**[Appium+Robotframework实现Android应用的自动化测试-1：Appium在Windows中的安装](http://www.cnblogs.com/testlife007/p/4710641.html)**

让我们开始在Windows中开始安装Appium吧，Appium在OS X中的具体安装后面的文章会介绍。  
  
另外，官网上说先要装Node.js，还要装Apache Ant和Apache Maven,Git以及cURL，不过我的经验是这些不是必须的，可以不装，当以后需要时再装也不迟，这样一开始安装比较容易和上手。  
  
废话少说，直接开始安装步骤：  
  
1.安装android的sdk包，(<http://developer.android.com/sdk/index.html>), 运行依赖sdk中的'android'工具。并确保你安装了Level17或以上的版本api，建议至少安装到19，我安装到了22。  
  
2.设置ANDROID\_HOME系统变量为你的Android SDK路径，并把tools platform-tools两个目录加入到系统的Path路径里。因为这里面包含有一些执行命令。  
  
3.安装java的JDK，并设置JAVA\_HOME 变量为你的JDK目录。  
  
4.安装Appium for Windows版：下载路径 <http://appium.io/downloads.html>，我用的是最新的1.4版本，很好。  
  
5.安装好Python 2.7版本，虽然Appium支持很多语言，但个人最偏爱Python，还有一个原因是RobotFramework也支持Python，接下来我是用RobotFramework来写Appium的哦。  
  
6. 安装Appium Client中的Python支持包，打开命令行，输入：pip install Appium-Python-Client，也可以直接下载python-client-master.zip，然后将其解压缩，打开命令行，先切到解压 缩所在的路径，之后输入：python setup.py install 来完成安装。  
  
如果都没有报错，那恭喜你，成功了！如果遇到问题，请仔细查看错误提示，一般都能解决。  
  
待续，请持续关注...

# [Appium+Robotframework实现Android应用的自动化测试-2：Windows中启动Appium和模拟器](http://www.cnblogs.com/testlife007/p/4925318.html)

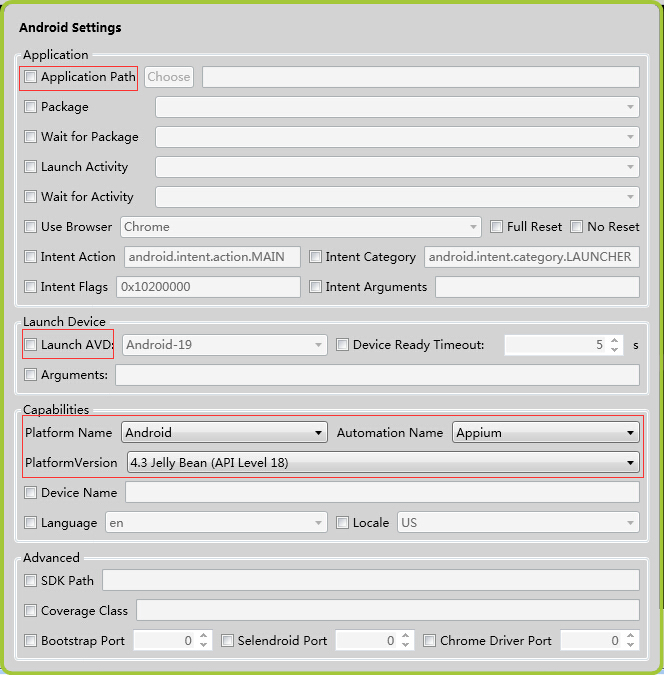
一.启动Appium

安装好了之后，在桌面或者菜单中找到Appium，分别双击或点击打开Appium.exe，如果一切正常，接着会出现一个Appium启动后的界面窗口，如下图所示。



1.1 Android Settings

点击左上角的第一个机器人图标，弹出Android Settings窗口，如下图所示。



在该窗口可以对将要进行测试的Android应用进行设置，因为接下来我们写脚本来跑测试，所以可以统统不进行设置，保持其默认值即可。如果不在脚本中设置，则需要在这里设置下，不过大部分保持默认值即可，需要进行设置主要有5个：

1.Application Path：点击Choose按钮后会弹出选择Android应用的路径选择框，选择需要测试的应用即可。

2.Launch AVD: 如果有多个模拟器，这里选择一个作为测试用的模拟器。

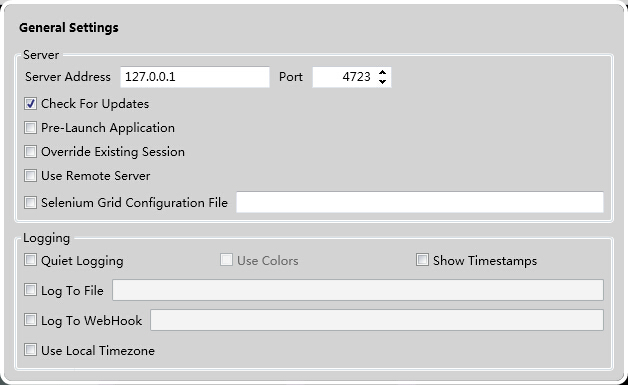
3.Platform Name:这个当然选择Android，因为现在是在说Android的自动化测试。

4.Automation Name:当然选择Appium，我们不是在玩Appium吗？

5.PlatformVersion:这个当然选择和模拟器中一样的版本啦。

1.2 General Settings

点击Appium左上角的第二个齿轮图标则弹出General Settings 窗口，如下图所示。



如果就在本机安装了Appium，则都可以保持默认值，如果把Appium安装在了其他机器，则需要设置Server Address，值就是Appium所在机器的IP地址，端口号一般无需更改。

1.3 启动Appium

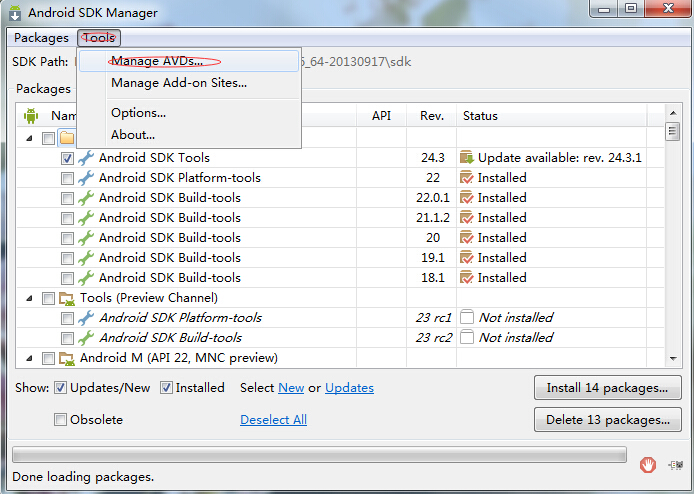
这次直接跑Appium，所以进行相关的设置，点击Appium右上角的正方形图标启动Appium，如果能看到如下图所示的信息就表示启动成功了。

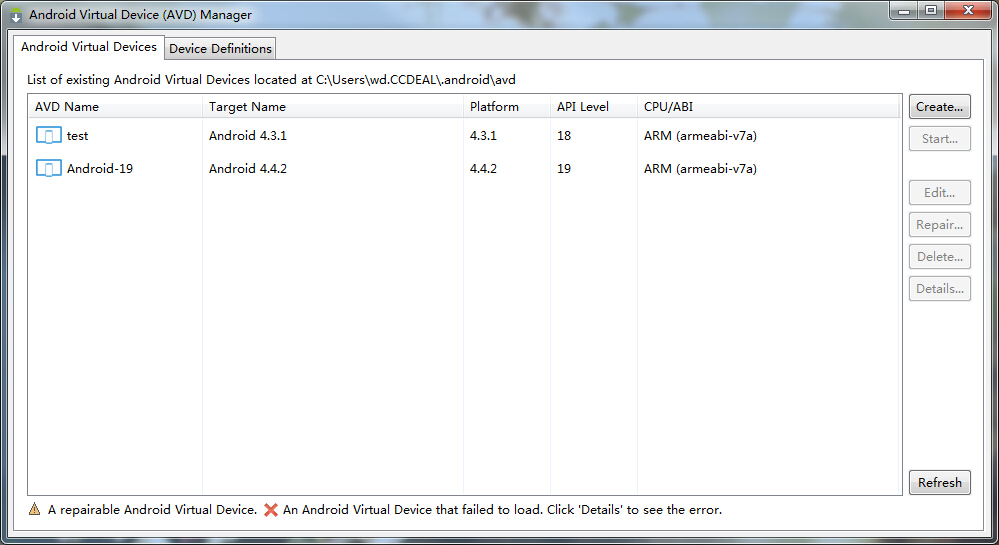


二.启动模拟器

2.1 启动虚拟设备管理器

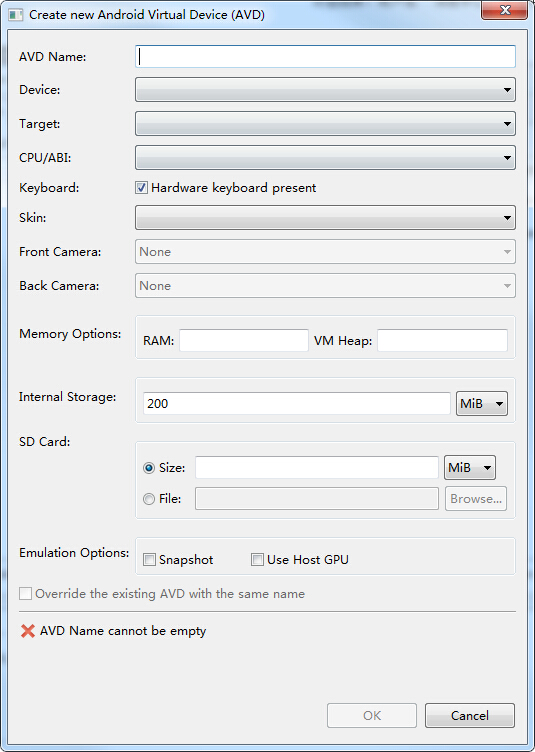
进入到ANDROID\_HOME\sdk目录中，然后双击AVD Manager.exe文件，如果没有该文件，则可双击SDK Manager.exe文件，然后在打开的窗口中点击菜单Tools，接着点击其Manage AVDs,,,子菜单项，这样就打开了Android Virtual Device(AVD) Manager，如下图所示。这个是Android虚拟设备管理器，利用它我们可以创建，编辑和启动具体的模拟器。





2.2 创建和编辑模拟器

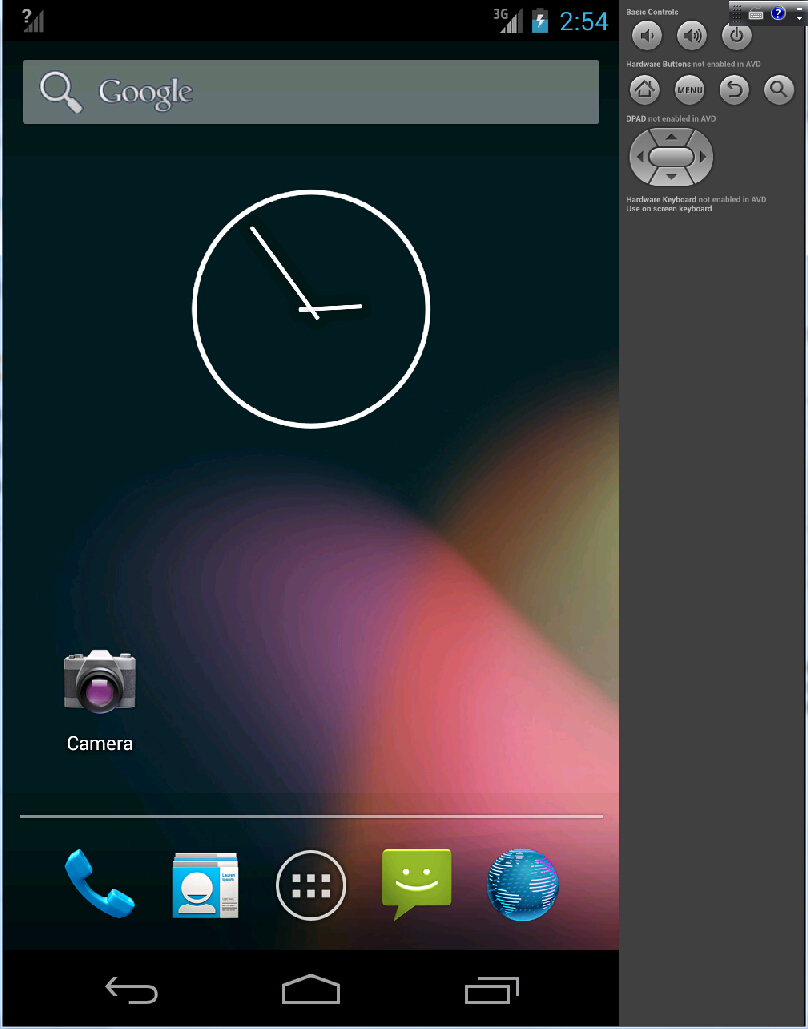
点击Create按钮用于创建一个新的模拟器，这将会打开一个创建新的模拟器的窗口，如下图所示。



在这个窗口中输入或者选择所需的模拟器的参数，这里需要注意的是Target要选择 和Appium或者脚本中相同的参数，Device要选择符合对应的Target，另外要保证CPU/ABI的值不能为空，Memory Options中的RAM不要设置过大，我一般设置为256或者512，VM Heamp设置为64，其他选项都保持默认值即可，最后点击底部的OK按钮就创建完毕了。创建成功后会出现在列表中。选中该项，可进行编辑操作，和创建类 似，不再赘述。

2.3  启动模拟器

确定都没有问题了，选中需要启动的模拟器，点击Start按钮，在弹出的 Launch Options对话框中直接点击Launch按钮，就会出现启动模拟器的进度条，之后出现模拟器的窗口，刚开始该窗口中一片漆黑，那是因为模拟器还没有启 动完毕，等待一会儿，可以趁机泡杯茶，就看到启动成功了，如下图所示。完全就是一个Android手机的感觉有木有。

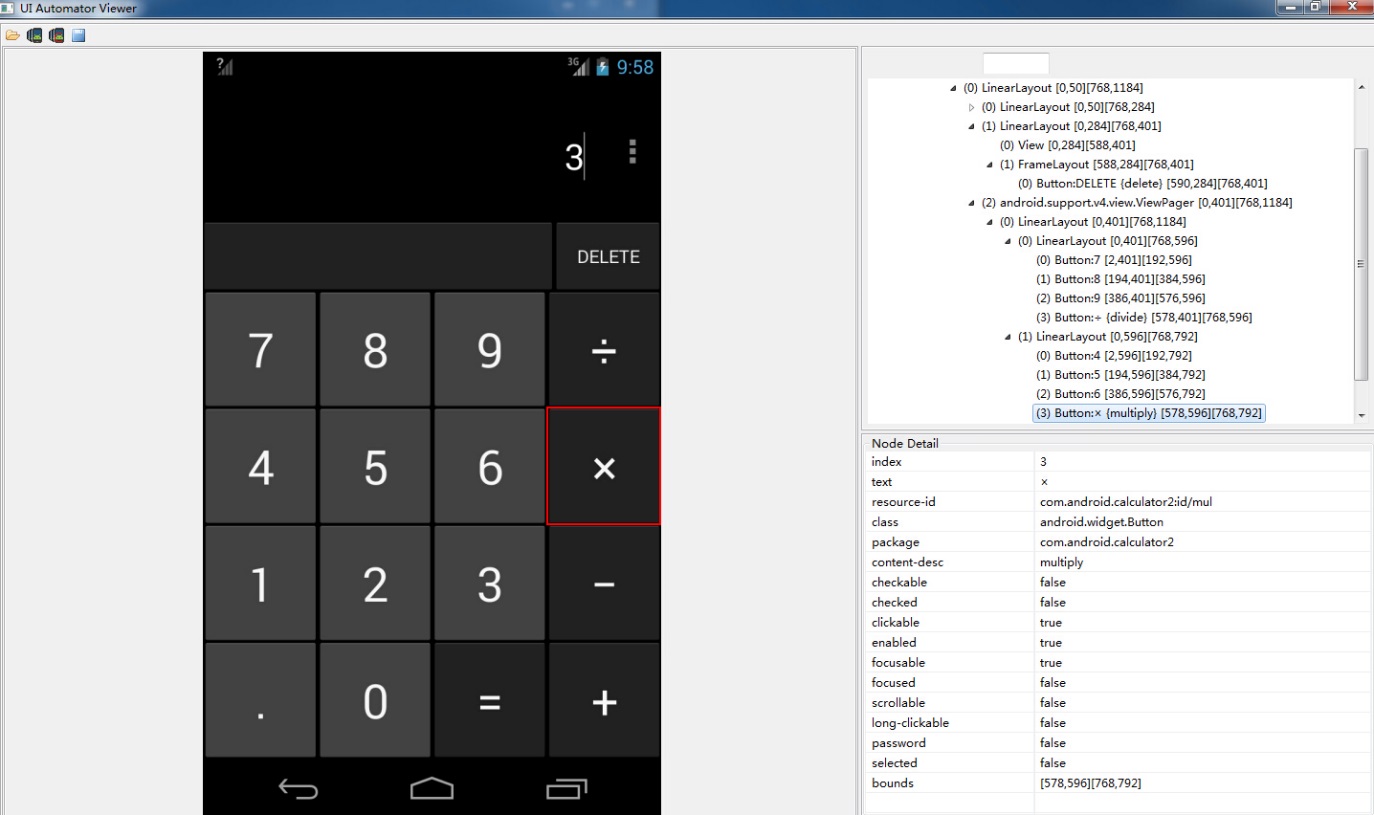


# [Appium+Robotframework实现Android应用的自动化测试-3：一个必不可少的工具介绍](http://www.cnblogs.com/testlife007/p/4925319.html)

万事具备，接下来我们就要开始编写测试脚本了。

不过，有个重要的问题还没有解决。我们知道RobotFramework在web 网页中主要依靠元素的id，name或者xpath来定位页面上的元素，我们依赖浏览器的插件firepath来做到这一点，但在Android应用中， 怎么定位页面上的元素呢？现在，这个关键的工具就该出场了，它就是uiautomatorviewer.bat，这个工具位 于%ANDROID\_HOME%\tools目录下，双击运行它。

如果模拟器还没有启动，也把它启动起来，等模拟器启动完毕出现手机的待机界面时，对 模拟器进行操作，我们打开计算器，点击数字6，再点击乘号，这时点击uiautomatorviewer.bat左上角的第二个图标Device Screenshot，这个图标点击后能对模拟器进行截图，但更神奇的是当我们将光标移动到截图中的元素后，在其右侧就会自动出现该元素的属性，如下图所 示。



经常会用到的是resource-id，text，class属 性，resource-id在AppiumLibrary(后面的文章会介绍)中其实就是id属性，想起来了吧，在RobotFramework中我们不 是经常使用id吗？不过要注意：只有当测试的应用的版本设置为4.3及以上，才能看到resource-id哦。

有了这个方便好用的工具，接下来的编写代码脚本的工作就简单多了。

# [Appium+Robotframework实现Android应用的自动化测试-4：AppiumLibrary介绍和安装](http://www.cnblogs.com/testlife007/p/4925323.html)

Appium是个好东东，Android，iOS都支持，并且居然RobotFramework也支持Appium了，这就是本文要介绍的AppiumLibrary。

通过前面的文章大家知道可以使用多种语言来写Appium的测试脚本，但如果从编写效率和学习曲线上来说，当然是用如RobotFramework的关键字的方式最为简单，所以AppiumLibrary就出现了，从[Appium官网](https://github.com/jollychang/robotframework-appiumlibrary" \t "_blank)可知最新的版本是1.2.5，在官网有其介绍和如何安装，为了方便不喜欢看英文的朋友，下面我把安装的方法简单介绍下。

Appium的安装有两种方式，第一种就是使用pip指令，具体为：pip install robotframework-appiumlibrary，当然前提条件是安装好了Python2.7，并且也安装了pip工具。

第二种方式就是使用setup.py，可以官网页面右侧的下载链接下载后再进行解压缩，在命令行模式下进行到解压缩后所在的目录，最后执行指令python setup.py install，或者先把git安装好，然后依次执行三条指令：

git clone <https://github.com/jollychang/robotframework-appiumlibrary.git>  
cd robotframework-appiumlibrary  
python setup.py install

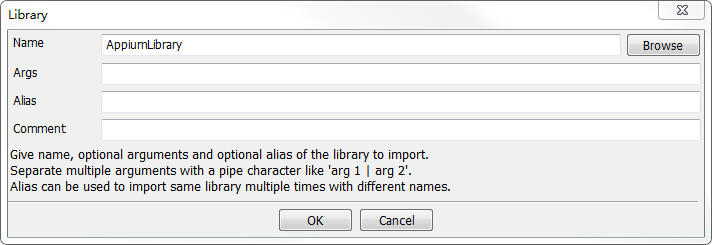
没有提示错误就是安装成功了。

# Appium+Robotframework实现Android应用的自动化测试-5：RIDE中AppiumLibrary的配置

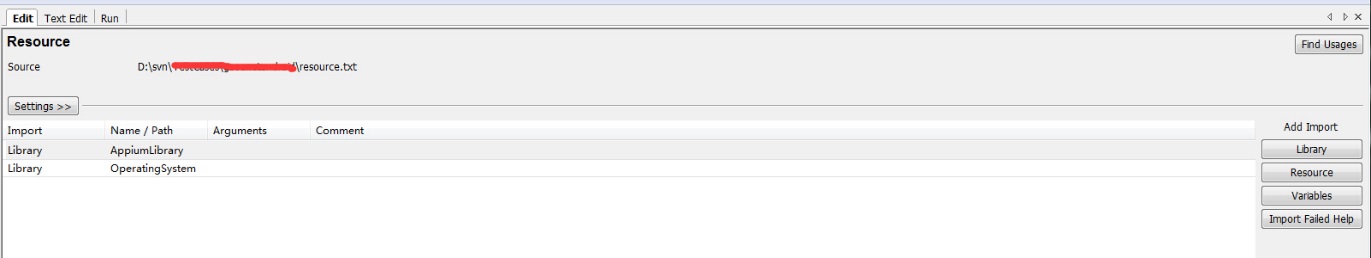
可能很多朋友已经迫不及待的想要用RobotFramework+AppiumLibrary来写Android App的测试脚本了，那我们也废话少说，直接开始。

首先打开RIDE，这是编写RobotFramework测试脚本的集成环境，如果没有则先安装好，当然RobotFramework也要安装好，这些大家可以在网上查找相关的资料来解决如何安装的问题。

其次，新建一个resource.txt文件，这个文件是用来放置共用的脚本的，之后点击Edit选项卡，再点击右侧的Library按钮用于添加AppiumLibrary库，在弹出的的Name输入框中输入AppiumLibrary，如下图所示。



最后点击OK按钮。如果在Settings下方出现黑色的AppiumLibrary行，就表明设置成功了，如下图所示。



# [Appium+Robotframework实现Android应用的自动化测试-6：一个简单的例子](http://www.cnblogs.com/testlife007/p/4925327.html)

万事具备，只欠编码！

下面看一个简单的示例，这个示例验证Android手机自带的通讯录的添加联系人的操作是否成功。这个例子是Appium官网自带的示例，有兴趣的同学也可以自己下载来研究和学习，下载地址：<https://github.com/appium/sample-code>

首先请看resource.txt文件的代码：

1. \*\*\* Settings \*\*\*
2. Library           AppiumLibrary
3. \*\*\* Variables \*\*\*
4. ${REMOTE\_URL}     http://localhost:4723/wd/hub
5. ${PLATFORM\_NAME}    Android
6. ${PLATFORM\_VERSION}    4.4.4
7. ${DEVICE\_NAME}    Android Emulator
8. ${APP}            ../../../ContactManager/ContactManager.apk
9. \*\*\* Keywords \*\*\*
10. add new contact
11. [Arguments]    ${contact\_name}    ${contact\_phone}    ${contact\_email}
12. Open Application    ${REMOTE\_URL}    ${PLATFORM\_NAME}    ${PLATFORM\_VERSION}    ${DEVICE\_NAME}    ${APP}
13. Click Element    accessibility\_id=Add Contact
14. Input Text    id=com.example.android.contactmanager:id/contactNameEditText    ${contact\_name}
15. Input Text    id=com.example.android.contactmanager:id/contactPhoneEditText    ${contact\_phone}
16. Input Text    id=com.example.android.contactmanager:id/contactEmailEditText    ${contact\_email}
17. Click Element    accessibility\_id=Save

复制代码

这里重点要说的是Variables下的五个变量，它们都是Open Application关键字的参数，用于在测试执行时提供给Appium相关的参数设置，之前的文章曾经说过可以不设置，在测试脚本中进行设置，这五个就是进行相关设置的参数。下面分别做下解释说明。

${REMOTE\_URL}：远程URL，指的是Appium所在的地址以及端口号，之后的/wd/hub为固定格式，请不要改动。

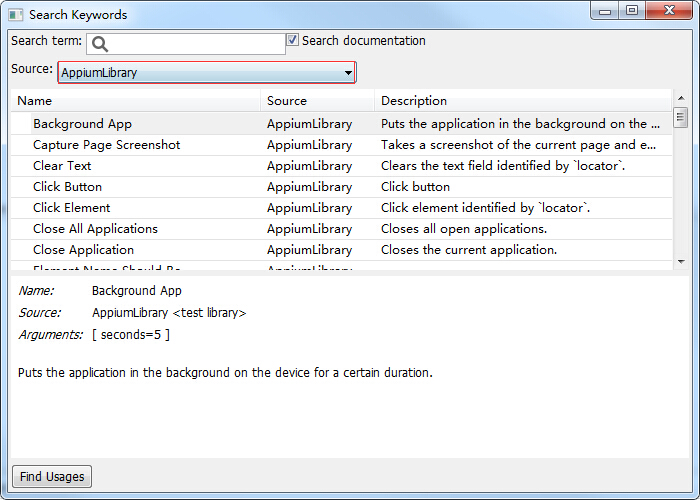
${PLATFORM\_NAME}：平台名称，我们在Android中进行测试，只能写Android。

${PLATFORM\_VERSION}：平台版本，也就是Android的版本号，这个要和模拟器中设定的版本相同即可。

${DEVICE\_NAME}：设备名称，就是运行中的模拟器的名称，如果不知道，可以通过在命令行中输入adb devices指令取得。

${APP}：要测试的App的全路径，注意这个路径指的是相对于Appium所在的机器的路径，这里的App需要从是Appium官网下载，下载地址：[示例代码下载](https://github.com/appium/sample-code" \t "_blank)，下载后在sample-code\apps\ContactManager目录下可找到ContactManager.apk。

接下来的Keywords就简单了，可以打开[AppiumLibrary](http://jollychang.github.io/robotframework-appiumlibrary/doc/AppimuLibrary.html" \t "_blank)得知每个关键字的具体含义，也可以更直接的在RIDE中查看关键字的含义，点击左上方的K图标，然后弹出Search Keywords窗口，再选择Source为AppiumLibrary，如下图所示。



如何获取元素在之前的文章说过了，怎么样？写起来是不是很简单？

接下来再来看看contacts.txt里面的脚本：

1. \*\*\* Settings \*\*\*
2. Resource          resource.txt
3. \*\*\* Test Cases \*\*\*
4. add\_contact
5. [Documentation]    demo for android\_contacts(https://github.com/appium/sample-code/blob/master/sample-code/examples/python/android\_contacts.py)
6. [Tags]    demo
7. add new contact    Appium User    someone@appium.io    5555555555
8. Page Should Contain Text    Appium User

复制代码

这个就更简单了，调用resource.txt文件里面的add new contact关键字，参数就是要添加的联系人的姓名，邮件和电话，最后一行是验证添加是否成功。

好了，有兴趣的朋友可以自己试试哦。

# [Appium+Robotframework实现Android应用的自动化测试-7：模拟器频繁挂掉的解决方案](http://www.cnblogs.com/testlife007/p/4925328.html)

如果测试用例比较多，则当持续运行多个测试用例后，经常会出现模拟器崩溃或者Appium无法连接到该模拟器的情况出现。

经过分析，本人认为这应该是模拟器或者Appium的缺陷造成的，目前并没有直接的解决方案。

不过可以曲线救国，我的解决方案是：在开始测试时启动模拟器，在测试结束后关闭模拟器，这样虽然需要更多的时间，但这个问题确实不再出现了，时间有的是，可以让其在夜间自动启动执行就可以了。

在Settings中加入Process库，这个库是Robotframework自带的，写明即可。

1. \*\*\* Settings \*\*\*
2. Library           AppiumLibrary
3. Library           Process

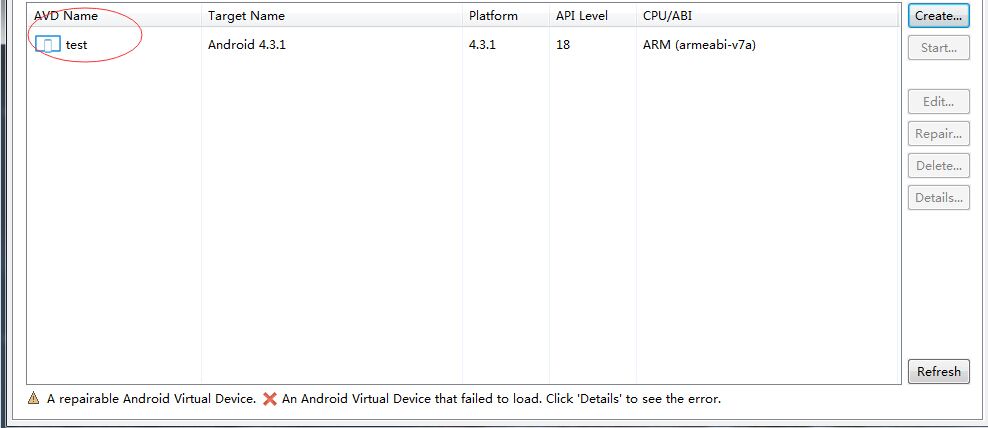
复制代码

启动和关闭模拟器的脚本如下：

1. StartEmulator
2. Run Process    emulator    @xxx
3. Sleep    180s
4. EndEmulator
5. Run Process    tskill    emulator-arm
6. Run Process    tskill    adb

复制代码

通过调用Process库的Run Process关键字来做到调用相关的程序。调用emulator就会启动模拟器，xxx是模拟器的名字，如下图所示：



Sleep 　　180s的目的是等待模拟器完成启动过程，这个三分钟足够了。

调用tskill后杀死指定名称的进程，不多解释了。

这里再介绍下Suite Setup，Suite Teardown以及Test Setup和Test Teardown ，它们分别是两组关键字：Suite Setup表示在一组测试用例执行前执行，Suite Teardown表示在一组测试用例执行后再执行；Test Setup表示在每个测试用例执行前执行，Test Teardown表示在每个测试用例执行后再执行。一组测试用例写在同一个文件中。

在具体的测试脚本的文件的Settings中有如下脚本：

1. \*\*\* Settings \*\*\*
2. Suite Setup       StartEmulator
3. Suite Teardown    EndEmulator

复制代码