Table of Contents

UI自动化测试课程 1.1

PyTest框架 1.2

- PyTest基本使用 1.2.1
- PyTest常用插件 1.2.2
- PyTest高级用法 1.2.3

UI自动化测试课程

序号	章节	知识点
1	UI自动化测试介绍	1. UI自动化测试
2	Web自动化测试基础	1. Web自动化测试框架 2. 环境搭建 3. 元素定位和元素操作 4. 鼠标和键盘操作 5. 元素等待 6. HTML特殊元素处理 7. 验证码处理
3	移动自动化测试基础	1. 移动自动化测试框架 2. ADB调试工具 3. UIAutomatorViewer工具 4. 元素定位和元素操作 5. 滑动和拖拽事件 6. 高级手势TouchAction 7. 手机操作
4	PyTest框架	1. PyTest基本使用 2. PyTest常用插件 3. PyTest高级用法
5	PO模式	1. 方法封装 2. PO模式介绍 3. PO模式实战
6	数据驱动	1. 数据驱动介绍 2. 数据驱动实战
7	日志收集	1. 日志相关概念 2. 日志的基本方法 3. 日志的高级方法
8	黑马头条项目实战	1. 自动化测试流程 2. 项目实战演练

课程目标

1. 掌握使用Selenium实现Web自动化测试的流程和方法，并且能够完成自动化测试脚本的编写。
2. 掌握使用Appium实现移动自动化测试的流程和方法，并且能够完成自动化测试脚本的编写。
3. 掌握如何通过PyTest管理用例脚本，并使用Allure生成HTML测试报告。
4. 掌握使用PO模式来设计自动化测试代码的架构。
5. 掌握使用数据驱动来实现自动化测试代码和测试数据的分离。
6. 掌握使用logging来实现日志的收集。

PyTest框架

目标

1. 掌握pytest框架的基本用法
2. 能够生成 pytest-html 测试报告
3. 能够控制 pytest 函数执行的顺序
4. 能够掌握 pytest 失败重试
5. 能够掌握 pytest 跳过函数
6. 能够掌握 pytest 数据参数化

PyTest基本使用

目标

```
1. 能够安装 pytest 框架
2. 能够了解 pytest 主函数的运行方式
3. 能够掌握 pytest 命令行的运行方式
4. 能够掌握 setup 和 teardown 方法
5. 能够掌握 setup_class 和 teardown_class 方法
6. 能够理解 pytest 配置文件的含义
```

1. 介绍和安装

1.1 概念

pytest 是 python 的一种单元测试框架，同自带的 UnitTest 测试框架类似，相比于 UnitTest 框架使用起来更简洁，效率更高。

1.2 特点

1. 非常容易上手，入门简单，文档丰富，文档中有很多实例可以参考
2. 支持简单的单元测试和复杂的功能测试
3. 支持参数化
4. 执行测试过程中可以将某些测试跳过，或者对某些预期失败的 Case 标记成失败
5. 支持重复执行失败的 Case
6. 支持运行由 Nose，UnitTest 编写的测试 Case
7. 具有很多第三方插件，并且可以自定义扩展
8. 方便的和持续集成工具集成

1.3 安装

```
pip3 install pytest
```

安装校验

1. 进入命令行
2. 输入命令 `pytest --version` 会展示当前已安装版本

2. 运行方式

代码准备

test_login.py

```
class TestLogin:

    def test_a(self):  # test开头的测试函数
        print("------->test_a")
        assert 1  # 断言成功

    def test_b(self):
        print("------->test_b")
        assert 0  # 断言失败
```

运行方式分为两种：

- 命令行模式【建议】
  - 命令行中执行 `pytest -s test_login.py`
- 主函数模式
  - 在 test_login.py 文件中增加主函数

    ```
    if __name__ == '__main__':
        pytest.main(["-s", "login.py"])
    ```

  - -s 表示支持控制台打印，如果不加，print 不会出现任何内容

运行结果

```
test_login.py ------->test_a
.------->test_b
F
```

. 表示成功

F 表示失败

小结

建议使用命令行的形式运行，对比主函数模式更加方便

3. setup 和 teardown

应用场景

pytest 在运行自动化脚本的前后会执行两个特殊的方法，分别是 setup 和 teardown 。在执行脚本之前会执行 setup 方法，在执行脚本之后会执行 teardown 方法。有了这两个方法，我们可以在 setup 中进行获取驱动对象的操作，在 teardown 中进行关闭驱动对象的操作。

3.1 函数级别方法

运行于测试方法的始末，运行一次测试函数会运行一次 setup 和 teardown。

示例代码

```
import pytest

class TestLogin:

    # 函数级开始
    def setup(self):
        print("------->setup_method")

    # 函数级结束
    def teardown(self):
        print("------->teardown_method")

    def test_a(self):
        print("------->test_a")

    def test_b(self):
        print("------->test_b")
```

执行结果

```
scripts/test_login.py ------->setup_method  # 第一次 setup()
------->test_a
.------->teardown_method  # 第一次 teardown()
------->setup_method  # 第二次 setup()
------->test_b
.------->teardown_method  # 第二次 teardown()
```

3.2 类级别方法

运行于测试类的始末，在一个测试内只运行一次 setup_class 和 teardown_class，不关心测试类内有多少个测试函数。

示例代码

```
class TestLogin:

    # 测试类级开始
    def setup_class(self):
        print("------->setup_class")
```

```
    # 测试类级结束
    def teardown_class(self):
        print("------->teardown_class")

    def test_a(self):
        print("------->test_a")

    def test_b(self):
        print("------->test_b")
```

执行结果

```
scripts/test_login.py ------->setup_class  # 第一次 setup_class()
------->test_a
.------->test_b
.------->teardown_class  # 第一次 teardown_class()
```

4. 配置文件

应用场景

使用配置文件后可以快速的使用配置的项来选择执行哪些测试模块。

使用方式

1. 项目下新建 scripts 模块
2. 将测试脚本文件放到 scripts 中
3. pytest 的配置文件放在自动化项目目录下
4. 名称为 pytest.ini
5. 命令行运行时会使用该配置文件中的配置
6. 第一行内容为 `[pytest]`

示例

```
[pytest]
# 添加命令行参数
addopts = -s
# 文件搜索路径
testpaths = ./scripts
# 文件名称
python_files = test_*.py
# 类名称
python_classes = Test*
# 方法名称
python_functions = test_*
```

addopts = -s

表示命令行参数

testpaths，python_files，python_classes，python_functions

表示 哪一个文件夹 下的 哪一个文件 下的 哪一个类 下的 哪一个函数

表示执行 scripts 文件夹下的 test 开头 *.py* 结尾的文件下的 *Test* 开头的类下的 *test*开头的函数

PyTest常用插件

目标

```
1. 能够生成 pytest-html 测试报告
2. 能够控制 pytest 函数执行的顺序
3. 能够掌握 pytest 失败重试
```

1. 测试报告

应用场景

自动化测试脚本最终执行是通过还是不通过，需要通过测试报告进行体现。

安装

使用命令 `pip3 install pytest-html` 进行安装

使用

在配置文件中的命令行参数中增加 `--html=用户路径/report.html`

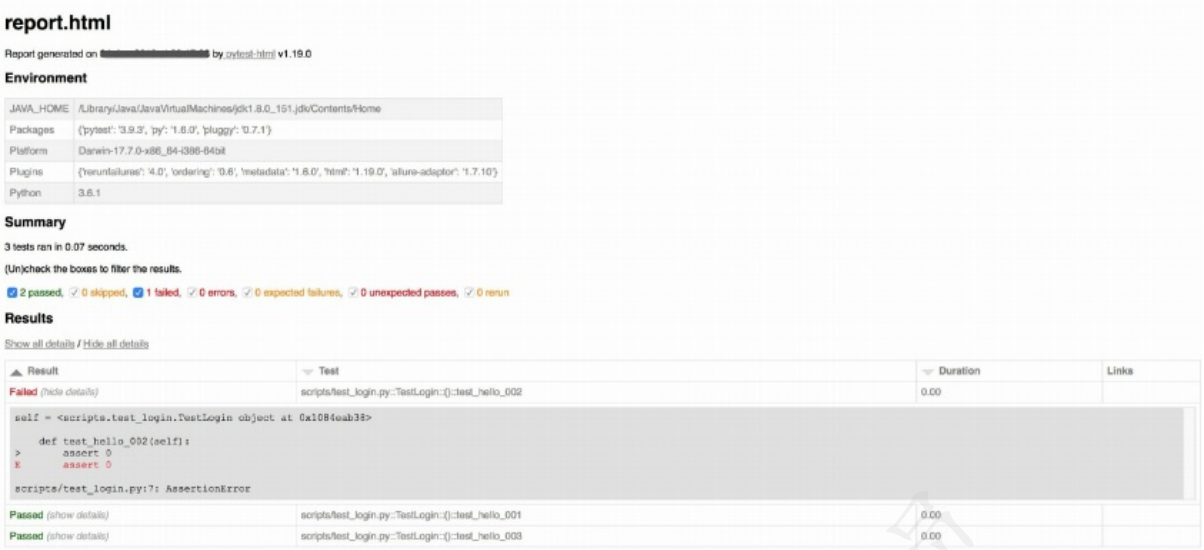
示例

pytest.ini

```
addopts = -s --html=report/report.html
```

结果

在项目目录下会对一个 report 文件夹，里面有个 report.html 即为测试报告



report.html

Report generated on [redacted] by pytest-html v1.19.0

Environment

JAVA_HOME	/Library/Java/JavaVirtualMachines/jdk1.8.0_151.jdk/Contents/Home
Packages	{'pytest': '3.9.3', 'py': '1.6.0', 'pluggy': '0.7.1'}
Platform	Darwin-17.7.0-x86_64-i386-64bit
Plugins	{'rerunfailures': '4.0', 'ordering': '0.6', 'metadata': '1.6.0', 'html': '1.19.0', 'allure-adaptor': '1.7.10'}
Python	3.6.1

Summary

3 tests ran in 0.07 seconds.

(Un)check the boxes to filter the results.

2 passed, 0 skipped, 1 failed, 0 errors, 0 expected failures, 0 unexpected passes, 0 rerun

Results

Show all details / Hide all details

Result	Test	Duration	Links
Failed (hide details)	scripts/test_login.py::TestLogin::()::test_hello_002	0.00	

```
self = <scripts.test_login.TestLogin object at 0x1084eab38>

    def test_hello_002(self):
>       assert 0
E       assert 0

scripts/test_login.py:7: AssertionError
```

Passed (show details)	scripts/test_login.py::TestLogin::()::test_hello_001	0.00	
Passed (show details)	scripts/test_login.py::TestLogin::()::test_hello_003	0.00	

2. 控制函数执行顺序

应用场景

现实生活中，如果想下订单，必须先登录，我们可以通过插件的形式来控制函数执行的顺序。

安装

使用命令 `pip3 install pytest-ordering` 进行安装

使用

1. 标记于被测试函数，@pytest.mark.run(order=x)
2. 根据order传入的参数来解决运行顺序
3. order值全为正数或全为负数时，运行顺序：值越小，优先级越高
4. 正数和负数同时存在：正数优先级高

示例

```
import pytest


class TestLogin:

    def test_hello_001(self):
        print("test_hello_001")

    @pytest.mark.run(order=1)
    def test_hello_002(self):
        print("test_hello_002")

    @pytest.mark.run(order=2)
```

```
    def test_hello_003(self):
        print("test_hello_003")
```

结果

```
scripts/test_login.py test_hello_002  # 先运行2
.test_hello_003  # 再运行3
.test_hello_001
.
```

---

3. 失败重试

应用场景

自动化测试脚本可能会使用到网络，如果网络不好可能最终会使脚本不通过。像这种情况可能并不是脚本本身的问题，仅仅是因为网络忽快忽慢，那么我们可以使用失败重试的插件，当失败后尝试再次运行。一般情况最终成功可以视为成功，但最好进行进行排查时候是脚本问题。

安装

使用命令 `pip3 install pytest-rerunfailures` 进行安装

使用

在配置文件中的命令行参数中增加 `--reruns n`

示例

pytest.ini

```
addopts = -s --reruns 3
```

test_login.py

```
class TestLogin:

    def test_a(self):  # test开头的测试函数
        print("------->test_a")
        assert 1  # 断言成功

    def test_b(self):
        print("------->test_b")
        assert 0  # 断言失败
```

结果

```
scripts/test_login.py ------->test_a
.------->test_b
R------->test_b
R------->test_b
R------->test_b
F
```

> R 表示重试

注意点

> 重试时，如果脚本通过，那么后续不再重试

PyTest高级用法

目标

1. 能够掌握 pytest 跳过函数
2. 能够掌握 pytest 数据参数化

1. 跳过测试函数

应用场景

同一个软件在不同的设备上可能会有不同的效果，比如，iOS 的 3d touch 操作是需要 6s 以上设备支持的，6 和 6s 都可以安装同一款应用，如果设备不支持，那么根本没有必要去测试这个功能。此时，我们可以让这种函数进行跳过。

方法名

```
# 跳过测试函数
# 参数：
#     condition：跳过的条件，必传参数
#     reason：标注原因，必传参数
@pytest.mark.skipif(condition, reason=None)
```

使用方式

在需要跳过的测试脚本之上加上装饰器 `@pytest.mark.skipif(condition, reason="xxx")`

示例

```
import pytest


class TestLogin:

    def test_a(self):  # test开头的测试函数
        print("------->test_a")
        assert 1  # 断言成功

    @pytest.mark.skipif(condition=True, reason="xxx")
    def test_b(self):
        print("------->test_b")
        assert 0  # 断言失败
```

结果

```
scripts/test_login.py ------->test_a
.s
```

2. 数据参数化

应用场景

登录功能都是输入用户名，输入密码，点击登录。但登录的用户名和密码如果想测试多个值是没有办法用普通的操作实现的。数据参数化可以帮我实现这样的效果。

方法名

```
# 数据参数化
# 参数:
#    argnames：参数名
#    argvalues：参数对应值，类型必须为可迭代类型，一般使用list
@pytest.mark.parametrize(argnames, argvalues, indirect=False, ids=None, scope=None)
```

一个参数使用方式

1. argnames 为字符串类型，根据需求决定何时的参数名
2. argvalues 为列表类型，根据需求决定列表元素中的内容
3. 在测试脚本中，参数，名字与 argnames 保持一致
4. 在测试脚本中正常使用

> argvalues 列表有多少个内容，这个脚本就会运行几次

示例

```
import pytest


class TestLogin:

    @pytest.mark.parametrize("name", ["xiaoming", "xiaohong"])
    def test_a(self, name):
        print(name)
        assert 1
```

结果

```
scripts/test_login.py xiaoming
.xiaohong
.
```

多个参数使用方式

示例

```
import pytest


class TestLogin:

    @pytest.mark.parametrize(("username", "password"), [("zhangsan", "zhangsan123"), ("
xiaoming", "xiaoming123")])
    def test_a(self, username, password):
        print(username)
        print(password)
        assert 1
```

结果

```
scripts/test_login.py zhangsan
zhangsan123
.xiaoming
xiaoming123
.
```

> 多个参数还可以将装饰器写成 `@pytest.mark.parametrize("username,password", [("zhangsan", "zhangsan123"), ("xiaoming", "xiaoming123")])` 效果是一样的。