# 一.pytest单元测试框架

(1)什么是单元测试框架

单元测试是指在软件开发当中，针对软件的最小单位（函数，方法)进行正确性的检查测试。

(2)单元测试框架

java : junit和testng

python : unittest和pytest

(3)单元测试框架主要做什么?

1.测试发现:从多个文件里面去找到我们测试用例

2.测试执行∶按照一定的顺序和规则去执行。并生成结果

3.测试判断:通过断言判断预期结果和实际结果的差异

4.测试报告︰统计测试进度，耗时，通过率，生成测试报告。

# 二、单元测试框架和自动化测试框架有什么关系?

(1)什么是自动化测试框架

Requests + Pytest +Pymysql + Logging + Git + Jenkins

(2)自动化框架的作用

1.提高测试效率，降低维护成本。

2.减少人工干预，提高测试的准确性，增加代码的重用性。

3.核心思想是让不懂代码的人也能够通过这个框架去实现自动化测试。

(3)pytest单元测试框架和自动化测试框架的关系

单元测试框架:只是自动化测试框架中的组成部分之一。

# 三.pytest简介

1.pytest是一个非常成熟的python的单元测试框架，比unittest更灵活，容易上手。完全兼容unittest。

2.pytest可以和selenium , requests, appium结合实现web自动化，接口自动化，app自动化。

3.pytest可以实现测试用例的跳过以及reruns失败用例重试。

4.pytest可以和allure生成非常美观的测试报告。

5.pytest可以和Jenkins持续集成。

6.pytest有很多非常强大的插件，并且这些插件能够实现很多的实用的操作。

pytest-html (生成html格式的自动化测试报告)

pytest-xdist 测试用例分布式执行。多CPU分发。

pytest-ordering 用于改变测试用例的执行顺序

pytest-rerunfallures 用例失败后重跑

allure-pytest 用于生成美观的测试报告。

验证是否安装成功使用命令 pytest --version

# 四、使用pytest，默认的测试用例的规则以及基础应用

1.模块名必须以test\_开头或者\_test结尾

2.测试类必须以Test开头，并且不能有init方法。

3.测试方法必须以test开头

# 五、pytest测试用例的运行方式

## 1.主函数模式

(1)运行所有: pytest.main()

(2)指定模块: pytest.main(['-vs', "模块名.py"])

(3)指定目录: pytest.main(['-vs',’./当前目录下指定文件夹'])

(4)通过nodeid指定用例运行:nodeid由模块名，分隔符，类名，方法名，函数名组成。

pytest.main([‘-vs’,’./当前目录下指定文件夹/模块名.py::方法名’])

pytest.main([‘-vs’,’./当前目录下指定文件夹/模块名.py::类名::方法名')



如果使用pytest.main()生效，再pycharm中必须设置Tools>Python Integrated Tools>Default test runner=Unittests；如果设置为pytest则是无效的。

列表可选参数详解:

pytest.main(['-s', '-v', '-q', '-x','-k "print"'])

-s 对于代码里print语句和日志信息会将输出至控制台

-v 详细的输出用例执行的过程和结果信息

-q 静默模式,精简输出运行结果信息

-x 出现一条用例执行失败就退出测试，调试阶段常用

-k "print" 运行包含某个字符串的用例，如pytest -k add XX.py 表示运行XX.py中包含add的测试用例等等,匹配规则是符合的目录名、模块名、类名、方法名

-m 对用例进行标记，用例需注释@pytest.mark.xxx,将xxx作为参数传入,标记需要在pytest.ini文件注册,否则会有warning信息

-l 显示变量信息

-rsxX 表示pytest报告所有测试用例被跳过、预计失败、预计失败但实际被通过的原因

--html ./report/report.html 生成html的测试报告

## 2.命令行模式

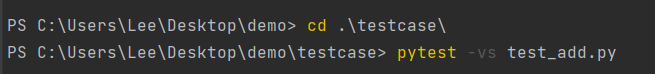
(1)运行所有:pytest

(2)指定模块: pytest -vs telst\_login.py

(3)指定目录: pytest -vs ./demo

(4)指定目录: pytest -vs ./demo /test\_add.py::test\_04





## 3.通过读取pytest.ini配置文件运行。

pytest.ini这个文件它是pytest单元测试框架的核心配置文件。

1.位置∶一般放在项目的根目录

2.作用∶改变pytest默认的行为。

3.运行的规则 ：不管是主函数的模式运行，命令行模式运行，都会去读取这个配置文件。

[pytest]

addopts = -vs #命令行的参数，用空格分隔

testpaths = ./testcase #测试用例的路径

python\_filles = test\_\* \*\_test check\_\* #模块名的规则

python\_classes = Test\* Check\* #类名的规则

python\_functions = test­\_\* check\_\* #方法名的规则

markers =

smoke:冒烟用例

usermanage:用户管理模块

productmanage:商品管理模块

# 六. pytest执行测试用例的顺序

unittest: ascii码的大小来绝对的执行的顺序

pytest: 默认从上到下

改变默认的执行顺序∶使用mark标记。

@pytest.mark .run (order=3) #需要安装pytest-ordering库

# 七、如何分组执行（冒烟，分模块执行，分接口和web执行)

smoke : 冒烟用例，分布在各个模块里面

pytest -m "smoke"

pytest -m "smoke or usermanage or productmanage"

# 八. pytest跳过测试用例

(1)无条件跳过

@pytest.mark.skip(reason="你跳过的原因")

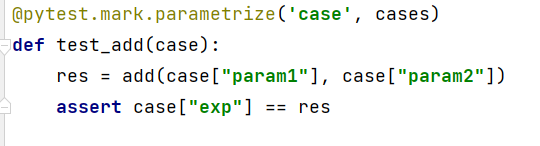
(2)有条件跳过

@pytest.mark .skipif(age>=18, reasan="你跳过的原因")

# 九. pytest的参数化

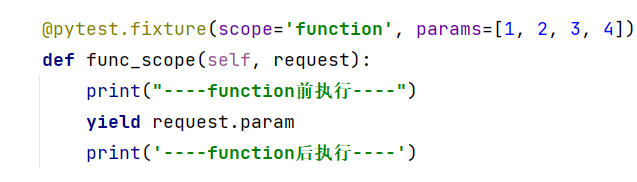
(1) @pytest.mark.parametrize(变量名:str,数据源:list)

通过装饰器装饰需要参数化的测试case,第一个参数是变量名,case必须接收相同的变量名参数；第二个参数数据源要求是列表类型,列表中可以是字典、字符串成员,列表中每有一个成员case会被执行一次



(2)通过测试固件参数化

@pytest.fixture(params=数据:list) 使用夹具的测试用例会被重复执行,次数等于params列表的成员个数，注意只有把夹具当作入参传给测试用例时才可以使用夹具传的参数



# 十. pytest的测试固件(夹具)

## 1.使用夹具的四种方式

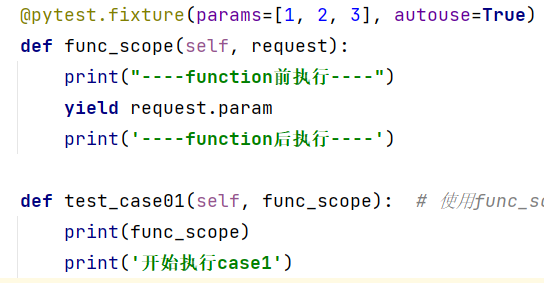
(1)声明夹具时设置自动使用:

@pytest.fixture(autouse=True)

(2)在需要使用夹具的类或方法前使用装饰器:

@pytest.mark.usefixtures("夹具名")

(3)把夹具名作为入参传入方法中,这种方式可以接收夹具返回的参数:



(4)在conftest.py文件配置全局固件,注意不同包下可以有多个conftest.py，但不可以跨包调用。根目录的conftest.py文件会影响所有的case。当根目录和当前目录的conftest.py冲突时,按就近原则，当前目录下的conftest.py优先级最高

## 2.声明夹具的参数

(1) scope参数:

function（函数）：每个测试用例函数执行时都会执行一次，默认值

class（类）：不论有多少测试用例，整个类只会运行一次

module（模块）：不论有多少测试用例，整个模块（文件）下只运行一次

package（包）：不论有多少测试用例，整个包（文件夹）下只运行一次

session：不论有多少测试用例，整个pytest下只会运行一次

(2) autouse参数:

默认为False,设置为True，即为自动的意思，默认就会使用fixtrue

(3) params 参数:

可使用列表格式数据参数化测试用例,效果同@pytest.mark.parametrize装饰器

(4) ids参数:

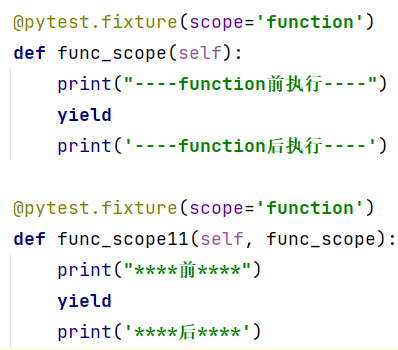
可以为参数化固件运行的case重命名

(5) name参数:

为当前的固件重命名,注意一旦重命名原固件名就无法使用固件

## 3.夹具之间可以相互依赖

夹具之间可以相互依赖,一旦你使用夹具那么他的依赖夹具也会被调用



# 十一. pytest生成allure报告

1.windows下载对应版本的allure压缩包，解压到D:/Tools目录下: https://repo.maven.apache.org/maven2/io/qameta/allure/allure-commandline/



2. 追加allure的bin目录到环境变量path中

3. 在cmd命令行输入 allure 回车,验证是否安装成功

4. 在cmd命令行输入 pip install allure-pytest 安装pytest框架的allure插件

pytest中使用allure生成测试报告有两种命令方式：

1.第一种：

第一步：输入下面的命令，执行pytest生成allure的json结果文件：

Pytest –alluredir ./report\_json test\_case.py

第二步：输入下面的命令生成html文件并启动一个服务，通过访问链接浏览html报告：  
allure serve ./report\_json

2.第二种：

第一步：输入下面的命令，执行pytest生成allure的json结果文件：

Pytest –alluredir ./report\_json test\_case.py

第二步：生成html文件，文件存放路径为【./allure\_report/】

allure generate ./report\_json –o ./allure\_report --clean

第三步：启动一个服务，通过访问链接浏览html报告：

allure open ./allure\_report/

# 十二. allure装饰器介绍



#### @allure 装饰器中的一些功能点：

1）想对你的用例集分层，增加可读性，可以使用以下三个装饰器，写在类或方法前面：

* @allure.epic ：敏捷里的概念，定义史诗
* @allure.feature ：用于定义被测试的功能，被测产品的需求点
* @allure.story ： 用于定义被测功能的用户场景，即子功能点

2）想对单个用例添加标题和用例描述，增加可读性，使用以下两个装饰器：

* @allure.description ：添加测试用例描述，参数类型为str，与使用''' '''效果类似。如果想传html，请使用 @allure.description\_html
* @allure.title ：修改单个测试用例标题，支持传递关键字参数

3）动态生成功能，以下方法都支持动态生成，也可以覆盖装饰器 @allure 的内容

* allure.dynamic.feature
* allure.dynamic.link
* allure.dynamic.issue
* allure.dynamic.testcase
* allure.dynamic.story
* allure.dynamic.title
* allure.dynamic.description

4）如果想对每个测试用例进行非常详细的步骤信息说明，提高可读性，可以使用以下两个方法：

* allure.step ：对每个测试用例进行步骤说明
* allure.attach ：输出内容

先看下 allure.attach(body, name=None, attachment\_type=None, extension=None) 定义，参数如下

* body：要显示的内容（附件）
* name：附件名字
* attachment\_type：附件类型，是 allure.attachment\_type 里面的其中一种
* extension：附件的扩展名

也可以使用文件路径：allure.attach.file( source, body, name=None, attachment\_type=None, extension=None)，source为文件路径，其他参数和上述一致