内容回顾

1. pytest下载：`pip install pytest`
2. pytest.main(["-s", "-v", "demo.py"])
    1. pytest的脚本名字有要求：
        1. 脚本名称中不能含有`.`，如`1.内容回顾.py`，你可以使用`1-内容回顾.py`
        2. 脚本名称中避免使用中文，如`1.内容回顾.py`
3. pytest中，关于用例定义必须：
    1. 函数形式，必须是`test`开头
    2. 类的形式，必须是`Test`开头，类中的方法必须是以`test`开头
4. setup/teardown
    1. 模块级别的
        1. setup_module
        2. teardown_module
    2. 函数级别的
        1. setup_function
        2. teardown_function
    3. 类级别的
        1. setup_class
        2. teardown_class
    4. 类中方法级别的
        1. setup_method
        2. teardown_method
5. pytest中的配置文件`pytest.ini`的使用，如果在项目目录下新建了`pytest.ini`那么，在执行用例的时候，会自动能的加载配置文件中的配置信息。

```
[pytest]
addopts = -s -v
testpaths = ./scripts
python_files = test_*.py
python_classes = Test*
python_functions = test_*
```

注意，配置文件中不允许有中文。

6. 跳过：
    1. pytest.mark.skip(reason)， 无条件跳过，reason是跳过原因。
    2. pytest.mark.skipif(condittion, reason)， condition是跳过的条件，reason是跳过原因。

7. 标记预期失败，如一个用例我们认为它会执行失败，那我们就预期它失败，并希望对该用例进行标记。
    1. 预期失败，执行失败，`XFAILD`
    2. 预期失败，执行成功，`XPASS`
    3. 如果预期失败，但执行成功，我们认为它不符合逻辑，要把这种用例判定为执行失败，就要在配置文件中添加参数`xfail_strcit=true`
8. 参数化：

```
import pytest

l = [1, 2, 3, 4]

@pytest.mark.parametrize("name", l)
def test_case(name):
    print(name)

l1 = [1, 2, 3, 4]
l2 = ["a", "b", "c"]

@pytest.mark.parametrize("name1,name2", list(l1, l2))
def test_case_02(name1, name2):
    print(name1, name2)
```

9. 固件，也叫测试夹具，就在测试用例之前之后执行的一些方法，功能类似于setup/teardown，但是固件更加的灵活。

```
import pytest

@pytest.fixture()
def login():

    print("loging")

def test_index(login):

    print("index page")



# 预处理和后处理
@pytest.fixture()
def db():
    print("连接数据库....")
    yield

    print("断开数据库连接....")

def test_select_user(db):

    print("index page")
```

10. 失败重跑插件，`pip install pytest-rerunfailures`，使用时在配置文件中配置`addopts = -s --html=report/report.html --reruns=3`。
11. 控制用例执行顺序，`pip install pytest-ordering`，使用：

```python
import pytest

class TestCaseClass(object):
    @pytest.mark.run(order=3)
    def test_case_03(self):
        print('执行用例03.......')
        assert 1

@pytest.mark.run(order=2)
def test_case01():
    print('执行用例01.......')
    assert 1  # 断言成功

@pytest.mark.run(order=1)
def test_case02():
    print('执行用例02.......')
    assert 1  # 断言成功
```

10. 第三方的插件，生成测试报告`pip install pytest-html`，使用时在配置文件中配置`addopts = -s --html=report/report.html`，`report`目录和配置文件同级别。
11. allure报告，轻量级的第三方的库，有java语言实现。可以跨语言。
    1. 必须按照allure插件，然后将其内的`bin`目录添加到PATH中。
    2. 由于是java实现，所以依赖java环境，
    3. Python需要下载allure包，`pip install allure-pytest`
    4. 工作原理：
        1. 在pytest的配置文件中配置`addopts =  -v -s --html=report/report.html --alluredir ./report/result`，意思是将allure需要的json数据保存到什么位置。。
        2. 使用allure命令来读取上一步生成的json文件，生成allure报告，你要指定报告保存的位置

        ```
        allure generate json数据所在目录 -o 测试报告保存的目录 --clean
        ```

        3. 由于allure报告的查看看依赖HTTP服务器，所以，你可以：
            1. 使用pycharm自带的
            2. 使用allure open命令
    5. allure的其他用法
        1. title
        2. description
        3. severity
        4. feature/story，行为标记
        5. dynamic，跟pytest的参数化使用

今日内容

接口自动化的框架开发：

- 用到的知识点:
    - pytest

    - allure
    - 参数化
    - Excel操作，不会，用xlrd
    - 日志操作，学过，不太会
    - 邮件，会
    - 文件操作，文件压缩， 没讲，但你要会的，zipfile
    - 执行终端命令，os.system, subprocess:cell, popen
        - 如何使用python查看当前目录下的所有文件或者目录?
- 实现的个功能：
    - 将各个功能拆分为多个目录
    - 使用参数化读取Excel中的用例
        - 发请求
        - 获取请求结果
        - 校验/断言
    - 使用allure生成测试报告
    - 将allure测试报告所在的目录打包
    - 将打包的zip文件使用邮件发送到1206180814@qq.com
    - 在重点位置加日志

实现思路：

1. 读取Excel，每一行数据都是一个用例，你在读出来之后，把这个一行用例封装成一个对象，字典，列表。
2. 使用参数化每次讲一个用例对象传进去。
3. 使用requests获取用例对象中的相关参数发请求。
4. 然后将请求结果与预期值（用例对象）做断言
5. 此时，allure所需的json数据已经有了。
6. 使用allure命名读取josn数据生成测试报告
7. 将报告压缩
8. 使用发邮件功能将压缩文件发送
9. 在重点位置，添加日志功能