# 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

### **UnitTest基本应用**

### **UnitTest搭建基本应用规则**

### **UnitTest+数据驱动实现数据分离**

### **UnitTest断言**

### **UnitTest中的skip的使用**

### **UnitTest测试套件的使用**

### **UnitTest生成测试报告**

### **UnitTest自动化测试**

### **鼠标操作**

### **窗口切换**

### **Iframe操作**

## Python操作Excel