**用 Pytest+Appium+Allure 做 UI 自动化的那些事**

文本主要介绍下 Pytest+Allure+Appium 记录一些过程和经历，一些好用的方法什么的，之前也没写过什么文章，文章可能有点干，看官们多喝水

**主要用了啥:**

* Python3
* Appium
* Allure-pytest
* Pytest

**Appium 不常见却好用的方法**

**Appium 直接执行 adb shell 方法**

# Appium 启动时增加 --relaxed-security 参数 Appium 即可执行类似adb shell的方法  
> appium -p 4723 --relaxed-security

# 使用方法  
def adb\_shell(self, command, args, includeStderr=False):  
 """  
 appium --relaxed-security 方式启动  
 adb\_shell('ps',['|','grep','android'])  
  
 :param command:命令  
 :param args:参数  
 :param includeStderr: 为 True 则抛异常  
 :return:  
 """  
 result = self.driver.execute\_script('mobile: shell', {  
 'command': command,  
 'args': args,  
 'includeStderr': includeStderr,  
 'timeout': 5000  
 })  
 **return** result['stdout']

**Appium 直接截取元素图片的方法**

element = self.driver.find\_element\_by\_id('cn.xxxxxx:id/login\_sign')  
pngbyte = element.screenshot\_as\_png  
image\_data = BytesIO(pngbyte)  
img = Image.open(image\_data)  
img.save('element.png')  
# 该方式能直接获取到登录按钮区域的截图

**Appium 直接获取手机端日志**

# 使用该方法后，手机端 logcat 缓存会清除归零，从新记录  
# 建议每条用例执行完执行一边清理，遇到错误再保存减少陈余 log 输出  
# Android  
logcat = self.driver.get\_log('logcat')  
  
# iOS 需要安装 brew install libimobiledevice   
logcat = self.driver.get\_log('syslog')  
  
# web 获取控制台日志  
logcat = self.driver.get\_log('browser')  
  
c = '\n'.join([i['message'] **for** i in logcat])  
allure.attach(c, 'APPlog', allure.attachment\_type.TEXT)  
#写入到 allure 测试报告中

**Appium 直接与设备传输文件**

# 发送文件  
#Android  
driver.push\_file('/sdcard/element.png', source\_path='D:\works\element.png')  
  
# 获取手机文件  
png = driver.pull\_file('/sdcard/element.png')  
with open('element.png', 'wb') as png1:  
 png1.write(base64.b64decode(png))  
  
# 获取手机文件夹，导出的是zip文件  
folder = driver.pull\_folder('/sdcard/test')  
with open('test.zip', 'wb') as folder1:  
 folder1.write(base64.b64decode(folder))  
  
# iOS  
# 需要安装 ifuse  
# > brew install ifuse 或者 > brew cask install osxfuse 或者 自行搜索安装方式  
  
driver.push\_file('/Documents/xx/element.png', source\_path='D:\works\element.png')  
  
# 向 App 沙盒中发送文件  
# iOS 8.3 之后需要应用开启 UIFileSharingEnabled 权限不然会报错  
bundleId = 'cn.xxx.xxx' # APP名字  
driver.push\_file('@{bundleId}/Documents/xx/element.png'.format(bundleId=bundleId), source\_path='D:\works\element.png')

**Pytest 与 Unittest 初始化上的区别**

很多人都使用过 unitest 先说一下 pytest 和 unitest 在 Hook method上的一些区别

**1.Pytest 与 unitest 类似，有些许区别，以下是 Pytest**

**class** TestExample:  
 def setup(self):  
 print("setup class:TestStuff")  
  
 def teardown(self):  
 print ("teardown class:TestStuff")  
  
 def setup\_class(cls):  
 print ("setup\_class class:%s" % cls.\_\_name\_\_)  
  
 def teardown\_class(cls):  
 print ("teardown\_class class:%s" % cls.\_\_name\_\_)  
  
 def setup\_method(self, method):  
 print ("setup\_method method:%s" % method.\_\_name\_\_)  
  
 def teardown\_method(self, method):  
 print ("teardown\_method method:%s" % method.\_\_name\_\_)

**2.使用 pytest.fixture()**

@pytest.fixture()  
def driver\_setup(request):  
 request.instance.Action = DriverClient().init\_driver('android')  
 def driver\_teardown():  
 request.instance.Action.quit()  
 request.addfinalizer(driver\_teardown)

**初始化实例**

**1.setup\_class 方式调用**

**class** Singleton(object):  
 """单例   
 ElementActions 为自己封装操作类"""  
 Action = None  
  
 def \_\_new\_\_(cls, \*args, \*\*kw):  
 **if** not hasattr(cls, '\_instance'):  
 desired\_caps={}  
 host = "http://localhost:4723/wd/hub"  
 driver = webdriver.Remote(host, desired\_caps)  
 Action = ElementActions(driver, desired\_caps)  
 orig = **super**(Singleton, cls)  
 cls.\_instance = orig.\_\_new\_\_(cls, \*args, \*\*kw)  
 cls.\_instance.Action = Action  
 **return** cls.\_instance  
  
**class** DriverClient(Singleton):  
 pass

测试用例中调用

**class** TestExample:  
 def setup\_class(cls):  
 cls.Action = DriverClient().Action  
  
 def teardown\_class(cls):  
 cls.Action.clear()  
  
  
 def test\_demo(self)  
 self.Action.driver.launch\_app()  
 self.Action.set\_text('123')

**2.pytest.fixture() 方式调用**

**class** DriverClient():  
  
 def init\_driver(self,device\_name):  
 desired\_caps={}  
 host = "http://localhost:4723/wd/hub"  
 driver = webdriver.Remote(host, desired\_caps)  
 Action = ElementActions(driver, desired\_caps)  
 **return** Action  
  
  
  
# 该函数需要放置在 conftest.py, pytest 运行时会自动拾取  
@pytest.fixture()  
def driver\_setup(request):  
 request.instance.Action = DriverClient().init\_driver()  
 def driver\_teardown():  
 request.instance.Action.clear()  
 request.addfinalizer(driver\_teardown)

测试用例中调用

#该装饰器会直接引入driver\_setup函数  
@pytest.mark.usefixtures('driver\_setup')  
**class** TestExample:  
  
 def test\_demo(self):  
 self.Action.driver.launch\_app()  
 self.Action.set\_text('123')

**Pytest 参数化方法**

**1.第一种方法 parametrize 装饰器参数化方法**

@pytest.mark.parametrize(('kewords'), [(u"小明"), (u"小红"), (u"小白")])  
def test\_kewords(self,kewords):  
 print(kewords)  
  
# 多个参数   
@pytest.mark.parametrize("test\_input,expected", [  
 ("3+5", 8),  
 ("2+4", 6),  
 ("6\*9", 42),  
])  
def test\_eval(test\_input, expected):  
 **assert** eval(test\_input) == expected

**2.第二种方法，使用 pytest hook 批量加参数化**

# conftest.py  
def pytest\_generate\_tests(metafunc):  
 """  
 使用 hook 给用例加加上参数  
 metafunc.cls.params 对应类中的 params 参数  
  
 """  
 **try**:  
 **if** metafunc.cls.params and metafunc.function.\_\_name\_\_ in metafunc.cls.params: ## 对应 TestClass params  
 funcarglist = metafunc.cls.params[metafunc.function.\_\_name\_\_]  
 argnames = list(funcarglist[0])  
 metafunc.parametrize(argnames, [[funcargs[name] **for** name in argnames] **for** funcargs in funcarglist])  
 except AttributeError:  
 pass  
  
# test\_demo.py  
**class** TestClass:  
 """  
 :params 对应 hook 中 metafunc.cls.params  
 """  
 # params = Parameterize('TestClass.yaml').getdata()  
  
 params = {  
 'test\_a': [{'a': 1, 'b': 2}, {'a': 1, 'b': 2}],  
 'test\_b': [{'a': 1, 'b': 2}, {'a': 1, 'b': 2}],  
 }  
 def test\_a(self, a, b):  
 **assert** a == b  
 def test\_b(self, a, b):  
 **assert** a == b

**Pytest 用例依赖关系**

使用 pytest-dependency 库可以创造依赖关系  
当上层用例没通过，后续依赖关系用例将直接跳过，可以跨 Class 类筛选  
如果需要跨.py 文件运行 需要将 site-packages/pytest\_dependency.py 文件的

**class** DependencyManager(object):  
 """Dependency manager, stores the results of tests.  
 """  
  
 ScopeCls = {'module':pytest.Module, 'session':pytest.Session}  
  
 @classmethod  
 def getManager(cls, item, scope='session'): # 这里修改成 session

如果

> pip install pytest-dependency

**class** TestExample(object):  
  
 @pytest.mark.dependency()  
 def test\_a(self):  
 **assert** False  
  
 @pytest.mark.dependency()  
 def test\_b(self):  
 **assert** False  
  
 @pytest.mark.dependency(depends=["TestExample::test\_a"])  
 def test\_c(self):  
 # TestExample::test\_a 没通过则不执行该条用例  
 # 可以跨 Class 筛选  
 print("Hello I am in test\_c")  
  
 @pytest.mark.dependency(depends=["TestExample::test\_a","TestExample::test\_b"])  
 def test\_d(self):  
 print("Hello I am in test\_d")

pytest -v test\_demo.py   
2 failed  
 - test\_1.py:6 TestExample.test\_a  
 - test\_1.py:10 TestExample.test\_b  
2 skipped

**Pytest 自定义标记，执行用例筛选作用**

**1.使用 *[@](https://testerhome.com/pytest.mark" \o "@pytest.mark)*[pytest.mark](https://testerhome.com/pytest.mark" \o "@pytest.mark) 模块给类或者函数加上标记，用于执行用例时进行筛选**

@pytest.mark.webtest  
def test\_webtest():  
 pass   
  
  
@pytest.mark.apitest  
**class** TestExample(object):  
 def test\_a(self):  
 pass  
  
 @pytest.mark.httptest  
 def test\_b(self):  
 pass

仅执行标记 webtest 的用例

pytest -v -m webtest  
  
Results (0.03s):  
 1 passed  
 2 deselected

执行标记多条用例

pytest -v -m "webtest or apitest"  
  
Results (0.05s):  
 3 passed

仅不执行标记 webtest 的用例

pytest -v -m "not webtest"  
  
Results (0.04s):  
 2 passed  
 1 deselected

不执行标记多条用例

pytest -v -m "not webtest and not apitest"  
  
Results (0.02s):  
 3 deselected

**2.根据 test 节点选择用例**

pytest -v Test\_example.py::TestClass::test\_a  
pytest -v Test\_example.py::TestClass  
pytest -v Test\_example.py Test\_example2.py

**3.使用 pytest hook 批量标记用例**

# conftet.py  
  
def pytest\_collection\_modifyitems(items):  
 """  
 获取每个函数名字，当用例中含有该字符则打上标记  
 """  
 **for** item in items:  
 **if** "http" in item.nodeid:  
 item.add\_marker(pytest.mark.http)  
 elif "api" in item.nodeid:  
 item.add\_marker(pytest.mark.api)

**class** TestExample(object):  
 def test\_api\_1(self):  
 pass  
  
 def test\_api\_2(self):  
 pass  
  
 def test\_http\_1(self):  
 pass  
  
 def test\_http\_2(self):  
 pass  
 def test\_demo(self):  
 pass

仅执行标记 api 的用例

pytest -v -m api  
Results (0.03s):  
 2 passed  
 3 deselected  
可以看到使用批量标记之后，测试用例中只执行了带有 api 的方法

**用例错误处理截图，app 日志等**

1.第一种使用 python 函数装饰器方法

def monitorapp(function):  
 """  
 用例装饰器，截图，日志，是否跳过等  
 获取系统log，Android logcat、ios 使用syslog  
 """  
  
 @wraps(function)  
 def wrapper(self, \*args, \*\*kwargs):  
 **try**:  
 allure.dynamic.description('用例开始时间:{}'.format(datetime.datetime.now()))  
 function(self, \*args, \*\*kwargs)  
 self.Action.driver.get\_log('logcat')  
 except Exception as E:  
 f = self.Action.driver.get\_screenshot\_as\_png()  
 allure.attach(f, '失败截图', allure.attachment\_type.PNG)  
 logcat = self.Action.driver.get\_log('logcat')  
 c = '\n'.join([i['message'] **for** i in logcat])  
 allure.attach(c, 'APPlog', allure.attachment\_type.TEXT)  
 raise E  
 **finally**:  
 **if** self.Action.get\_app\_pid() != self.Action.Apppid:  
 raise Exception('设备进程 ID 变化，可能发生崩溃')  
 **return** wrapper

2.第二种使用 pytest hook 方法 (与方法一选一)

@pytest.hookimpl(tryfirst=True, hookwrapper=True)  
def pytest\_runtest\_makereport(item, call):  
 Action = DriverClient().Action  
 outcome = yield  
 rep = outcome.get\_result()  
 **if** rep.when == "call" and rep.failed:  
 f = Action.driver.get\_screenshot\_as\_png()  
 allure.attach(f, '失败截图', allure.attachment\_type.PNG)  
 logcat = Action.driver.get\_log('logcat')  
 c = '\n'.join([i['message'] **for** i in logcat])  
 allure.attach(c, 'APPlog', allure.attachment\_type.TEXT)  
 **if** Action.get\_app\_pid() != Action.apppid:  
 raise Exception('设备进程 ID 变化，可能发生崩溃')

**Pytest 另一些 hook 的使用方法**

**1.自定义 Pytest 参数**

> pytest -s -all

# content of conftest.py  
def pytest\_addoption(parser):  
 """  
 自定义参数  
 """  
 parser.addoption("--all", action="store\_true",**default**="type1",help="run all combinations")  
  
def pytest\_generate\_tests(metafunc):  
 **if** 'param' in metafunc.fixturenames:  
 **if** metafunc.config.option.all: # 这里能获取到自定义参数   
 paramlist = [1,2,3]  
 **else**:  
 paramlist = [1,2,4]  
 metafunc.parametrize("param",paramlist) # 给用例加参数化  
  
# 怎么在测试用例中获取自定义参数呢  
# content of conftest.py  
def pytest\_addoption(parser):  
 """  
 自定义参数  
 """  
 parser.addoption("--cmdopt", action="store\_true",**default**="type1",help="run all combinations")  
  
  
@pytest.fixture  
def cmdopt(request):  
 **return** request.config.getoption("--cmdopt")  
  
  
# test\_sample.py 测试用例中使用  
def test\_sample(cmdopt):  
 **if** cmdopt == "type1":  
 print("first")  
 elif cmdopt == "type2":  
 print("second")  
 **assert** 1  
  
> pytest -q --cmdopt=type2  
second  
.  
1 passed in 0.09 seconds

**2.Pytest 过滤测试目录**

#过滤 pytest 需要执行的文件夹或者文件名字  
def pytest\_ignore\_collect(path,config):  
 **if** 'logcat' in path.dirname:  
 **return** True #返回 True 则该文件不执行

**Pytest 一些常用方法**

**Pytest 用例优先级（比如优先登录什么的）**

> pip install pytest-ordering

@pytest.mark.run(order=1)  
**class** TestExample:  
 def test\_a(self):

**Pytest 用例失败重试**

#原始方法  
pytet -s test\_demo.py  
pytet -s --lf test\_demo.py #第二次执行时，只会执行失败的用例  
pytet -s --ll test\_demo.py #第二次执行时，会执行所有用例，但会优先执行失败用例  
#使用第三方插件  
pip install pytest-rerunfailures #使用插件  
pytest --reruns 2 # 失败case重试两次

**Pytest 其他常用参数**

pytest --maxfail=10 #失败超过10次则停止运行  
pytest -x test\_demo.py #出现失败则停止

\*本文转自testerHome