# 接口自动化测试方案

## 1、引言

### 1.1 项目情况

|  |  |
| --- | --- |
| 项目名称 | 项目管理平台 |
| 项目版本 | V1.0 |
| 项目经理 | XX |
| 测试人员 | XXXXX，XXX |
| 所属部门 | XX |
| 备注 |  |

### 1.2 文档目的

　　本文档主要用于指导《项目管理平台》项目常用接口自动化测试工作的开展。本文档的主要目的在于提供项目接口自动化测试的技术方案、实施方案和计划方案等。

## 2、接口自动化实施目标

### 2.1 实施原则

　　《项目管理平台》项目采用接口自动化测试，主要目的是为了应对迭代版本测试过程中的重复工作任务，以期达到效果如下：

* **降低测试成本**
* **提高测试效率**
* **更频繁地执行覆盖重要接口**
* **提供更高的准确性和一致性**
* **节约时间成本**

　　虽然能达到上述预期效果，但实际实施过程中需要注意的是，接口自动化的高效应用，对于被测系统有着更高的要求，也需要遵循合理的方法流程，现总结如下：

接口自动化的实施应该被用于解决测试过程中高重复性的工作，很大一部分是用于回归测试老的功能接口，否则其本身工作量投入会大于其收益，所以不能盲目对所有接口或功能追求自动化。

对于提测版本，自身稳定性需要有一定程度的保障。过于频繁的接口变动，会加大后续接口自动化的实施难度，增加自动化脚本维护地成本。

接口自动化的整体实现应采用分布进行，测试过程中优先覆盖功能稳定且比较重要的接口，进而逐步扩展到整体项目的接口回归。

接口自动化测试是一个长期的过程，随着项目版本的不停迭代优化，项目本身的接口也会不断优化或新开发，所以后续自动化测试脚本的代码维护和调优也具有可观的工作量。

### 2.2 接口自动化测试范围

　　系统范围：

|  |  |  |
| --- | --- | --- |
| 自动化实施阶段 | 被测模块 | 功能接口范围 |
| 第一阶段 | 个人主页（申请、审批等） 我的项目（项目创建修改，任务创建分配等） | 被测模块涉及到的接口 |
| 第二阶段 | 项目管理平台所有模块 | 当前项目所有接口 |

### 2.3 接口自动化测试任务

**制定测试方案**

脚本编码前，需要对项目有一个整体把握，合理预估接口数量与复杂度。结合版本迭代时间，预估自动化脚本开发时间，并制定出相应的接口自动化测试方案。

**提取分析测试点**

根据前面写好的接口自动化测试范围，分析每个接口的测试点，包含请求方式，传入参数，请求头，返回状态，返回数据等。这个过程中，需要和相对应的开发对接清楚在测试范围内的接口的相关信息，并提前在postman中逐一确认调通，必要时生成相应的测试文档或编写进入测试用例中。

**搭建测试框架**

　　此次接口自动化测试框架采用的是以Python语言为脚本开发语言，pytest+allure+jenkins测试框架。目的希望达成可配置，能自动运行脚本，自动生成测试报告并将生成的测试报告发送到指定邮件。

**编写脚本代码**

　　脚本首次实现不需要覆盖到每个接口。先预计挑选几个重要接口进行覆盖测试，等整体测试框架搭建好后，整体流程确认调通无误后，再后续维护完善脚本，覆盖更多的功能接口。

**持续集成**

　　同上，初次脚本代码完成后，需要对现有自动化脚本进行升级持续集成开发，不断完成尚未覆盖到的接口，将这些接口加入到自动化测试的范围内，使得整体自动化程度进一步加深，更大程度上节约人力和时间成本。

**脚本维护**

　　脚本维护是在整体自动化脚本阶段性完成后，将现有生成的交付物归档整理好给相应的负责人管理，并进行阶段性的更新整理维护。包含项目日常版本迭代维护过程中对接口有改动的部分，和后续新加入接口得自动化覆盖等。

## 3、接口自动化技术选型

### 3.1 整体体系

　　本次自动化实现主要是以接口自动化的形式来开展。整个自动化脚本以 Python3.X 中 requests 库为核心机制，以 pytest 作为测试组织，以 allure-report生成最终测试报告，Jenkins 实现持续集成，通过SMTP发送各模块通过率邮件，并选取 Python3.X 作为编程语言实现。

### 3.2 核心技术

#### 3.2.1 接口自动化执行库--Requests

　　首先，Requests 是使用 Python 语言编写，基于 urllib，采用 Apache2 licensed 许可证的 HTTP 库。它比一般的 urllib 等库更加方便、简洁，可以节约我们大量的工作，完全满足HTTP 测试需求，总结为一句话：Requests 是 Python 实现的简单易用的 HTTP 库。其次，Requests 库安装和导入非常方便。

　　pip3 install requests  ## 安装 Requests 库

　　import requests ## 导入 Requests 库到项目中

　　我们可以使用该库实现以下各种方法：

　　requests.get("https://url.cn")              # GET请求

　　requests.post("http://url.cn")              # POST请求

　　requests.put("http://url.cn")               # PUT请求

　　requests.delete("http://url.cn")            # DELETE请求

　　requests.patch("http://url.cn")              # PATCH请求

#### 3.2.2 测试框架主体--pytest

　　pytest 是 Python 的一款开源测试框架。

##### 3.2.2.1主要特点

* 简单灵活，文档丰富；
* 能够支持单元测试、功能测试、接口自动化测试等；
* 具有众多第三方插件，且可以自定义扩展；
* 支持参数化，可控制测试用例的粒度；
* 可以很好的和 CI 工具结合，例如 Jenkins 。

##### 3.2.2.2使用参考

###### 命名约束

测试文件以"test\_"开头/以"\_test"结尾

测试类以"Test"开头，并且不能带有 \_\_init\_\_ 方法

测试函数以"test\_"开头

默认情况下，pytest 查找当前目录下所有以 test 开始或结尾的 Python 脚本，并执行文件内的所有以 test 开始的函数和方法。

###### 运行

在终端按需输入以下命令来运行 case：

运行所有 case ：pytest

运行指定 case ：pytest testCase/test\_demo1.py

运行关键字匹配的 case ：pytest -k “表达式”

运行标记匹配的 case ：pytest -m "标记"

运行后控制台展示详细信息：pytest -v

###### 断言assert

pytest 中可以使用 Python 原生的断言语句 assert 来进行验证。

常用：

assert xx：判断 xx 为真

assert not xx：判断 xx 不为真

assert a in b：判断 b 包含 a

assert a == b：判断 a 等于 b

assert a !=b：判断 a 不等于 b

可以指定断言失败的返回信息，举例如下：

assert response["result\_code"] == expected["response"]["result\_code"], "实际的返回结果是：{}".format(response["result\_code"])

###### fixture

fixture 是 pytest 特有功能，目的是提供一种手段去运行最基本的、重复的测试内容，是将测试前后的预备工作、清理工作的代码分离出核心测试逻辑的一种机制。通常用来对测试方法、测试函数、测试类和整个测试文件进行初始化或还原测试环境。

@pytest.fixture(scope="function", params=None, autouse=False, ids=None, name=None)

scope：作用域，（

function：方法，默认为function

class：类

module：.py文件

session：会话）

params：可选的参数列表，可使多个参数调用fixture功能

autouse：是否默认调用该fixture，默认为False

ids：每个字符串id的列表

name：装饰器名字

1. **fixture 作为函数参数**

可以通过使用 @pytest.fixture 注册成为一个 fixture 函数，来为测试方法提供一个 fixture 对象。

如下图所示，connect\_mysql\_uds 函数被注册成了一个 fixture 函数，在 test\_query\_couplist 方法中可以直接被作为参数使用。

1. **在 conftest.py 中管理 fixture**

如果多个文件都需要用到这个fixture函数，则可以将该函数放到conftest.py文件中管理，所需的 fixture 对象会自动被 pytest 发现。

1. **执行顺序**

较高范围的 fixture（例如session会话级）比低范围的 fixture（例如function函数级或class类级）优先执行。

相同范围的 fixture 对象的按引入的顺序或依赖关系按顺序调用。

1. **yield**

yield 语句之后的所有代码都视为 teardown 代码。

举例，如下图所示，整个会话结束后，会打印出”关闭数据库”的语句。

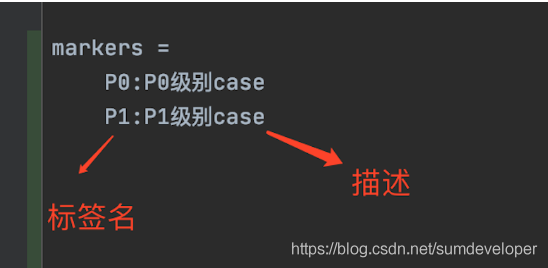
###### mark

1. **标记和分类用例**

@pytest.mark.level

用法步骤:

* 先在 pytest.ini 文件中对标签进行注册



* 对用例进行标记

@pytest.mark.P0

* 运行指定标记的用例

在终端输入： pytest -m "P0"

1. **标记跳过用例**

无条件跳过：@pytest.mark.skip(reason="跳过原因描述")

在指定条件下跳过：@pytest.mark.skipif(condition, reason="跳过原因描述")

1. **标记用例为预期失败**

@pytest.mark.xfail(reason="预期失败描述")

对于功能未完成或已知有问题的用例，可以用 xfail 标记，这样即使断言失败，也不会影响其他测试用例的的执行。

执行 case 后，控制台中显示：

XFAIL：表示预期失败，实际也失败

XPASS：表示预期失败，实际运行没有失败

1. **参数化**

我们进行接口测试时，往往很多时候操作步骤都是一样的，只有数据不一样，这时候就可以使用参数化功能，来减少代码冗余。

@pytest.mark.parametrize(argnames, argvalues, ids=None)

argnames：参数名称，字符串格式

argvalues：参数值列表，列表格式

ids：用例的 ID ，字符串列表格式

举例，如下图所示，装饰器会自动对 list\_params 解包并赋值给装饰器的第一参数。装饰器的第一个参数中逗号分隔的变量可以作为测试方法的参数，在测试方法内就可以直接获取这些变量的值。

#### 3.2.3 测试报告生成--Allure

Allure 是一种灵活、轻量级、多语言的测试报告工具，以简洁精美的 web 报告形式展示。

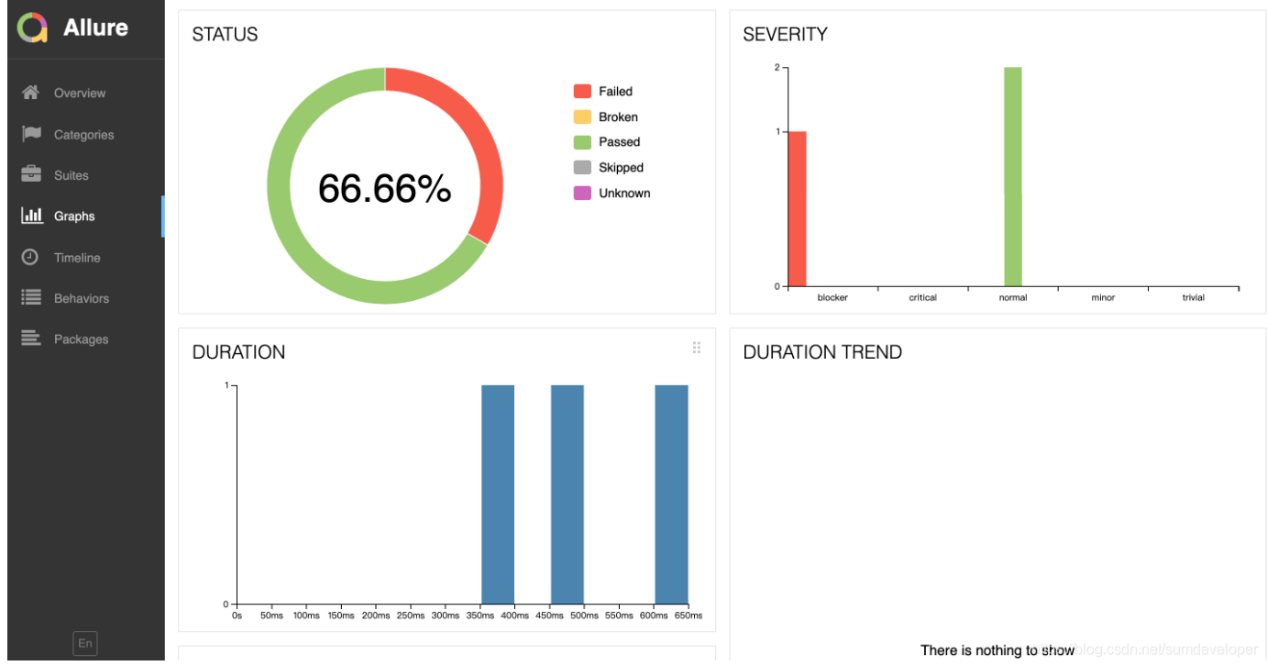
Python 中有一个第三方库：allure-pytest，用于连接 pytest 和 allure ，使它们可以更好的配合使用。

Allure的常用用法：

|  |  |
| --- | --- |
| 使用方法 | 说明 |
| @allure.epic() | 描述史诗，即未拆解的story |
| @allure.feature() | 描述功能模块 |
| @allure.story() | 描述用户故事 |
| @allure.title() | 描述用例标题 |
| @allure.step() | 描述用例步骤 |
| @allure.severity() | 描述用例等级：blocker，critical，normal，minor，trivial |

Graphs

该页面展示本次执行结果的统计信息，比如执行结果成功失败比例、测试用例等级分布、用例执行时间等。



Behaviors

该页面根据代码中设定的 @allure.feature()、@allure.story() 等来展示用例执行的结果，这里可以看到用例执行的详细信息。

#### 3.2.4 持续集成机制--Jenkins

这里我们脚本持续集成选择  Jenkins 来实现，通过使用 Jenkins，我们能够实现脚本自动化执行，包含定时执行自动化测试脚本，和自动化脚本运行后的测试报告发送到指定的邮箱中。

### 3.3 框架思想

　　整个接口自动化测试脚本采用面向对象的封装思想，尽量将一些可配置的模块单独提取出来，便于后续操作配置，使得整体项目更加灵活多变，便于后续地迭代维护和二次开发。封装思想主要体现在测试环境可配置，测试用例可导入，测试数据和脚本分离，文件路径采用相对路径表示等，具体我们会在实际编码分层中得到体现。

## 4、测试环境需求

### 4.1 硬件环境

目前暂未涉及到性能相关或者需要分布式执行的内容，因此对硬件要求不是很高，日常办公硬件即可。如果后续有涉及性能相关内容，硬件环境需要再另外的性能测试方案中体现。

|  |  |
| --- | --- |
| 硬件相关 | 备注 |
| window执行机/计算云 | 自动化工厂连跑 |
| 服务器 | 测试环境部署 |

### 4.2 软件环境

|  |  |  |
| --- | --- | --- |
| 软件相关 | 版本号 | 备注 |
| Python | v3.6 | 脚本编码语言为 Python3.x |
| PyCharm | v2020.2 | IDE |
| Git | / | 代码管理 |
| Github | / | 代码管理 |
| Jenkins | / | 持续集成 |

## 5、人员进度安排

### 5.1 职责分配

|  |  |  |
| --- | --- | --- |
| 组别/人员 | 职责 | 备注 |
| 王东 | 方案编写&框架搭建 |  |
|  |  |  |

### 5.2 进度安排

|  |  |  |  |
| --- | --- | --- | --- |
| 测试任务 | 负责人 | 开始时间 | 备注 |
| 自动化测试方案制定 | 王东 |  |  |
| 接口用例编写 | 王东 |  |  |
| 自动化测试环境搭建 |  |  |  |
| 自动化测试框架搭建 | 王东 |  |  |
| 自动化脚本代码编写 |  |  |  |
| 持续集成实现 | 王东 |  |  |
| 测试报告输出 | 王东 |  |  |
| 脚本二次维护开发 |  |  |  |

### 5.3 交付物管理

|  |  |  |
| --- | --- | --- |
| 交付物 | 负责人 | 备注 |
| 《自动化测试方案》 | 王东 |  |
| 自动化框架 | 王东 |  |
| 自动化脚本代码 |  |  |
| 测试执行报告 | 王东 |  |