自动化测试规划方案

# 接口自动化搭建

###### 前置条件

1. 执行环境：生产
2. 开发范围：核心接口用例(原生客户端)
3. 开发框架：Python + Pytest + Requests + allure
4. 开发工具：Pycharm
5. 代码管理库：Git

###### 框架选取原则

1. python：脚本开发及执行效率高，团队使用率较高而选择此后端语言。
2. pytest：易于维护测试用例组织，断言丰富，可扩展性强。
3. allure：可将批量数据执行结果生成报告，展示效果简约美观。
4. requests：请求接口简单高效，易于后期维护。

###### 目录结构

1. [config.py](https://config.py) （总配置文件）
2. [public.py](https://public.py)(公共方法集合)
3. [README.md](https://README.md)
4. requirements.txt
5. 生产环境接口用例集合文件夹
   1. 执行文件
   2. 方法文件
   3. 数据集合
   4. 业务文件夹-(一级业务)
   5. 业务文件夹-(二级业务)
6. 测试环境接口用例结合文件夹
   1. 执行文件
   2. 方法文件
   3. 数据集合
   4. 业务文件夹-(一级业务)
   5. 业务文件夹-(二级业务)

###### 接口用例编写规范

1. 参数化
2. @pytest.mark.parametrize
3. allure报告目录展示层级
4. @allure.eipc() 用例一级目录模块名称
5. @allure.feature() 用例二级目录模块名称
6. @allure.story() 用例第三级目录模块名称
7. @allure.title() 用例第四级标题名称
8. 断言
9. 响应结果最外层信息
10. 响应结果深层次信息校验
11. 动态参可视情况而定校验不允许为空
12. 数值及逻辑校验
13. 执行逻辑
14. 异常判断机制
15. 每环节及步骤进行log明细打印（无论成功与否）
16. 类命名
17. 将同一业务类型放入同一个类下
18. 命名以Test开头加业务名称英文，“\_”分割
19. 方法命名
20. 命名以test开头加用例标题英文，“\_”分割
21. 赋值命名
22. 英文名称，“\_”分割
23. 封装
24. 公共方法封装后存放在public中

# UI自动化搭建

###### 前置条件

1. 测试环境：生产
2. 开发范围：基于回归测试用例开发UI用例(业务为原生客户端)
3. 开发框架：python + pytest + uiautomator2 + allure + adb
4. 开发工具：Pycharm
5. 代码管理库：Git
6. 测试设备使用android真机设备【使用Windows系统连接】
7. 测试设备使用IOS真机设备【使用Mac系统连接】

###### 框架选取原则

1. python：脚本开发及执行效率高，团队使用率高情况而选择此后端语言。
2. pytest：易于维护测试用例组织，断言丰富，可扩展性强。
3. allure：可将批量数据执行结果生成报告。展示效果简约美观。
4. uiautomator2：
   1. 基于python与手机通信的库。
   2. 从环境配置而言，比appium更便捷，同时没有过多的第三方依赖。
   3. 从开发脚本效率而言，比appium更快构建代码。
   4. 从脚本执行而言，比appium执行效率更高。
5. ADB：连接安卓机与PC端命令行工具，可让设备进行通信.

###### 辅助测试工具安装方法

1. ATX（手机端应用，可与PC端通信，辅助PC拿到app前端元素定位）
   1. USB数据线连接电脑与测试机
   2. 打开终端cmd 输入adb devices
   3. 打开终端cmd 输入weditor命令
   4. 测试机出现弹窗提示是否安装ATX
   5. 点击安装按钮
   6. 测试机出现弹窗提示是否安装uiautomator server
   7. 点击安装按钮(安装成功手机页面展示atx app图标，无需对其做任何操作)
   8. cmd执行python -m weditor命令
   9. PC弹出浏览器即可看到手机屏幕，操作对应的元素可进行定位

⚠️：uiautomator server为一个服务，负责与移动设备通信，不会出现在测试机桌面上.

###### 架构目录（包/文件名称及描述）

1. cases（用例目录--存放计划执行的用例）

a. \_\_init\_\_.py（包初始化文件）

b. [home.py](https://home.py)（首页模块用例集）

c. [service.py](https://service.py)（服务模块用例集）

d. [mine.py](https://mine.py)（我的模块用例集）

e. [wealth.py](https://wealth.py)（财富模块用例集）

1. devices（设备目录--存放手机设备信息）

a. [decices\_data.py](https://decices_data.py)

1. Picture（图片目录--存放用例执行后的截图）

a. Pass文件夹（存放用例执行成功后的截图）

b. Fail文件夹（存放用例执行失败后的截图）

1. page\_oblect（页面目录--存放源生客户端各个页面的逻辑代码）

a. home文件夹（存放首页各元素逻辑代码文件）

b. service文件夹（存放服务各元素逻辑代码文件）

c. mine文件夹（存放我的各元素逻辑代码文件）

d. wealth文件夹（存放财富各元素逻辑代码文件）

1. public（配置目录--存放一些配置类文件）

a. config.py（总配置文件--定义了系统、测试环境、异常是否退出）

b. log.py（日志文件--方便将自定义的日志内容加进allure报告中展示）

c. base\_page.py(程序driver初始化、driver操作APP元素）

1. report（报告目录--存放测试报告文件）
2. tools（工具目录--存放工具类脚本）

a. screenshot.py（截图工具脚本）

b. threading.py（异步装饰器脚本）

1. utils（公共目录--存放公共方法类文件）

a. base.py（公共方法脚本）

1. main文件（执行文件，便于一次性执行ui自动化全部或多个测试用例）
2. read文件（记录一些方法、命令描述）
3. test文件（便于局部调试代码）

###### UI用例编写规范：

1. 参数化
   1. @pytest.mark.parametrize
2. allure报告展示
   1. *同理接口allure规范*
3. 断言
   1. 组件元素检验
   2. 文本检验
   3. 数值运算
4. 执行逻辑
   1. 异常判断机制-异常截图
   2. 用例执行完毕截图
5. 类命名
   1. 将同一业务类型编入同一个类
   2. 命名以Test开头加业务名称英文，“\_”分割
6. 方法命名
   1. 命名以test开头加用例标题英文，“\_”分割
7. 赋值命名
   1. 英文名称，“\_”分割
8. 封装
   1. 代码编写要遵循PO模型，将界面业务功能与元素分开。
   2. 公共方法封装后存放在utils中

###### PO模型封装及使用

1. 通过python第三方库uiautomator2实现对联通APP驱动控制（driver)
2. 实现对APP界面截图，用例运行成功后截图放到Pass文件夹，用例运行失败后截图放到Fail文件夹，图片标题标明截图时间及用例名称
3. 通过不同的定位方式（text、xpath、resourceId、description、className、textStartsWith、textContains）获取APP界面元素
4. 实现对元素的点击、输入、长按操作并增加log日志明细
5. 实现对界面的滑动、拖动操作并加log日志
6. 封装检查校验界面元素是否存在的判断
7. 封装获取元素详细信息及判断元素是否为可点击状态
8. 获取设备详细信息、可对设备进行操作（获取屏幕状态、按下home键、按下返回键等）

# 脚本开发规范

###### 脚本开发目的

1. 基于业务及回归测试用例，通过脚本开发及批量执行的方式对用例进行实现，起到高效精准回归目的

###### 脚本验证点校验

1. 基于回归测试用例断言所有可校验的验证点
2. 校验返回值中关键字段及文本内容
3. 基于UI用例可对当前页面组件进行非空校验

###### 脚本代码格式

1. 规范代码缩进、空格、换行等，使代码具备良好可读性及高度维护性和兼容性。

###### 脚本注释编写

1. 在新增代码中明确添加注释(含代码功能/目的/编写人等)

###### 脚本方法设计

1. 基于清晰、明了、简洁、易理解原则进行方法设计
2. 确保方法兼容性，不影响其余相关功能
3. 避免方法中代码耦合及重复

###### 脚本异常处理

1. 代码需添加必要异常机制校验，使其合理输出异常情况报错

###### 脚本代码兼容

1. 支持使用代码(用例)进行复用，减少代码冗余

###### 脚本代码评审

1. 规范代码格式，已便评审时按照规定检查点进行review。严格按照编写及执行规范进行评审，未通过需打回重写后再次评审，直至评审通过。可提交代码至主分支。

# 资源申请

1. Git代码管理库

# 执行节点

1. 生产版本验证当晚