

从零开始的Appium测试on Android

本文从Appium的概念开始，逐步介绍了Appium的环境配置、简单使用，并以本部门Xcall app为例，说明了如何使用Appium控制两部手机互相拨打电话并测试。

一、背景

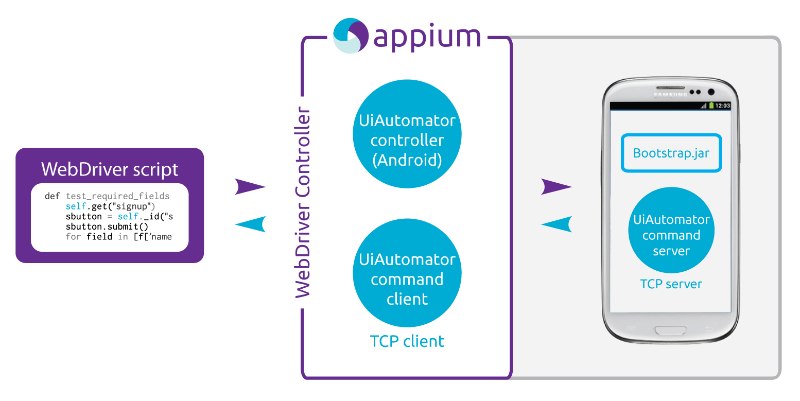
1. Appium简介

Appium 是一个自动化测试开源工具，支持 iOS 平台和 Android 平台上的原生应用，web 应用和混合应用。

所谓的“移动原生应用”是指那些用 iOS 或者 Android SDK 写的应用。所谓的“移动 web 应用”是指使用移动浏览器访问的应用（Appium 支持 iOS 上的 Safari 和 Android 上的 Chrome）。所谓的“混合应用”是指原生代码封装网页视图——原生代码和 web 内容交互。比如，像 Phonegap，可以帮助开发者使用网页技术开发应用，然后用原生代码封装，这些就是混合应用。更重要的是，Appium 是一个跨平台的工具：它允许测试人员在不同的平台（iOS，Android）使用同一套API来写自动化测试脚本，这样大大增加了 iOS 和 Android 测试套件间代码的复用性。

1. Appium相关概念
2. C/S 架构

Appium 的核心是一个 web 服务器，它提供了一套 REST 的接口。它收到客户端的连接，监听到命令，接着在移动设备上执行这些命令，然后将执行结果放在 HTTP响应中返还给客户端。事实上，这种客户端/服务端的架构给予了许多的可能性：比如我们可以使用任何实现了该客户端的语言来写我们的测试代码。比如我们可以把服务端放在不同的机器上。比如我们可以只写测试代码，然后使用像 Sauce Labs 这样的云服务来解释命令。图中，UIAutomator是Android的本机UI自动化框架，它支持从命令行直接运行junit测试用例。它使用java作为编程语言，但是Appium将使用任何WebDriver支持的语言运行它（包括Python，PHP，Ruby等）。在图中可以看到，有一个Bootstrap.jar代替了在java编译时代表测试用例的bootstrap.js。一旦它启动，它产生一个TCP服务器。这里的TCP服务器位于设备内，客户端位于Appium进程中。



1. Session

自动化始终围绕一个session进行，客户端初始化一个seesion（会话）来与服务端交互，不同的语言有不同的实现方式，但是他们最终都是发送为一个POST请求给服务端，请求中包含一个JSON对象，被称作“desired capabilities”。此时，服务端就会开启一个自动化的 session，然后返回一个 session ID，session ID将会被用户发送后续的命令。

1. Desired Capabilities

Desired capabilities 是一些键值对的集合 (比如，一个 map 或者 hash），客户端将这些键值对发给服务端，告诉服务端我们想要怎么测试。比如，我们可以把platformName capability 设置为 iOS，告诉 Appium 服务端，我们想要一个iOS 的 session，而不是一个 Android 的。我们也可以设置 safariAllowPopups capability 为 true，确保在 Safari 自动化 session 中，我们可以使用 javascript 来打开新窗口。参见 capabilities 文档，查看完整的 capabilities 列表。

1. Appium Server

Appium server 是用 Node.js 写的。我们可以用源码编译或者从 NPM 直接安装。

1. Appium 服务端

Appium 服务端有很多语言库 Java, Ruby, Python, PHP, JavaScript 和 C#，这些库都实现了Appium 对 WebDriver 协议的扩展。当使用 Appium 的时候，你只需使用这些库代替常规的 WebDriver 库就可以了。

1. Appium.app, Appium.exe

Appium的官方网站提供了 GUI 封装的 Appium 服务端下载，它封装了运行 Appium服务端的所有依赖，而不需要担心怎样安装Node.js。其中还包括一个Inspector工具，可以帮助你检查应用的界面层级，这样写测试用例时更方便。

二、环境配置：

1. 安装好nodejs，npm和Android SDK；
2. 使用npm命令安装Appium：

npm install -g appium；

1. 安装Appium中对应驱动：

npm install appium-android-driver

1. Appium支持包括PHP、Ruby、JavaScript、Python在内的多种语言，本文以Python为例：

安装python35，并配置环境变量

1. 使用python自带的pip命令配置客户端环境：

pip install Appium-Python-Client

pip install pytest；

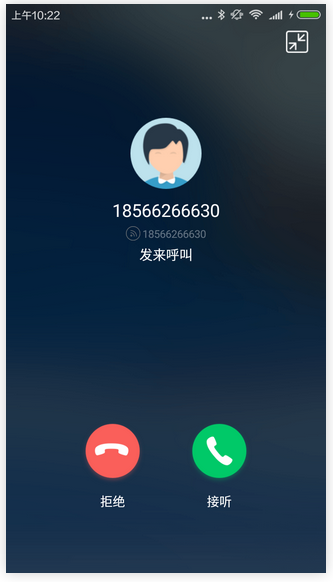
安装完毕后，可以下载官网例程运行：

git clone <https://github.com/appium/sample-code>。

三、使用Appium流程

1. 使用uiautomatorviewer或者appium的app-inspector来查看app中元素的id。

经过查看后得知，Xcall通话界面的元素id大致如下图。



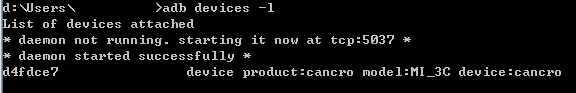
1.接听：ctrip.android.xcall:id/btn\_answer

2.挂断：ctrip.android.xcall:id/btn\_reject

ctrip.android.xcall:id/tv\_call\_state

ctrip.android.xcall:id/ll\_user\_info

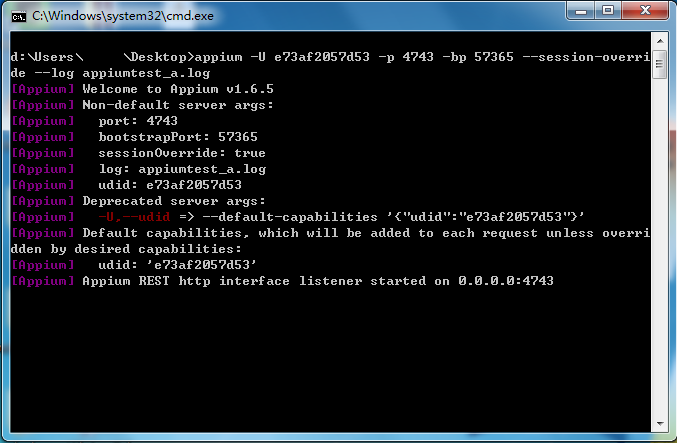
1. 插上真机或者启动了虚拟机之后，在命令行中使用adb devices –l命令来查看设备信息



1. 参考官网例程中的测试文件，修改python测试文件代码中的desired\_caps配置信息使之符合2中真机或虚拟机的信息，主要是修改虚拟机或真机的系统版本号（platformVersion）和设备号（udid）、安装包的路径信息（app）以及要连接的Appium服务器的地址（port\_info）。

|  |
| --- |
| desired\_caps[**'platformVersion'**] = **'6.0.1'**  desired\_caps[**'app'**] = PATH(  **'D:/apps/xcall-android-v0.9-debug.apk'** )  desired\_caps[**'udid'**] = **'e73af2057d53'**  port\_info = **'http://localhost:4743/wd/hub'** |

1. 启动appium的server。



1. 运行python脚本

四、多进程通信控制多台手机同时测试

多进程通信控制多台手机同时测试的架构如下：

 在主进程中启动两个子进程，两个子进程分别对应两个不同的Appium服务器，子进程之间使用Pipe进行同步，以保证两者的测试程序在顺序上一一对应。

1. 启动两个appium服务器，分别指定不同的端口

|  |
| --- |
| appium -U e73af2057d53 -p 4743 -bp 57365 --session-override --log appiumtest\_a.log  appium -U 174ac0047d43 -p 4733 -bp 56494 --session-override --log appiumtest\_b.log |

1. 对于手机a和手机b，写两个python脚本，分别指定端口对应appium服务器。

其中，desired\_caps是指系统属性，可以在Appium的官网上查到，’noReset’如果手机中已经安装此APP，则测试时不重新安装，’udid’是指测试手机的设备号，可以通过adb devices –l命令查询得到，port\_info中的端口号对应上一条中服务器启动的端口号。

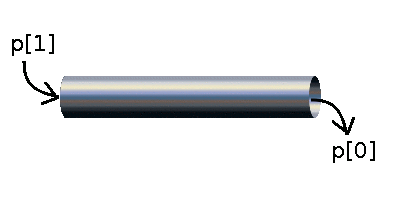
|  |
| --- |
| **class** SimpleAndroidTests(unittest.TestCase):  **def** setUp(self):  desired\_caps = {}  desired\_caps[**'platformName'**] = **'Android'** desired\_caps[**'platformVersion'**] = **'6.0.1'** desired\_caps[**'deviceName'**] = **'Android Emulator'** desired\_caps[**'app'**] = PATH(  **'D:/git/sample-code/sample-code/apps/xcall-android-v0.9-debug.apk'** )desired\_caps[**'noReset'**] = **True** *# phone\_a, the golden redmi* desired\_caps[**'udid'**] = **'e73af2057d53'**   port\_info = **'http://localhost:4743/wd/hub'** self.driver = webdriver.Remote(port\_info, desired\_caps)   **def** test\_something(self):  *# do something*  pass   **def** tearDown(self):  self.driver.quit() |

1. 多进程通信解决同步问题：
2. Python下使用多进程

|  |
| --- |
| p1 = multiprocessing.Process(target=one\_test, args=(pipe[0],)) p2 = multiprocessing.Process(target=two\_test, args=(pipe[1],))  p1.start() p2.start()  p1.join() p2.join() |

使用Python的multiprocessing模块即可创建多进程，其中target是指以函数one\_test/two\_test为对象创建多进程，也可以以类为对象创建，target是指传入的参数，此处为了控制多个手机，传入了用于进程通信的pipe对象。

1. 使用pipe通信



|  |
| --- |
| pipe = multiprocessing.Pipe()  p1 = multiprocessing.Process(target=one\_test, args=(pipe[0],)) p2 = multiprocessing.Process(target=two\_test, args=(pipe[1],)) |

multiprocessing模块的Pipe类是用于进程间通信的，一者使用pipe.send()函数发送，一者使用pipe.recv()函数接收，特点在于recv()函数是阻塞的，程序走到这一步会等待另一方的send函数，等待到了才运行下一步程序。

1. 解决pytest类的参数传递问题

unittest类存在一个问题，就是不能接受参数，而我们的测试类是继承unittest因此需要进行一些改造。首先写一个类继承unittest类，使其可以接受参数，然后让待测试类继承它。具体的参数传递过程为：

1. multiProcessing.Pipe()，生成了管道pipe
2. 从args传递到one\_test和two\_test
3. 从parametrize传递到ParametrizedTestCase类中，传递给关键字黑色的param，
4. 黑色的param在testcase\_klass()处使用它的构造函数，也就是上面那个\_\_init\_\_，传入紫色的param中，在\_\_init\_\_中就是传入了self.param中

至此，param的参数传入到了self.param中，在TestOne和TestTwo中就可以直接使用self.param了。

|  |
| --- |
| *# ParametrizedTestCase.py*  **import** unittest  **class** ParametrizedTestCase(unittest.TestCase):**def** \_\_init\_\_(self, methodName=**'runTest'**, param=**None**):  super(ParametrizedTestCase, self).\_\_init\_\_(methodName)  self.param = param  @staticmethod  **def** parametrize(testcase\_klass, param=**None**):testloader = unittest.TestLoader()  testnames = testloader.getTestCaseNames(testcase\_klass)  suite = unittest.TestSuite()  **for** name **in** testnames:  suite.addTest(testcase\_klass(name, param=param))  **return** suite |

|  |
| --- |
| *# twoTest.py*  **import** unittest **import** multiprocessing **import** time  **from** ParametrizedTestCase **import** \* **class** TestOne(ParametrizedTestCase):  **def** test\_a(self):  i = 1  print(**"proc1 send\*\*\*: %s"** % (i))  self.param.send(i)  time.sleep(5)  print(**"do something @proc1"**)  print(**"proc1 rev \*\*\*:"**, self.param.recv()) **class** TestTwo(ParametrizedTestCase):  **def** test\_b(self):  i = 2  print(**"proc2 rev ---:"**, self.param.recv())  print(**"do something @proc2"**)  time.sleep(1)  print(**"proc2 send---: %s"** % (i))  self.param.send(i)  **def** one\_test(pipe):  suite = unittest.TestSuite()  suite.addTest(ParametrizedTestCase.parametrize(TestOne, param=pipe))  unittest.TextTestRunner(verbosity=2).run(suite)  **def** two\_test(pipe):  suite = unittest.TestSuite()  suite.addTest(ParametrizedTestCase.parametrize(TestTwo, param=pipe))  unittest.TextTestRunner(verbosity=2).run(suite)   **if** \_\_name\_\_ == **"\_\_main\_\_"**:  i = 0  max\_times = 100  **while** i<max\_times:  print(**"Start running test"** + str(i) +**"--------------------- "**)  pipe = multiprocessing.Pipe()  p1 = multiprocessing.Process(target=one\_test, args=(pipe[0],))  p2 = multiprocessing.Process(target=two\_test, args=(pipe[1],))   p1.start()  p2.start()   p1.join(time\_out\_sys\_max)  p2.join(time\_out\_sys\_max)   i = i+1 |

(4) 两手机同步

当AB手机的脚本中都包含多个测试函数且一一对应的时候，就需要使用Pipe来同步两个手机的行为。此处以Xcall中A手机拨打B手机，B手机挂断为例。

|  |
| --- |
| A手机流程 |
| 1. A手机调用Pipe的发送信息命令； 2. A进行它的工作（拨打B的电话）； 3. A手机调用Pipe的接收信息命令，在接收到信息前，A阻塞。 |

|  |
| --- |
| B手机流程 |
| 1. B手机调用Pipe的接收信息命令，在接收到信息前，B阻塞（目的，等待A开始拨打电话等行为）； 2. B进行它的工作（挂断A的电话）； 3. B手机调用Pipe的发送信息命令。 |

这样就可以保证，在A开始工作之后B才能开始工作，B结束工作之后，A才能结束工作。防止某一测试案例出错之后，后续测试案例的顺序变化，继而无法执行。

1. 验证执行结果
2. 以activity是否跳转回原本的activity的方式验证

|  |
| --- |
| **def** verify\_activity\_jump(self):  before\_activity = self.driver.current\_activity  *# do something* after\_activity = self.driver.current\_activity  self.driver.assertEquals(before\_activity, after\_activity)  在进行默写自动化操作之前，记录下当前的activity，然后在进行某些自动化操作之后，获得当前的activity，并用断言的方式比较。 |

1. 以某个元素是否存在的方式验证

|  |
| --- |
| **def** verify\_in\_one\_call(test\_evm):  **try**:  tv\_call\_state = test\_evm.driver.find\_element\_by\_id(tv\_call\_state\_id)  **except** Exception:  **return False  return True**  尝试寻找id为tv\_call\_state\_id的元素，如果没有找到，就说明这个元素不存在，返回False，如果找到了就返回True。 |

1. 以某个元素的内容验证

|  |
| --- |
| *# 确认通话结束 # 判断依据：不存在“通话状态”或者状态为正在结束通话。* **def** verify\_end(test\_evm):  i = 0  **while** i<500:  i = i+1  sleep(0.1)  **try**:  tv\_call\_state = test\_evm.driver.find\_element\_by\_id(tv\_call\_state\_id)  text = tv\_call\_state.text  **except**:  **continue  if** (text == **"正在结束通话"**):  **return True  return False**  尝试寻找id为tv\_call\_state \_id的对象，并同时取出它的内容，如果找到了这个对象且内容为“正在结束通话”，则返回True，否则经过50秒的找寻之后，返回False。  ctrip.android.xcall:id/btn\_hangup\_before  ctrip.android.xcall:id/tv\_call\_state  （结束通话时变为：正在结束通话） |

五、附录

Appium官网：<http://appium.io/>

Appium官方文档：[http://appium.io/slate/en/tutorial/android.html?ruby#](http://appium.io/slate/en/tutorial/android.html?ruby)

Appium中文文档：<https://testerhome.com/wiki/appiumdoccn>