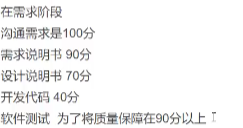
# 软件测试需要掌握

1. 测试思维（测试理论基础及对应的实践）
   1. 软件测试的目的：为了保障软件的质量
      1. 不是提升质量
      2. 不是找BUG，在实际测试过程中，BUG只是测试的产出物
2. 自动化测试技术
   1. Postman
   2. Jmeter
   3. Soapui
   4. robotFrameWork
   5. Katalon
   6. Appium
   7. Airtest

# Python

## Selenium

### 理论

Selenium：

本身是Firefox下的一个插件，主要用于火狐浏览器的UI操作录制。可以转换为Java、Ruby、js语言的内容。然后被Google关注，推出Chrome浏览器所支持的浏览器驱动：ChromeDriver

核心是基于js core来实现的

Selenium grid：传统的自动化测试框架都是基于代码的线性运行来实现的。对于 更为复杂、耗费时间成本更高的自动化测试，就会应用更为高级的方式来实现。例如：多用例的并发，测试框架分布式部署。Grid就是用于实现测试框架的分布式部署，基于M/S形态的架构部署，主从节点。主要通过主节点下发任务，从节点接收并执行任务，将结果返回至主节点进行记录。

Webdriver：

1. 不同的浏览器对应不同的webdriver
2. 不同的版本对应不同版本的webdriver，除了Chrome浏览器以为，其余浏览器对应的webdriver都可以在selenium官网了解。
3. 保存在python安装根路径即可
4. 浏览器安装在默认路径即可
5. Chromedriver本身是一个服务，在selenium自动化测试中，所有的代码都是不与浏览器交互的，其实都是只与webdriver服务进行交互，包含各类的指令数据和返回信息

### **八大元素定位法则**

**Close与quit区别**

Close关闭当前标签页

Quit关闭浏览器，释放进程

#### 八大元素定位（locator.py）

所有的ui层的自动化都是基于元素定位来实现的。所有被操作的元素都是WebElement对象

元素 = HTML标签

异常：Element not interactable 定位的元素无法进行交互

1. Id：基于元素属性中id的值来进行定位，类似于身份证上的身份证号码，不出意外都不会重复
2. Name：基于元素属性中name的值来进行定位，类似于身份证上的身份证名称，很可能会出现重名
3. Link text：主要用于超链接进行定位
4. Partial link text：link text的模糊查询版本，类似于数据库中的like%，当模糊查询匹配到多个符合条件的元素，选取第一个；当需要获取多个元素时用elements，使用for循环遍历结果集
5. Classname：基于元素样式进行定位，非常容易用到重复的
6. Tagname：基于标签名称来进行定位，在重复度最高只有在定位后需要二次筛选时才会使用
7. Cssselector：应用相对较多的一种行为，最初IE浏览器不支持Xpath，完全基于class属性进行定位
8. Xpath：目前应用最多的一种行为，基于页面结构进行定位。绝对路径：从HTML根路径下一层一层往下数，找到相对应的层级，从而找到元素；相对路径：基于匹配制度查找元素，依照xpath语法结构来定

例：//\*[@id=”kw”]

// 表示从根目录下开始查找元素

\* 任意元素

[] 表示筛选条件（查找条件）

@ 表示基于属性来筛选，例：@id=”kw”表示基于id属性值为kw的条件进行筛选

确认xpath路径是否正确：

1. 在开发者工具elements页面使用Ctrl+f查找，进行判断
2. 在console中输入$x()进行校验

基于text来定位元素：

在[]中添加text()=”文本内容”进行查找，例：//a[text()=’登录’]

基于value定位元素：

//input[@value=’百度一下’]

定位元素无法直接定位时，可以定位子级元素返回父级来获取元素

//input[contains(@id,’kw’)]

Contains:表示进一步查找，匹配项模糊查找

//input[contains(text(),’搜索’)]

### **Selenium三大等待（TestForWebUI）**

#### **等待的作用**

在系统的功能运行过程中，所有的内容是需要一定的时间来实现展示的。而时间耗费的长短与网络速度，与系统的框架设定，与接口的执行复杂程度

在自动化中，我们做ui测试，主要是基于系统的流程来实现自动化，往往会有下一步是需要上一步的一些特定的条件才可以继续执行的。在下一步准备要执行时上一步必须已经完成执行，且生成有实际结果。

考虑到运行的时间问题，以及流程的步骤问题，所以我们在实际执行自动化测试的时候，是需要设置有一个缓冲时间的。如果不预留缓冲时间，则运行自动化是非常容易出错的。同时，通过显式等待还可以设置有自动化测试的断言内容

#### **三大等待**

强制等待：通过导入time中的sleep，等待X秒之后再继续执行后续代码，无所谓当前处于什么阶段，执行了什么操作，后续是否已经准备就绪，反正就是要等X秒，一般都是在刚开始学习自动化的过程中进行使用，基本不在实际情况下使用

缺点：无法精准的把握等待时间（无法判断是否达到可以下一步进行操作/无法判定页面是否加载完成），如果一直使用强制等待，会严重影响整个自动化运行的效率，因为会浪费大量的时间

优点：简单使用，一般在调试的时候有奇效

隐式等待：设置一个隐形的等待，设置最长的等待时间，如果在这个时间内完成了页面的内容全部加载，则进行下一步，否则一直等待时间结束，再进行下一步

缺点：必须要页面全部加载完成才可以进行下一步，有些时候，页面中特定的元素已经被加载出来了，但是页面本身还是没有加载完。应用度上不是太灵活

优点：对整个webdriver的周期都能够起到作用，所以只需要设置一次

显式等待：专门用于对指定的条件进行等待。在设置的最大时长内，依照查找的时间频率来进行进行搜索，查找指定的对象，until表示如果找到，则继续下一步，否则报出异常（NoSuchElementException；TimeOut异常）Until\_Not()与until相反

优点：精确对某个特定条件进行等待，不会浪费多余的任何时间在等待上。如果条件成立，立即进入下一步；如果不成立则抛出异常

缺点：应用上而言，相较于其他两种等待较为复杂

#### **自动化中应该注意避免的坑**

1. 页面元素定位不到，基本都是因为没有添加等待时间或者元素定位的方式有误造成的

id/name/class/xpath/cssselector单纯的copy很可能会出错，需要通过手写xpath来进行定位，再经过验证确认元素定位的正确性

1. Iframe和handles的操作：切换Iframe用以操作Iframe的元素，句柄如果不进行切换，则永远在操作第一个句柄（标签页），如果需要操作其他的标签页，则需要切换句柄
2. 自动化测试执行完成的标志就是浏览器的退出。在实际的自动化中是一定需要断言的。

### **Yaml（data\_test\_yaml/yaml\_demo）**

#### **Yaml介绍**

Yaml介绍：在自动化测试领域，数据是需要与代码分离的，当数据与代码分离之后，如果出现有数据的变动，只需要修改数据文件，不需要修改代码；基于数据来驱动自动化测试的内容

数据驱动：Excel、Yaml一般选择数据驱动类型都是基于测试框架来定义的，也是基于企业实际框架的应用来决定的

环境搭建：

1. PyYaml安装：pip install PyYaml
2. Pycharm导入
3. 创建Yaml文件:是一种置标语言，xml
   1. 缩进来进行管理
   2. 能够实现各类型数据类型的展示

Yaml使用：（case\_demo/cases.py）

可以完美结合DDT来进行使用

### **关键字驱动**

什么是自动化：

自动化是将手工执行的测试内容，转换为机器可以识别的语言。交由机器来完成手工的执行，一般用于流程的校验和回归时进行

#### **关键字驱动介绍及作用**

RF（RobotFrameWork）基于表格编程来实现的关键字驱动测试框架，通过一个特定的关键词调用指定的对应事务操作

#查找一个指定的元素，对其进行输入操作

find\_element\_by\_xpath(xpath).send\_keys(“123456”)

当代码的流程过长的时候

1. 代码冗余非常高
2. 代码的维护性非常差
3. 代码阅读性差

#### **Selenium+关键字驱动实现自动化（test\_selenium/KeywordDemo.py）**

输入、点击、查找元素、启动浏览器......全部做一次二次封装

### **Iframe与句柄**

句柄：不同的标签页代表着不同的句柄，在程序世界里，标签页等于句柄；

## UnitTest（unit\_demo）

### **UnitTest基本应用规则**

UnitTest：在Python自动化领域主要流行的测试框架是：UnitTest与PyTest,UnitTest本身是是单元测试框架，截止到目前，已经可以基于框架来实现Selenium、Appium、Requests等实现UI自动化与接口自动化

四大特点：

1. TestCase：测试用例，所有的用例都是直接继承于UnitTest.Testcase类
2. TestFixture：SetUp和TearDown，作为前置条件和后置条件
3. TestSuite与TestRunner：测试套件与测试运行器，可以精细化管理测试用例以及生成测试报告
4. 断言机制：在TnitTest中封装好了成熟断言，可以直接通过self.来调用所有已封装好的断言函数

### **UnitTest搭建基本应用规则与测试用例编写**

环境搭建：Python安装时就已经默认封装好了UnitTest框架，调用框架的时候只需要import即可

UnitTest语法规则：

1. UnitTest中，测试用例定义都是以test\_开头
2. 用例排序与写的顺序无关，排序是遵循A-Z,a-z,0-9
3. cls是全局变量，self是类变量

### **UnitTest+数据驱动实现数据分离**

数据驱动：分割代码与数据，在UnitTest中通过DDT实现，DDT全程是：data driver test安装：pip install ddt

可以直接读取yaml文件

### **UnitTest断言**

断言：自动化测试主要用于对流程来进行测试，断言是自动化测试中的预期结果与实际结果的对比

|  |  |  |
| --- | --- | --- |
| 方法 | 检查 | 版本 |
| assertEqual(a,b) | a==b |  |
| assertNotEqual(a,b) | a != b |  |
| assertTrue(x) | bool(x) is True |  |
| assertFalse(x) | Bool(x) is False |  |
| assertIs(a,b) | a is b | 3.1 |
| assertIsNot(a,b) | a is not b | 3.1 |
| assertIsNone(x) | x is None | 3.1 |
| assertIsNotNone(x) | x is Not None | 3.1 |
| assertIn(a,b) | a in b | 3.1 |
| assertNotIn(a,b) | a not in b | 3.1 |
| assertIsInstance(a,b) | Isinstance(a,b) | 3.2 |
| assertNotIsInstance(a,b) | Not isinstance(a,b) | 3.2 |

### **UnitTest中的skip的使用**

跳过测试用例的执行，是通过装饰器执行的

### **UnitTest测试套件的使用**

Suite：必须新建一个类对象存放套件，直接在unittest类中运行无法生效

### **UnitTest生成测试报告**

HTMLTestRunner:

1. 环境搭建：
2. 导入环境
   1. 下载HTMLTestRunner.py，导入到python的lib/文件夹中
   2. 修改部分源码，应用语言python3中：
      1. 第94行，将import StringIO修改成import io
      2. 第539行，将self.outputBuffer = StringIO.StringIO()修改成self.outputBuffer = io.StringIO()
      3. 第642行，将if not rmap.has\_key(cls):修改成if not cls in rmap:
      4. 第766行，将uo = o.decode(‘latin-1’)修改成uo = e
      5. 第772行，将ue = e.decode(‘latin-1’)修改成ue = e
      6. 第631行，将print >> sys.stderr,’\nTime Elapsed:%s’% (self.stopTime-self.startTime)修改成print(sys.stderr,’\nTime Elapsed:%s’ % (self.stopTime-self.startTime))
3. 导包：直接import即可

## 自动发邮件（SMTP）

SMPT：通过SMPT来关联其他邮箱，在python中有一个已经被保留的，smtplib是专门用于处理邮件的库，mail库处理邮件内容的库

自动化邮件发送实操案例：

1. 要处理邮件的主题、寄件人、收件人、邮件正文、附近
2. 邮件正文：文本信息、HTML信息、图片内容等等
3. 附件内容：TXT、jpg、PDF、XLS、doc等等

发送邮件附件的流程定义：

1. 读取附件
2. 添加附件
3. 与邮件一起发送

### **自动化测试框架设计思路**

1. 自动化测试是唯一的一种可以在学习过程中直接看到成效的技能。自动化测试是从中级开始的
2. 性能，就是jmeter与loadrunner，这些只是实现性能测试的工具，性能讲究的是系统的效率。性能测试实际上就是对系统效率的测试，校验系统服务端、框架、通信、底层、中间件等等。市场需要的主流是自动化测试
3. 底层自动化：运行现成的自动化测试系统，设计测试用例，使用一个成熟的工具的人
4. 正常自动化：编写自动化相关内容，进行维护更新，进行执行自动化测试
   1. 源码实现关键字的驱动，进行文件读取，实现数据驱动，通过可视化界面上传文件，发出运行指令让服务端运行已上传的用例文件，进行自动化脚本的执行，运行过程中记录用例的成功和失败状态，产出测试结果生成测试报告，基于邮件的形式将报告发送至相关团队成员。关联代码库，发布新的版本时自动化部署，运行常规测试，产出报告邮件发送

### **自动化测试框架设计**

POM测试框架

1. Selenium概述

Selenium是目前业内最为核心的页面自动化测试的实现技术。全程是由JS来实现的浏览器的交互。搭配到webdriver来实现完整的自动化ui测试的效果

1. 测试框架设计模式介绍

常规的自动化测试的技术体系，测试框架是目前企业级应用最核心的形态。

主体的设计模式分为两种：

1. 关键字驱动
2. POM模式
3. 关键字驱动设计模式介绍

是所有测试框架的核心基础

适应场景：对于有多个项目在同时进行、公司内部是基于频繁的项目研发来实现营收的体系化。

关键字驱动是唯一一种可以以一套框架来适配多种不同类型的项目的形态但是对于所有的被测项目而言，无法准确评估覆盖率

1. POM介绍

POM是目前业内公认最佳的一种设计模式。是专门用于对应制定系统来量身打造的一套测试框架。

全程叫做页面对象模型，是将系统以页面来进行区分

举例：

实现一个用户信息修改的流程

登录——进入个人中心——修改个人信息并保存——再重新查看个人信息详情

从POM角度：

1. 进入登录页面，实现页面的核心流程
2. 进入个人信息页面，实现修改个人信息的操作
3. 进入个人信息页面，实现查看个人信息的操作
4. 校验修改结果是否成功

在POM体系下，全程都是基于页面来考虑这个流程的执行惯性，中间会关联多少页面，每个页面分别会执行哪些内容。

1. POM结构设计与编码实现
2. 工程结果形态：
   * 1. 代码与数据分离
     2. 逻辑代码与测试代码分离
3. POM结构
   * 1. 基类：逻辑代码的部分，主要生成一系列在页面对象中可以被调用的函数。本身作为一个工具库的存在
     2. \*页面对象类：逻辑代码的部分。结合实际的项目，提取的所有可被自动化执行的页面。类中包含页面的核心元素与页面的核心流程
     3. 测试用例类：测试代码部分。用于拼接各类页面对象，实现最终的测试流程
     4. 测试数据类：用于提取所有在实际测试过程中需要应用的数据内容。
4. POM+UnitTest实现自动化测试
   1. 简单的POM的体系设计：请看源码（dom）
   2. 自动化测试的实现是要关联到核心业务的，没有关联业务的自动化是无法实现落地的
5. 结合数据驱动分离测试数据
   1. Ddt + yaml 或ddt + data的形态来实现数据驱动

### **鼠标操作**

### **窗口切换**

### **Iframe操作**

## Python操作Excel