytest]  addopts = -s -v --html=../report/report.html  testpaths = jwtest  python\_files = test\*.py  python\_classes = Test\*  python\_functions = test\* |

运行用例



总结：

1语法加强 get和try

2 pytest 断言+ 数据驱动ddt

3 pytest.ini 文件即可

预告 logging+ 封装成工具类