|  |
| --- |
|  |
| Robot Framework + Appium Guide for Android Testing Automation |
|  |

|  |
| --- |
| KPMG Information Technology  Services (Nanjing) Limited  January 2019 |
| This report contains 10 pages |
|  |
| Appium Guide.docx |

KPMG Information Technology Services (Nanjing) Limited, a wholly foreign owned enterprise in China and a member firm of the KPMG network of independent member firms affiliated with KPMG International Cooperative (“KPMG International”), a Swiss entity. All rights reserved. Printed in China.

The KPMG name and logo are registered trademarks or trademarks of KPMG International.

**Document classification: KPMG Confidential**

Document review and approval

Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Date | Revision |
| 0.1 | Tian Zheng | 11/01/2019 |  |
| 0.2 | Tian Zheng | 18/01/2019 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

This document has been reviewed by

|  |  |  |
| --- | --- | --- |
|  | Reviewer | Date reviewed |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

This document has been approved by

|  |  |  |  |
| --- | --- | --- | --- |
|  | Subject matter experts | |  |
|  | Name | Signature | Date reviewed |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

1 Installation 1

2 AppiumLibrary Hands-on Guide 4

2.1 Open Application 4

2.2 Click Element 6

2.3 Other Keywords 8

2.4 Alternative way to inspect elements 9

3 Development Guide 12

3.1 Test Script Development 12

3.1.1 Debug Library Installation 12

3.1.2 Introducing RF Interactive Shell 12

3.2 Develop new keyword 14

3.2.1 Preliminaries 14

3.2.2 Appium Python Development 15

4 iOS Testing Guide 17

4.1 Installation 17

4.2 Differences from Android testing 17

A Appendix: Useful Documents 19

A.1 How to Write Effective XPath Selenium Selectors – All Tactics Explained 19

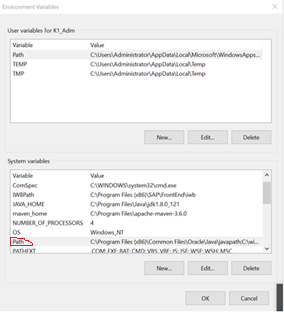
A.2 AppiumLibrary Document 19

A.3 Appium Document 19

# Installation

安装步骤如下：

1. 安装Python 2.7.15 (Python 2.7不行，至少2.7.10以上)
   1. 执行安装包：python-2.7.15.amd64.msi
   2. 将以下路径添加到系统变量Path:



(注意是上图中圈出来的这个PATH)

* + 1. C:\Python27
    2. C:\Python27\Scripts
  1. 安装检查：进入CMD或Bash, 执行以下命令，应该得到以下结果：

**$ python** **--**version

Python 2**.**7**.**15

**$ pip** **-**V

pip xx.x from c**:**\python27\lib\site-packages\pip **(python** 2**.**7**)**

1. 安装JDK
   1. 执行安装包：jdk-8u191-windows-x64.exe
   2. 在系统变量中加入以下变量：

JAVA\_HOME **=** C**:**\Program Files\Java\jdk1.8.0\_191

* 1. 安装检查：进入CMD或Bash, 执行以下命令，应该正确显示版本信息：

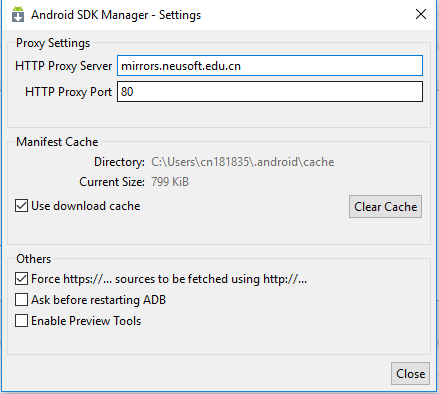
**$ java** **-**version

java version "1.8.0\_191"

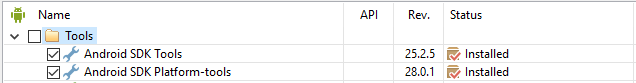
Java**(**TM**)** SE Runtime Environment **(**build 1**.**8**.**0\_191-b12**)**

Java HotSpot**(**TM**)** 64**-**Bit Server VM **(**build 25**.**191**-**b12**,** mixed mode**)**

1. 安装Android SDK
   1. 将 android-sdk\_r24.4.1-windows.zip 解压到C:\ (路径不能包含中文字符)
   2. 运行 SDK Manager.exe
   3. 连接外网，进入Tools -> Options, 按下图配置：



* 1. 勾选以下两个Packages, 点击Install Package， 完成之后Status变为Installed:



* 1. 将以下路径添加到系统环境变量PATH:

C**:**\android-sdk-windows\platform-tools

C**:**\android-sdk-windows\tools

C**:**\android-sdk-windows\tools\bin

* 1. 将以下变量添加到系统环境变量：

ANDROID\_HOME **=** C**:**\android-sdk-windows

* 1. 安装检查：进入CMD，应该能正常执行以下命令：

adb version

uiautomatorviewer

1. 安装Appium
   1. 执行安装包 appium-desktop-setup-1.9.1.exe
2. 安装Robot Framework, AppiumLibrary
   1. 连接外网，在CMD或Bash中执行以下命令：

**$ pip** install robotframework robotframework-appiumlibrary robotframework-debuglibrary

* 1. 检查是否安装成功：

**$ robot** **–-**version

应正确显示版本信息：

Robot Framework 3**.**1 **(**Python 2**.**7**.**15 on win32**)**

# AppiumLibrary Hands-on Guide

## Open Application

每一个测试任务都应该从打开应用(Open Application)开始。

打开应用需要一些设备和应用的相关参数，告诉Appium要测哪个设备，哪个App. 获取参数的步骤如下：

1. 获取设备名称：
   1. 将Android手机调试模式打开，连接电脑
   2. 在命令行中输入执行以下命令：

**$ adb** devices

得到结果：

List of devices attached

26c51f68 device

* 1. 这里的26c51f68就是设备名
  2. 此外，我们可以在Android的 设置/关于手机 中找到Android版本信息

1. 获取App名称：
   1. 手机上打开待测试的App
   2. 以KTEC应用为例，执行以下命令：

**$ adb** shell

$ dumpsys window windows **|** **grep** **-**E 'mCurrentFocus'

得到结果：

mCurrentFocus**=**Window**{**7a5a72c u0 com.kpmg.tax.prc.iit.employee.app**/**com.kpmg.tax.prc.iit.employee.app.MainActivity**}**

* 1. 这里, com.kpmg.tax.prc.iit.employee.app是我们后面用到的appPackage, com.kpmg.tax.prc.iit.employee.app.MainActivity为appActivity.

1. 新建一个 demo1.robot 文件，内容如下：

# demo1.robot

\*\*\* Settings \*\*\*

Library AppiumLibrary

\*\*\* Variables \*\*\*

# Device Specific

${PLATFORM} Android

${VERSION} 6.0.1

${DEVICENAME} 26c51f68

# App Specific

${PACKAGE} com.kpmg.tax.prc.iit.employee.app

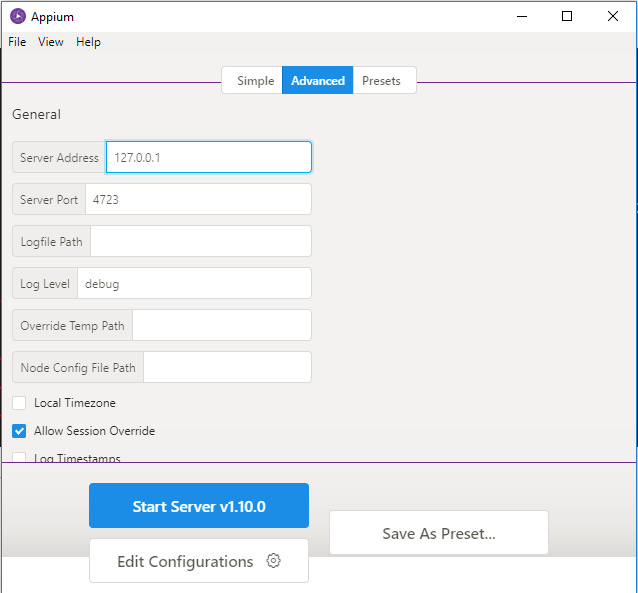
${ACTIVITY} com.kpmg.tax.prc.iit.employee.app.MainActivity

\*\*\* Test Cases \*\*\*

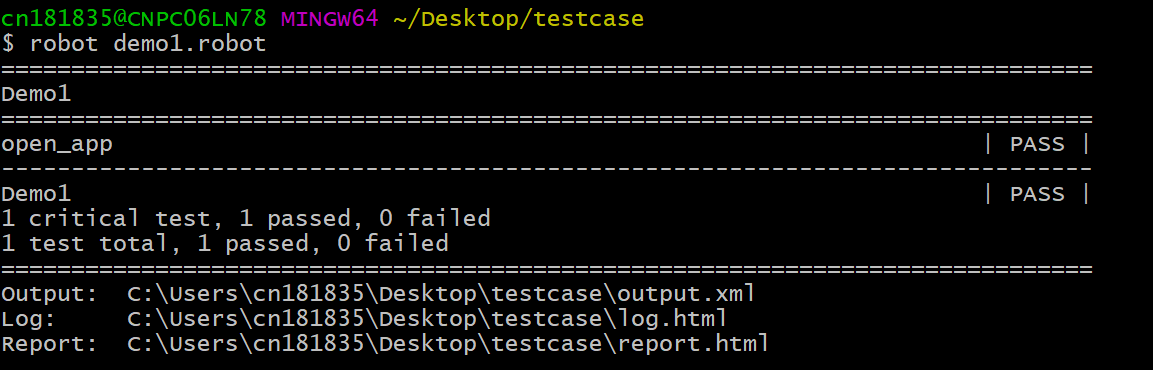
open\_app

Open Application http://localhost:4723/wd/hub platformName=${PLATFORM} platformVersion=${VERSION} deviceName=${DEVICENAME} appPackage=${PACKAGE} appActivity=${ACTIVITY}

1. 打开Appium, 按照下图设置参数，然后Start Server



1. 开始测试。执行**robot** demo1.robot 命令:



应该得到上图所示的PASS 结果，同时手机上自动打开KTEC应用。

## Click Element

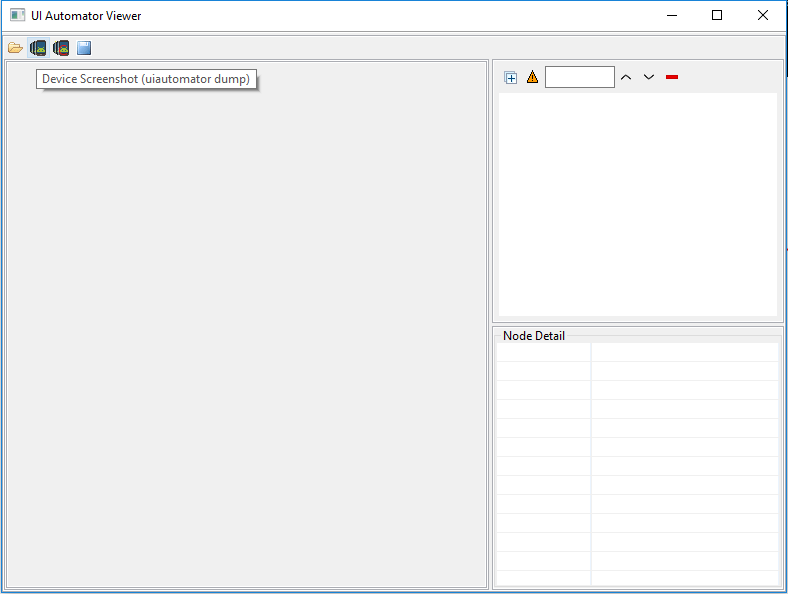
我们现在写一个最简单的测试脚本：点击一个按钮。

Appium Android 使用UIAutomator作为backend, 我们主要使用UIAutomator来定位元素。主要有以下步骤：

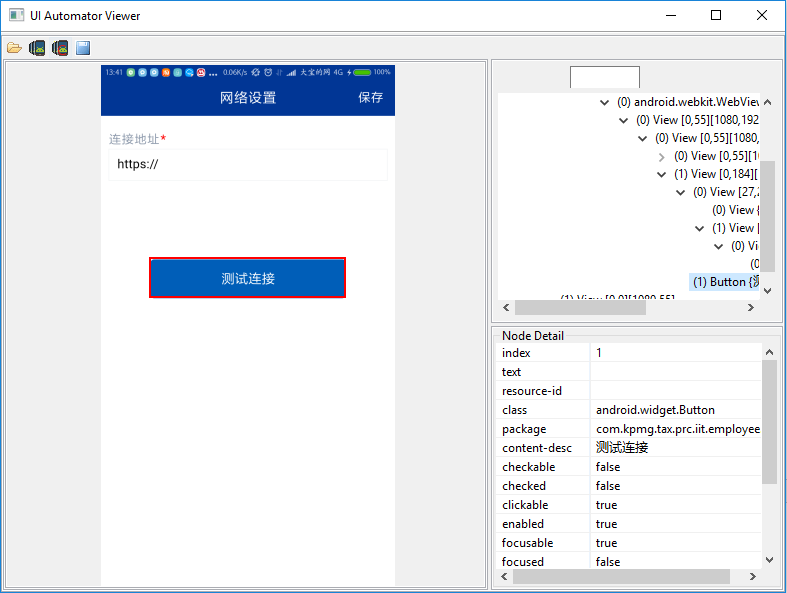
将Android手机调试模式打开，连接电脑。手机上打开待测试的App界面。

运行CMD, 执行命令：uiautomatorviewer

我们将打开以下界面：



点击第二个Button (Device Screenshot)：



这个截图将展示所有的页面元素。假设我们要执行的操作是点击按钮 [测试连接] ：

我们在界面中点击这个元素，找到右下栏将显示该元素所有的Attributes. 我们要找到标志性的Attribute来定位这个元素。这里我们选择content\_desc字段来定位它。

在上文的demo1.robot 的最后加入下面这行：

Click Element xpath=//android.widget.Button[@content-desc="测试连接 "]

这条xpath事实上是在搜索所有android.widget.Button类，筛选出符合content-desc="测试连接 " 的结果。

或者，我们还可以这么写：

Click Element xpath=//\*[@content-desc="测试连接 "]

这样就会对所有类进行搜索，筛选出符合content-desc="测试连接 " 的结果。

由于App会有启动屏幕，我们直接运行测试会报FAIL, 因为在启动屏幕时按钮还没有出现，因此找不到元素。所以我们还需要加入等待操作：

Wait Until Page Contains Element xpath=//android.widget.Button[@content-desc="测试连接 "]

Click Element xpath=//android.widget.Button[@content-desc="测试连接 "]

跟上一节一样运行测试脚本，我们就可以看到Appium将自动打开App并点击按钮，如果成功点击则PASS, 如果按钮不存在则会FAIL。如果FAIL, Appium将会自动保存一张当前手机页面的截图。

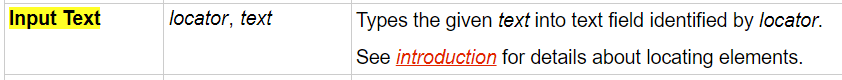
附录A.1提供了更高阶的xpath规则参考。

## Other Keywords

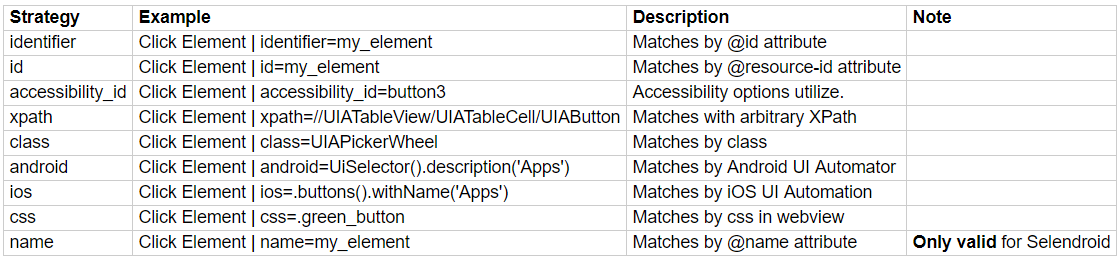
以上是两个最基础的操作：打开应用、点击元素。本节讲通过文档去找其他所有的操作的用法。

AppiumLibrary支持的所有操作可以在这里查看A.2：<http://serhatbolsu.github.io/robotframework-appiumlibrary/AppiumLibrary.html>

**举例。**比如我们要在一个文本框中输入某个字符串。首先我们在文档中找到对应功能的Keyword:



它的第一个变量是locator，用于找到我们要操作的文本框。这个变量其实在上一节中已经使用过了。Locator在文档的开头有描述：



简而言之，我们有这么多种方法去定位一个元素，而我们上一节2.2中使用的是xpath, 我们最常使用也最强大的也是xpath，具体参考上一节。

第二个变量就是要输入的内容，这里是一个字符串。写到RF脚本里面就是：

Input Text xpath=//android.widget.EditText[@text=' 密码'] Kpmg123@

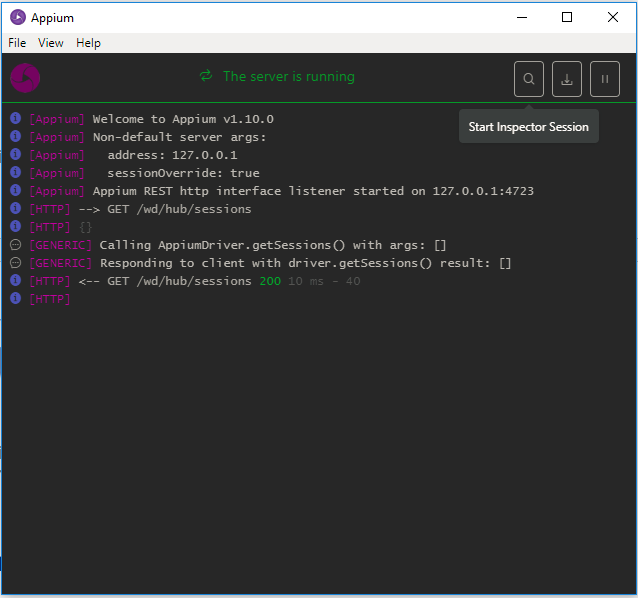
除了Input Text, 其他常用的Keyword 有：

* Click A Point x y: 点击某个坐标
* Swipe x1 y1 x2 x2 duration: 滑动，从(x1,y1)到(x2,y2)
* Wait Until Page Contains Element locator: 等待某个元素出现
* Sleep 2s: 等待2秒
* Clear Text locator: 清除文本框
* Element Text Should Be locator text: 验证元素text属性应该等于
* Element Attribute Should Match locator attribute value: 验证元素的某个属性

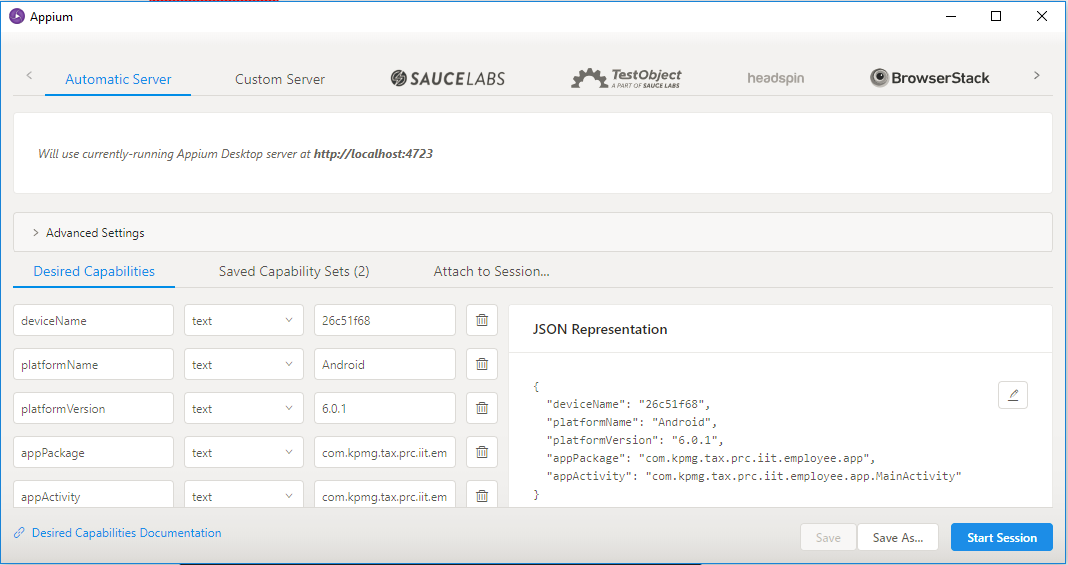
## Alternative way to inspect elements

上面介绍了如何使用UIAutomator查看页面元素。其实，Appium内置了另一个界面可以查看页面元素，叫Appium Inspector。它还提供了各种辅助元素操作的工具，可以结合UIAutomator使用。

首先，启动Appium Server, 点击右上角的放大镜图标。

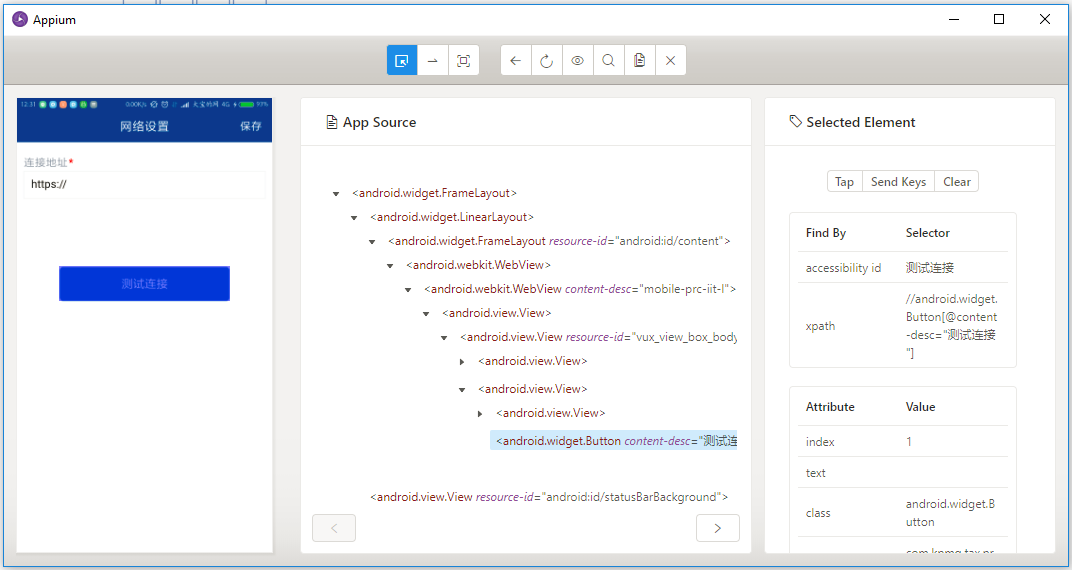


然后，在弹出的页面上输入参数：



可以看到，这里的所有参数与上文Open Application的参数是一致的。

然后点击Start Session。Appium就会启动一个新的界面：



这个界面跟UIAutomator一样可以查看各个元素的属性。除此之外，它还有以下功能：

* 直接与元素交互(Tap, Send Keys, Clear)
* 自动生成locator
* 录制动作, 直接生成代码

虽然方便，但是以下几点值得注意：

1. 自动生成的locator并不一定都能用，还是需要视具体情况去实验。
2. 录制动作功能直接生成的Robot framework代码也比较糟糕，很多情况下需要自己调整。
3. 录制动作并不会把每个操作之间的延时录制下来，我们不要忘了自己在步骤之间加上适当的Sleep命令。

# Development Guide

Section 2 快速介绍了怎么用RF + AppiumLibrary, 接下来讲实际去写test script时候的有用经验，以及如何适应项目需求开发新的Keyword.

## Test Script Development

一般来说，写robot script流程是这样的：

UIAutomator获取元素 -> 写robot script -> 执行robot script -> PASS/FAIL

实际开发中，往往不是很顺利。很多时候你觉得某个xpath看起来是正确的，但是在实际运行的时候却找不到这个元素，那我们就要debug。如果每次做了小修改都要从头开始运行（从打开应用开始, 做完之前的一系列操作），耗时非常长，同时得不到过程中的反馈，非常痛苦。好在，我们在写robot script的时候可以使用Robot Framework提供的Debug Library来在线调试，单独执行某一条指令，可以快速实验。

### Debug Library Installation

如果你follow了Section 1的步骤，Debug Library已经安装好了。如果还没有安装，执行以下命令：

**$ pip** install robotframework-debuglibrary

安装完之后，应该可以正常执行以下command, 显示版本号:

**$ rfdebug** **--**version

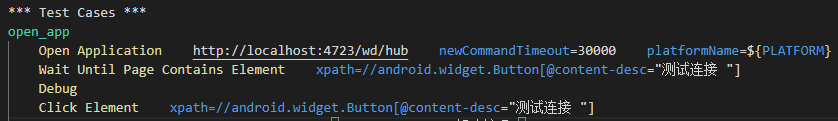
### Introducing RF Interactive Shell

进入Robot Framework 交互界面的方法如下：

在你的 .robot 文件中加入以下内容：

Library DebugLibrary

同时，在你希望停下来调试的地方加入Keyword: Debug



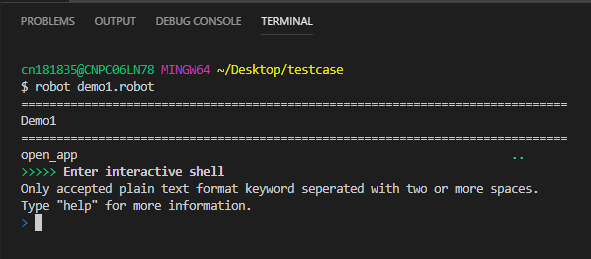
还有一个小细节是，为了延长Appium 的无操作等待时间，我们给Open Application加入了参数：

newCommandTimeout=30000

我们将source code贴在这里以便参考。



执行这个脚本，Robot Framework还是会从头依次执行指令, 但执行到Debug的地方，我们就进入了Debug Interactive Shell:

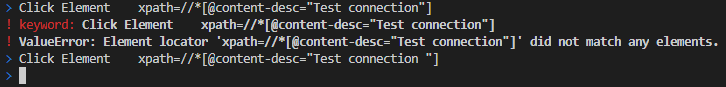


在这个界面，我们可以单独执行某个命令。

举例：假如你想点击某个看起来是Test connection的按钮：



执行一下，会发现找不到。然后你发现原来在UIAutomator中，Test connection后面还带了一个空格。修改后执行：



发现这次可以正确找到元素了。

通过这样单步调试的方法，我们可以debug, 或者快速验证脚本的可行性，甚至获得一些运行中的变量：



总之，Debug Library特别有用，在写robot script的时候往往少不了它。

其他高阶用法请查阅官方文档。

## Develop new keyword

有的时候我们会发现有些功能我们测试的时候需要用到，但是Appium Library没有提供，本节我们讲一下怎么开发新的Keyword.

本节以一个常用功能为例：向下滑动至某个元素出现。Appium Library中确实有一个Scroll Down Keyword, 但很可惜它只支持iOS, 但在Android测试中这种操作也非常常见，那我们看一下怎么样自己实现它。

### Preliminaries

首先得了解一下整个系统的机制：

Robot Framework -> AppiumLibrary -> Appium Python Client -> Appium Server -> Android Device

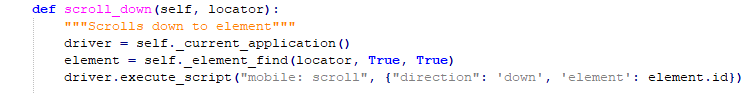
我们写的.robot脚本，由Robot Framework通过AppiumLibrary 来解析Keyword, 将.robot脚本转换成Python代码。Python通过Appium Python Client与Appium通信，告诉Appium Server应该对手机执行什么样的操作，最终Appium Server通过adb来实际操作Android手机。

显然，如果我们要增加新的Keyword, 我们应该对AppiumLibrary进行开发。

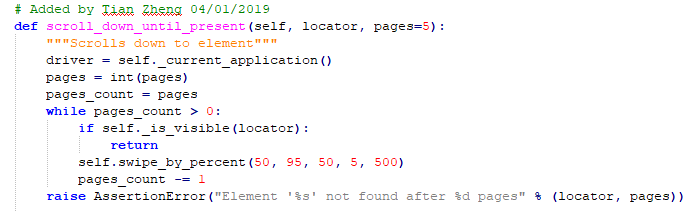
AppiumLibrary的位置在：

C**:**\Python27\Lib\site-packages\AppiumLibrary

所有Keywords都在 AppiumLibrary\keywords 下面。我们打开\_touch.py

我们可以看到，Scroll Down关键词的代码就在这里：

我们要实现的功能假设取名为Scoll Down Until Present, 我们只要在\_touch.py中加入我们自己写的函数就可以了：



这样，我们就可以在robot script里面这样调用：

Scroll Down Until Present xpath=//\*[@content-desc="邮政编码"]

或者

Scroll Down Until Present xpath=//\*[@content-desc="邮政编码"] 3

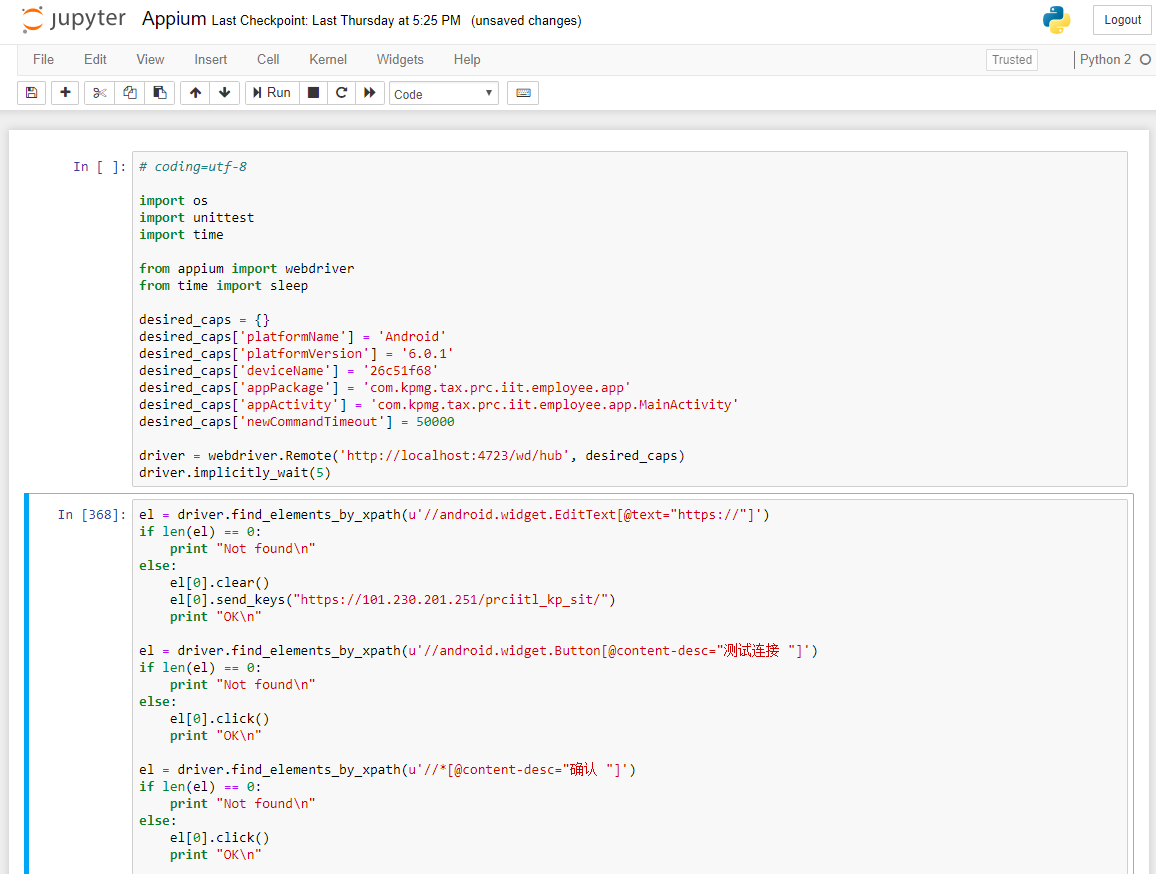
参数列表与同名Python函数的参数列表对应。

### Appium Python Development

同样，由于调试原因，直接在robot framework里写python代码比较痛苦，因为无法在运行中修改代码(change code on-the-fly)，也很难print一些变量来调试。同时每次修改之后重新运行都要等待App启动和一堆初始化操作，很慢。我个人采用的方法是Jupyter Notebook，这样可以在运行中修改代码与调试。

具体Jupyter Notebook如何安装使用就不再赘述，可以使用pip 安装。

在Jupyter Notebook中，代码分成cells, 每个cell可以单独执行。我们就可以把所有的App打开之类的准备动作放到第一个cell里面去，这样每次只需要执行一次启动动作，可以方便地在后面的cell里面调试我们自己的程序：



Appium Python Client具体使用方法请查询官方文档，链接在附录中A.3。当文档不够详细或错误的时候，也可以直接参考源码。Appium Python Client的路径在：

C**:**\Python27\Lib\site-packages\appium

# iOS Testing Guide

## Installation

1. 安装Xcode
2. 安装Python 2.7.10+
3. 安装pip
4. **$ pip** install robotframework robotframework-appiumlibrary robotframework-debuglibrary
5. 安装Appium, 按照以下文档（安装Appium Desktop版本，详见文档）

<http://appium.io/docs/en/about-appium/getting-started/#driver-specific-setup>

1. 安装iOS驱动：<http://appium.io/docs/en/drivers/ios-xcuitest/index.html>

验证安装：

1. 启动Appium
2. 打开Appium inspector
3. 输入以下参数：
   1. automationName : XCUITest
   2. platformVersion ： 12.1
   3. platformName : iOS
   4. deviceName : iPhone X
   5. app : 安装包所在的绝对路径 (或者使用bundleId = com.xxx.xxx 来启动已经安装的app)
4. 确定之后，将自动启动iOS虚拟机，并能正确显示APP的界面截图

## Differences from Android testing

iOS测试与Android在后端驱动上不一样。Android使用的是UIAutomator, 而iOS使用XCUITest. 因此在实际操作中也有一些不一样的地方：

1. Open Application的参数。如上一节所示，参数与Android有所差异。以在虚拟机中运行为例，参数如下：
   * 1. automationName : XCUITest
     2. platformVersion ： 12.1
     3. platformName : iOS
     4. deviceName : iPhone X
     5. app : 安装包所在的绝对路径 (或者使用bundleId = com.xxx.xxx 来启动已经安装的app)
2. 检索元素：只能使用Appium inspector。
3. 有一部分输入框元素并不能直接输入，直接Input Text会报错找不到元素或者键盘未启动。如果遇到这样的情况，解决方法是：
   1. 虚拟机设置中打开软键盘（Toggle Software Keyboard）
   2. 先点击一下文本框激活键盘（Click A Point）, 再执行Input Text
4. Appendix: Useful Documents
   1. How to Write Effective XPath Selenium Selectors – All Tactics Explained

本文是针对Web xpath书写的，Android xpath的规则事实上是一样的，可以参考。

<https://www.swtestacademy.com/xpath-selenium/#xpath-following-sibling>

* 1. AppiumLibrary Document

AppiumLibrary官方文档。

<http://serhatbolsu.github.io/robotframework-appiumlibrary/AppiumLibrary.html>

* 1. Appium Document

Appium的官方文档。

https://github.com/appium/appium/tree/master/docs/en/commands