# 环境搭建

## Android环境搭建

### \*\*jdk-1.8u66-windows-x64

#### 默认安装

C:\Program Files\Java\jre1.8.0\_66

C:\Program Files\Java\jdk1.8.0\_66

#### 设置三个环境变量

【JAVA\_HOME】

C:\Program Files\Java\jdk1.8.0\_66

【CLASSPATH】

%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar;

【PATH】

%JAVA\_HOME%\bin

%JAVA\_HOME%\jre\bin

#### 确认安装成功

C:\Users\ly>**java -version**

java version "1.8.0\_66"

Java(TM) SE Runtime Environment (build 1.8.0\_66-b18)

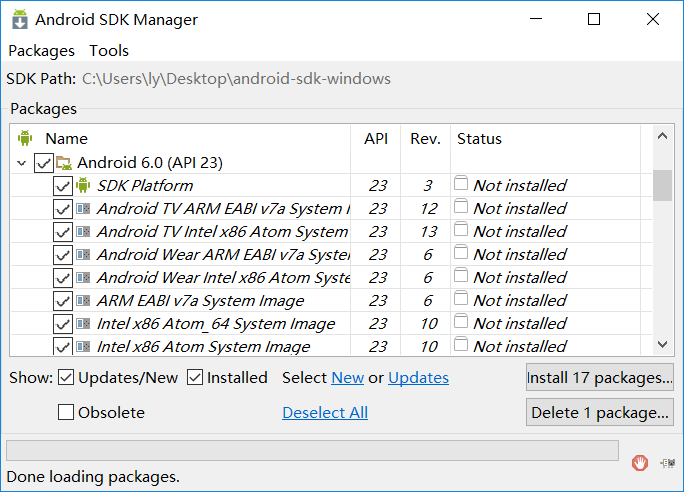
Java HotSpot(TM) 64-Bit Server VM (build 25.66-b18, mixed mode)

### \*\*android-sdk\_r24.3.4-windows

这个是做 android 测试和开发的必备环境

#### 双击SDK Manager

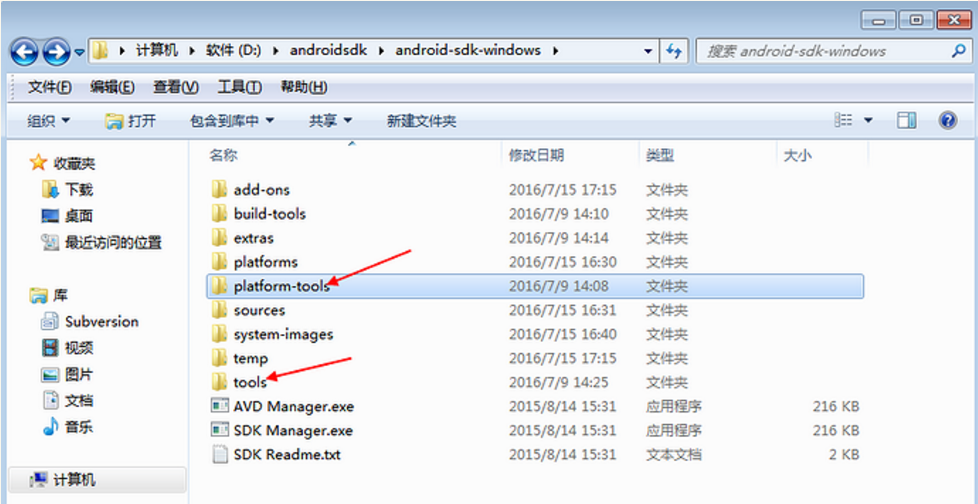
（手机上看到安卓版本6.0.1）勾选你要现在的 API 版本和对应的 android 版本



#### android-sdk 环境变量

1. 在系统变量新建：ANDROID\_HOME，对应变量值为： D:\androidsdk\android-sdk-windows（sdk 安装路径）

2. path 添加两个变量，将以下箭头所指的两个文件路径添加到 path 里



3. path 里面添加

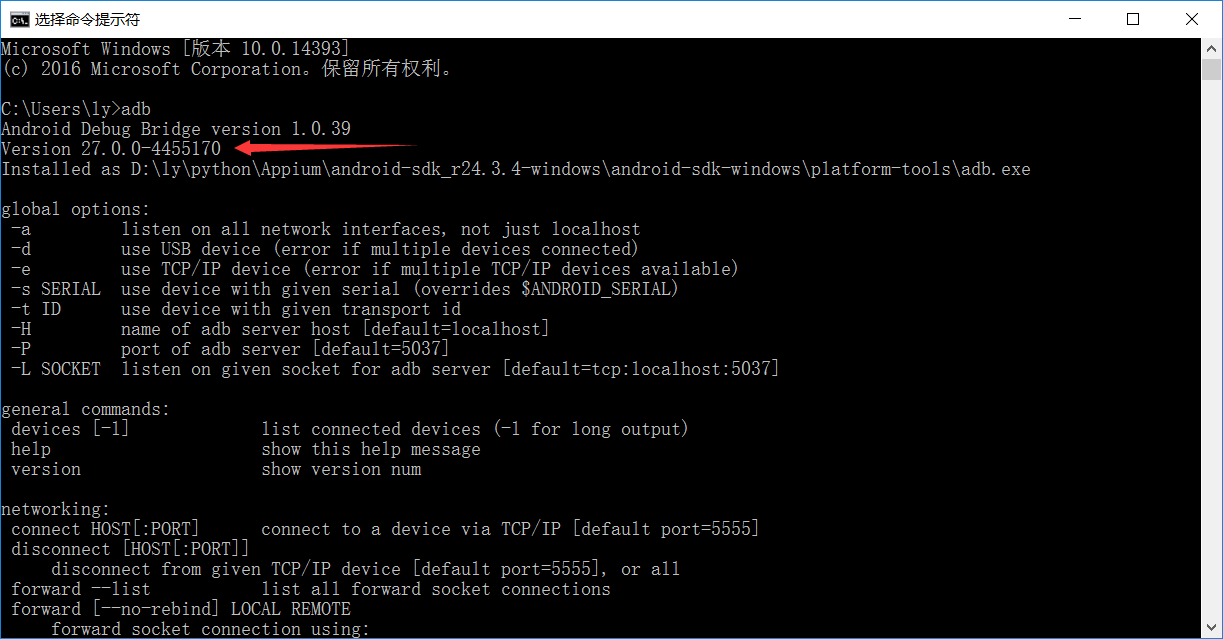
D:\androidsdk\android-sdk-windows\tools 和

D:\androidsdk\android-sdk-windows\platform-tools

#### adb环境

1.因为adb是在D:\androidsdk\android-sdk-windows\platform-tools这个目录下的，所以上面添加了环境变量后，可以直接在cmd里面运行了。

   2.在cmd输入adb可以查看对应版本号



#### 连接手机

1. 手机用数据线连电脑
2. 打开cmd输入：adb devices,当屏幕上出现一串字符，后面显示devices说明连接成功



到这里android的测试开发环境已经装好了

## android环境SDK Manager

### SDK Manager

1.双击打开SDK Manager界面

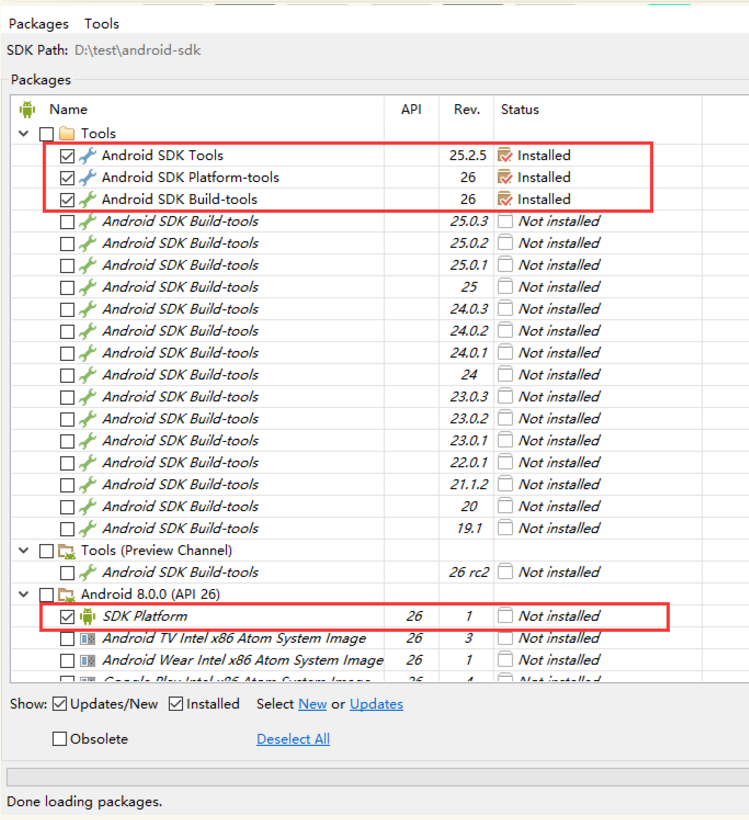
2.Tools里面前三个是必须下载的

--Android SDK Tools里面有各种实用工具，如：uiautomatorviewer，uiautomatorviewer，uiautomatorviewer

--Android SDK Platform-tools里面有adb,adb,adb

--Android SDK Build-tools里面有appt,aapt,appt

3.SDK Platform下个最新的就可以了，它可以向下兼容

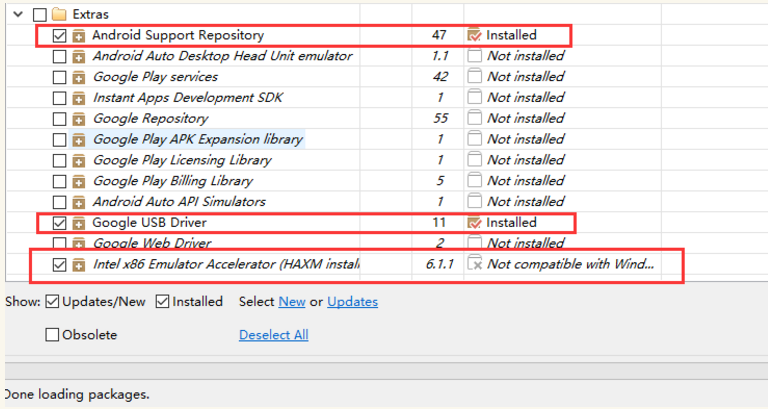


4.Extras这里选下面三个就行了

--Android Support Respository 安卓兼容库，必须安装

--最下面一个安卓模拟器加速器，这个随缘吧，安装不了莫强求

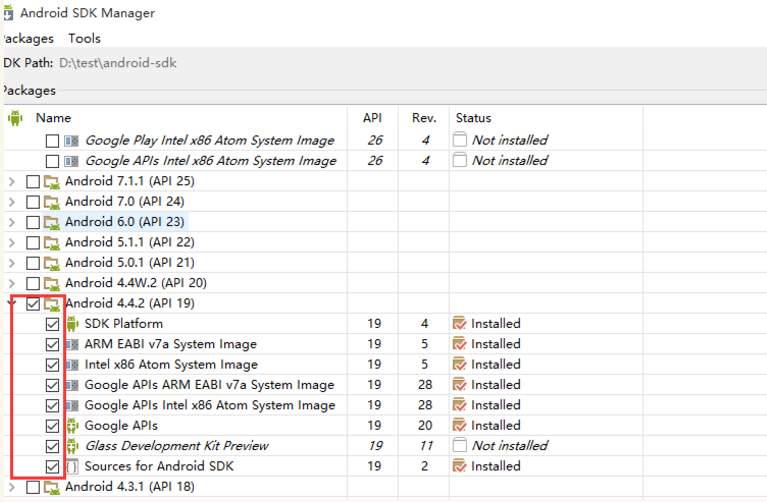
--其它的看心情了，想安装就安装



### Android版本

1.Android的版本这里我选择的是4.4.2（API 19）,其它的像Android 4.4w.2这种是手表的，还有一些带TV字样的是电视的，别乱勾选一些无关紧要的。

2.为了保险起见，这里4.4.2全家桶全部勾选了，其它的版本不需要的话，暂时不用勾选了。





## appium环境

环境装好后，可以用真机连电脑，也可以用android-sdk里面的模拟器（当然这个模拟器不是很好用），我一般喜欢真机，真机比较快。

### 必备软件安装

小编的环境是Windows 7版本 64位系统（32位的同学自己想办法哦）

    1.jdk1.6.0 （64位）

    2.android-sdk\_r24.3.4-windows

    3.python:2.7（3.6也可以）

    4.appium：1.4.13.1

    5.Node.js：node-v4.4.7-x64

6.Appium-Python-Client

### Appium的加载流程



### Python安装

1.小编的电脑操作系统：win7 64位系统

    2.下载Python安装包，选择2.7版本和3.6版本都可以

官网下载地址：https://www.python.org/15

    3.Python安装，双击傻瓜式安装（别安装在c盘哦）

    4.小编的安装目录在d盘：D:\python

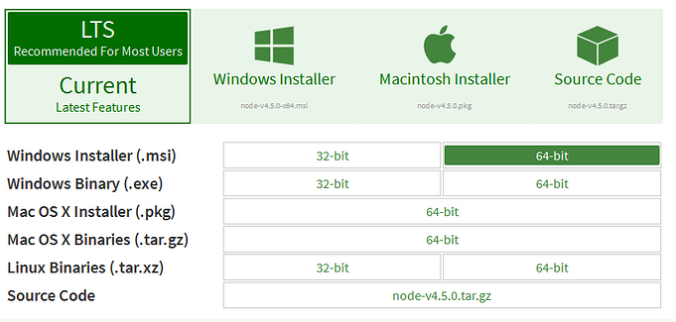
    5.安装完成后，看下这个目录D:\python\Scripts，有没pip.exe和easy\_install.exe（一般都有）

    6.将D:\python和D:\python\Scripts，添加到环境变量path下

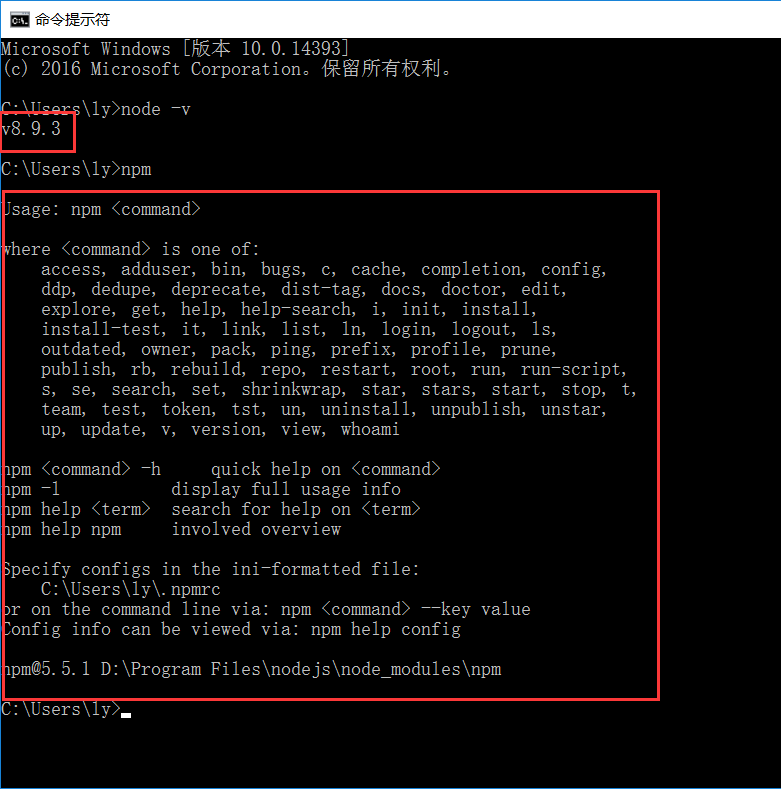
7.打开cmd输入python,出现版本号，然后输入print("hello world!")

### \*\*安装node.js

1. 下载官网地址：<https://nodejs.org/en/download/23（不会下载的在群文件找吧appium交流QQ群：512200893>）



1. 下载后一路傻瓜式安装，安装完成后，运行cmd，输入node –v查看版本号，然后输入npm



3.出现如上图信息，表示node.js安装成功。npm是一个node包管理和分发工具，有了npm，后面就可以输入指令在线安装appium(打开

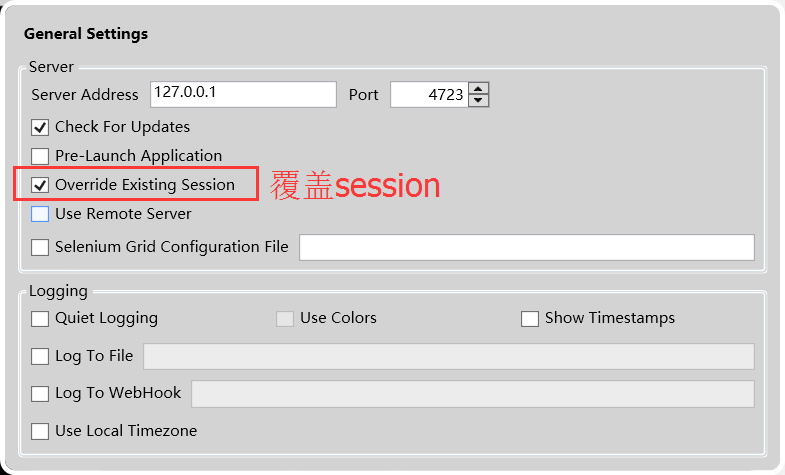
cmd输入：npm install –g appium但是一般不推荐这种，下载比较慢，所以用下面这种客户端安装)

### \*\*安装appium

1 下载安装地址：<https://bitbucket.org/appium/appium.app/downloads/16>

2 直接双击appium-installer.exe文件安装就好

记得要设置覆盖session



### 安装.net framework

    1.Appium是用.net开发的，所以需要安装.net framework4.5，下载地址：https://www.microsoft.com/zh-cn/download/details.aspx?id=3065332

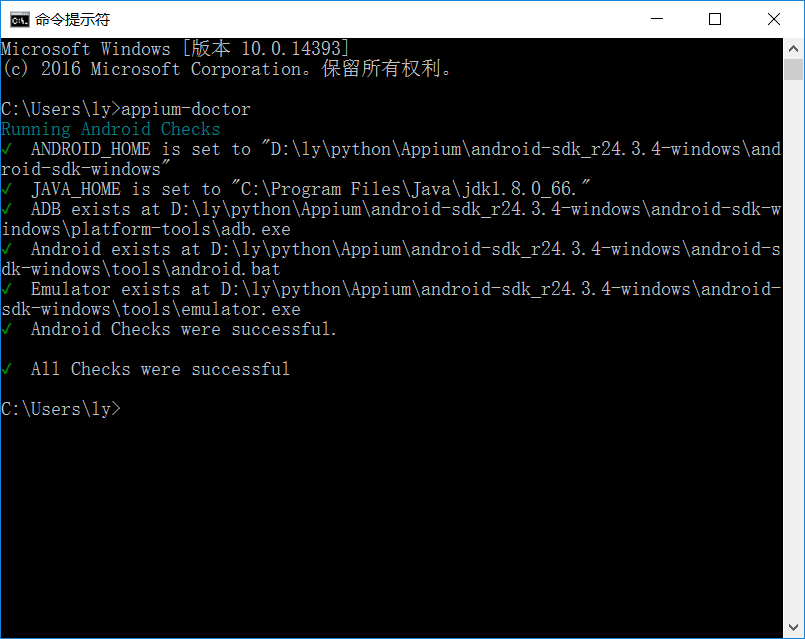
### appium-doctor

1. appium安装好后，找到这个文件目录

D:\Program Files (x86)\Appium\node\_modules\.bin

2.将上面的地址添加到环境变量path下

3.打卡cmd，输入appium-doctor,检查环境是否OK，出现如下图所示，说明环境OK



### 安装Appium-Python-Client

**Cmd里输入**

**pip install Appium-Python-Client**

Successfully installed Appium-Python-Client-0.25

### 问题确认

问题

1. error: Failed to start an Appium session, err was: Error: Requested a new session but one was in progress

【解决办法】

之前的会话没有关闭，然后你又运行了测试实例，也没有设置覆盖.   
解决:   
1. 重新停止appium服务，开启Appium服务   
2. 在Genarel Setting那里设置覆盖Session，重启Appium

测试结束在AfterClass加driver.quit()

## pycharm环境

## android模拟器AVD

### 模拟器配置

1.双击启动AVD Manager，进入配置界面

**D:\ly\python\Appium\android-sdk\_r24.3.4-windows\android-sdk-windows**

**下的AVD Manager.exe**

2.点Create按钮创建

3. 配置模拟器基本信息

--AVD Name:设备名称，自己定义一个，用英文（不要用中文）

--Device:设备型号，如下图选项意思是，4.7寸手机，768\*1280分辨率，屏幕密度：xhdpi

--Target:选对应API版本号，Android版本号4.4.2，对应API版本19

--CPU/ABI:cpu型号

--Keyboard:这个最好勾选，带键盘按钮

--Skin:皮肤设置，随意设置

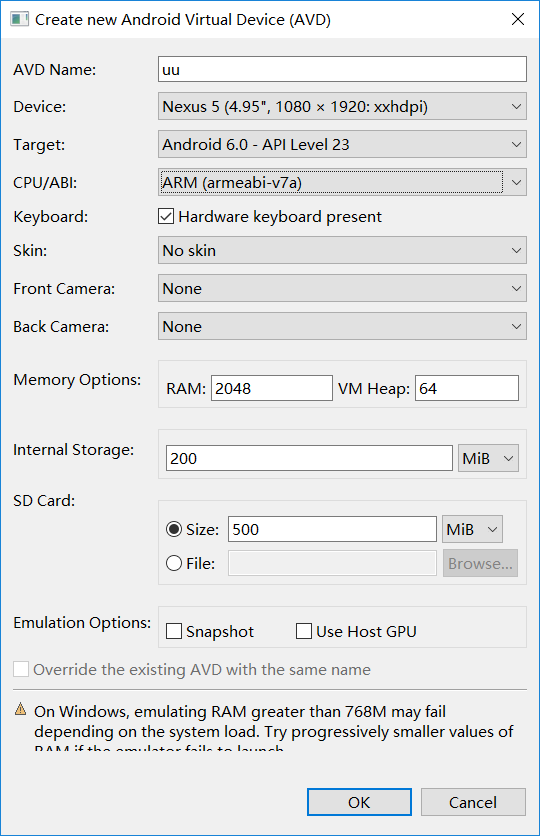
--Front Camera:前置摄像头，设置为None

--Back Camera:后置摄像头，设置为None

--Memory Option :RAM运行内存，设置512就行，VM Heap这个是app的运行内存默认设置的64就行

--Internal Storage:手机本身的存储卡大小

--SD Card：外部SD卡的内存



### 启动模拟器

选中上面创建成功的模拟器，点**start**按钮，然后点**Launch**按钮就能启动了

然后跟平常操作手机一样点点点了，接下来用adb工具检查下设备，打开cmd输入

>**adb devices**

模拟器起来黑屏，无解

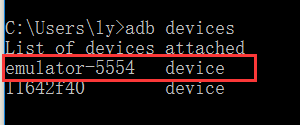
## adb使用（必会）

学android测试，adb是必学的，有几个常用的指令需要熟练掌握

### 检查设备

1.如何检查手机（或模拟器）是连上电脑的，在cmd输入：

>**adb devices**



2.一定要看到设备名称，然后接着是device(如果看到这里是offline，那就是adb端口被占了，打开电脑进程管理杀掉adb相关的，重启设备就行)

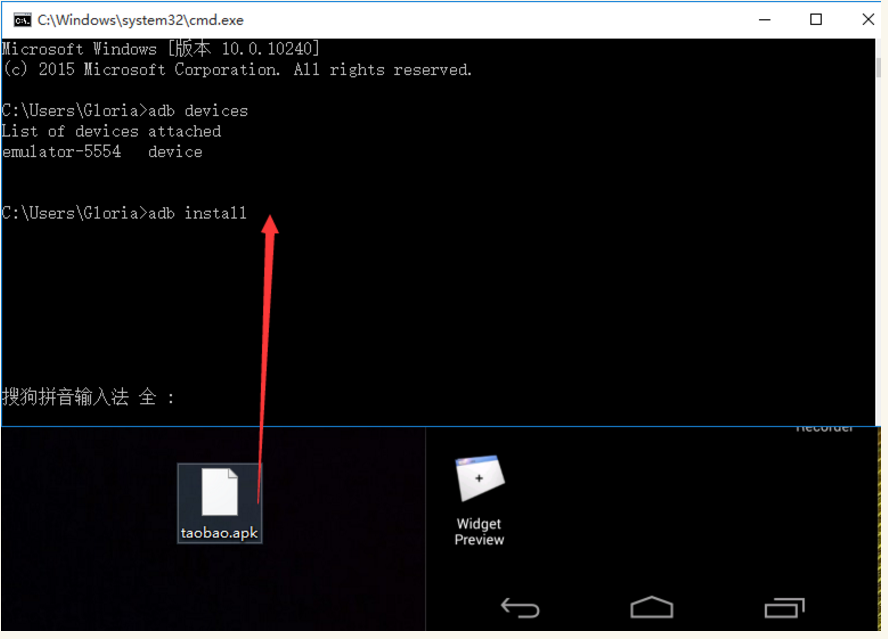
### 模拟器安装app

1.如何给电脑上的android模拟器安装app呢？

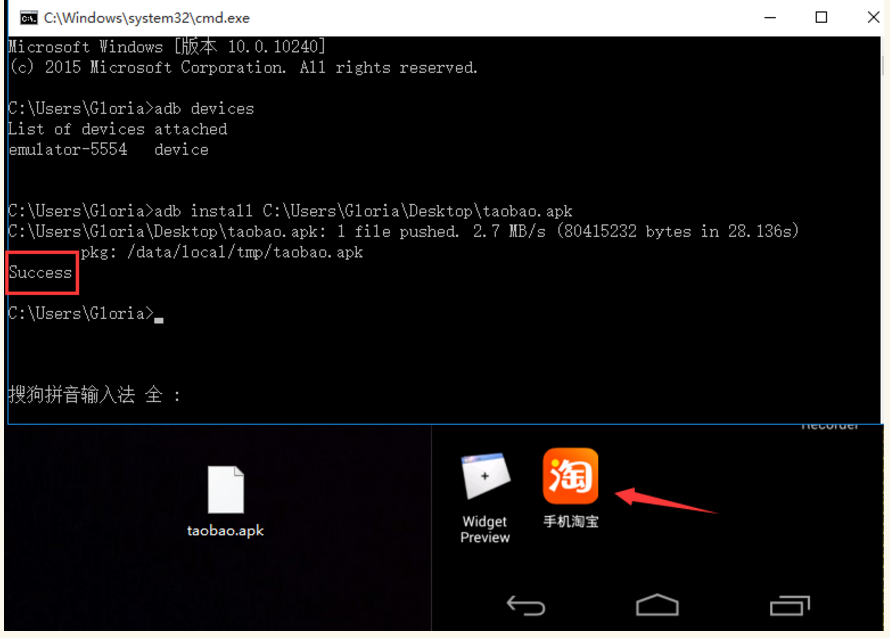
第一步：先下载apk文件包放到桌面上

第二步：cmd端口输入adb install app的路径

第三步：把apk包拖到cmd窗口，回车就能安装了



2.安装成功后看到Success，模拟器上会出现淘宝的app



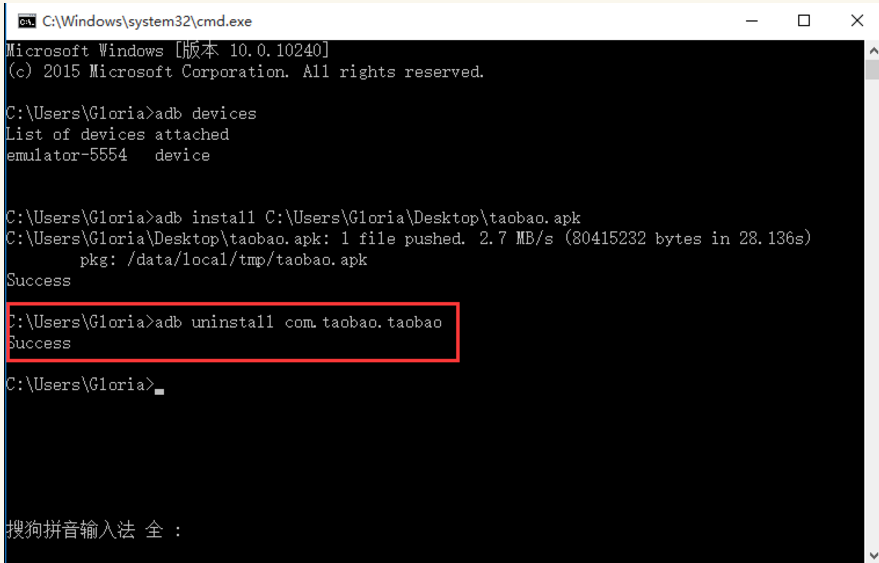
### 模拟器卸载app

1.在cmd输入以下指令，可以删除app

>adb uninstall 包名

（这里是app的包名，不是文件名，包名用2.1章节的的aapt工具查看）

2. 以淘宝为例，淘宝的包名是：com.taobao.taobao



### 其它的几个指令

1.杀掉adb进程

  adb kill-server

2.重启adb服务

  adb start-server

3.重启手机 adb reboot

4.进shell模式

 adb shell

5.挂载

 adb remount

6.从电脑发文件到手机

adb push <本地路径> <远程路径>

7.从手机下载文件到本地

adb pull <远程路径> <本地路径>

8.输出日志

第一种：输出到手机存储卡

adb logcat > /sdcard/mylogcat.txt

第二种：输出到电脑上

adb logcat > D:/Temp/1.txt（1.txt必须在电脑上存在，才能写入logcat内容）

# API详解

主要讲appium各api的用法，和各种特殊场景的处理方法

## 真机启动APP（淘宝）

环境已经搭建好了，接下来就是需要启动APP，如何启动app呢？首先要获取包名，然后获取launcherActivity。小编这里主要给大家推荐一个sdk自带的实用工具aapt。

aapt即Android Asset Packaging Tool，在SDK的build-tools目录下。该工具可以查看apk包名和launcherActivity，当然还有更多的功能，有兴趣的可以查看相关资料。

### 下载aapt

~~1.在android-sdk里面双击SDK-manager,下载buidl-tools~~

~~D:\ly\python\Appium\android-sdk\_r24.3.4-windows\android-sdk-windows~~

~~2.勾选build-tools，选最新版本（参考1.2章节详细介绍）~~

前面已经安装好了

3.下载完成后，在D:\ly\python\Appium\android-sdk\_r24.3.4-windows\android-sdk-windows\build-tools\27.0.2目录下找到aapt.exe，将这个路径设置环境变量，添加到path下

4.打开cmd。输入**aapt**出现如下界面，说明环境OK了

### 获取apk包名

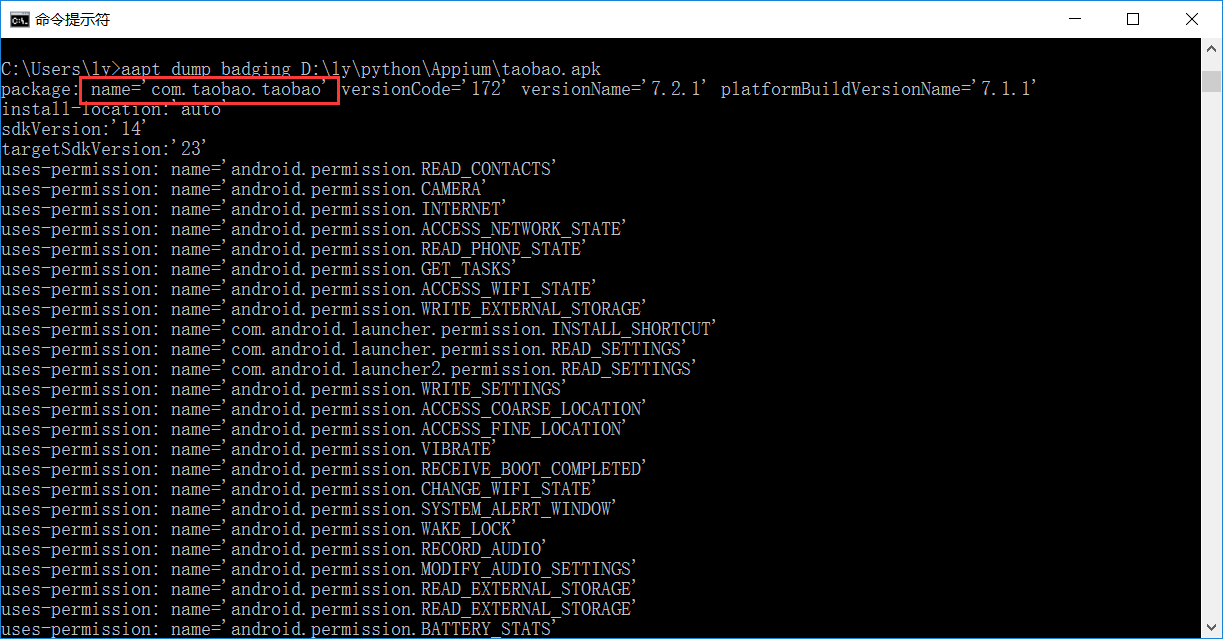
    1.将准备测试的APK放到D盘某个目录，如D:\test

    2.打开cmd,输入指令aapt dump badging D:\test\xxx.apk(APK的全名，如手机淘宝.apk)

（注：老司机可以直接把apk放在桌面上，输入aapt dump badging 后把apk拖到cmd框）

**aapt dump badging D:\ly\python\Appium\taobao.apk**

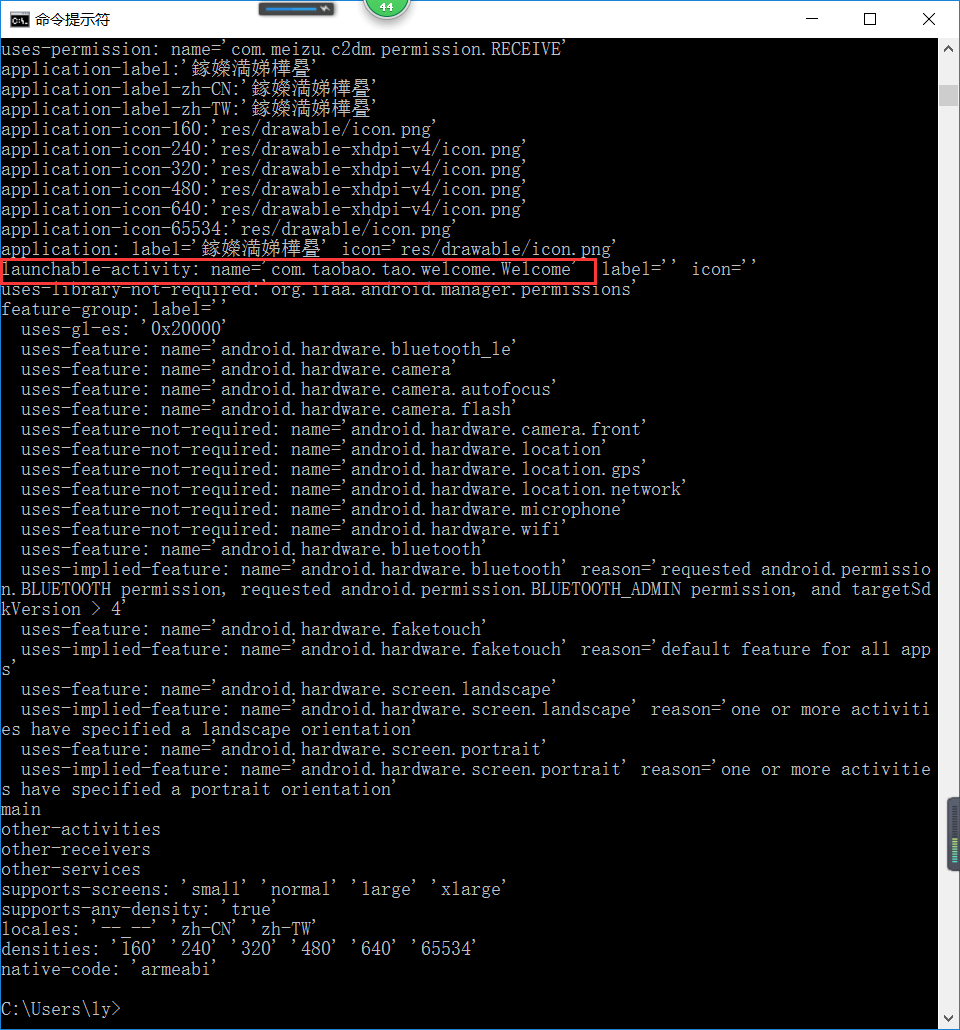
3. 以手机淘宝.apk为例，如下图, 这里就可以看到apk的包名：**com.taobao.taobao**



### 获取launcherActivity

    1.接着上一步操作，cmd屏幕拖到中间l找到launcherActivity

    2.这里可以看到，淘宝的launcherActivity值为**com.taobao.tao.welcome.Welcome**



### 写脚本

    1platformName：这里是声明android还是ios的环境

    2.deviceName：手机设备名称，通过adb devices查看

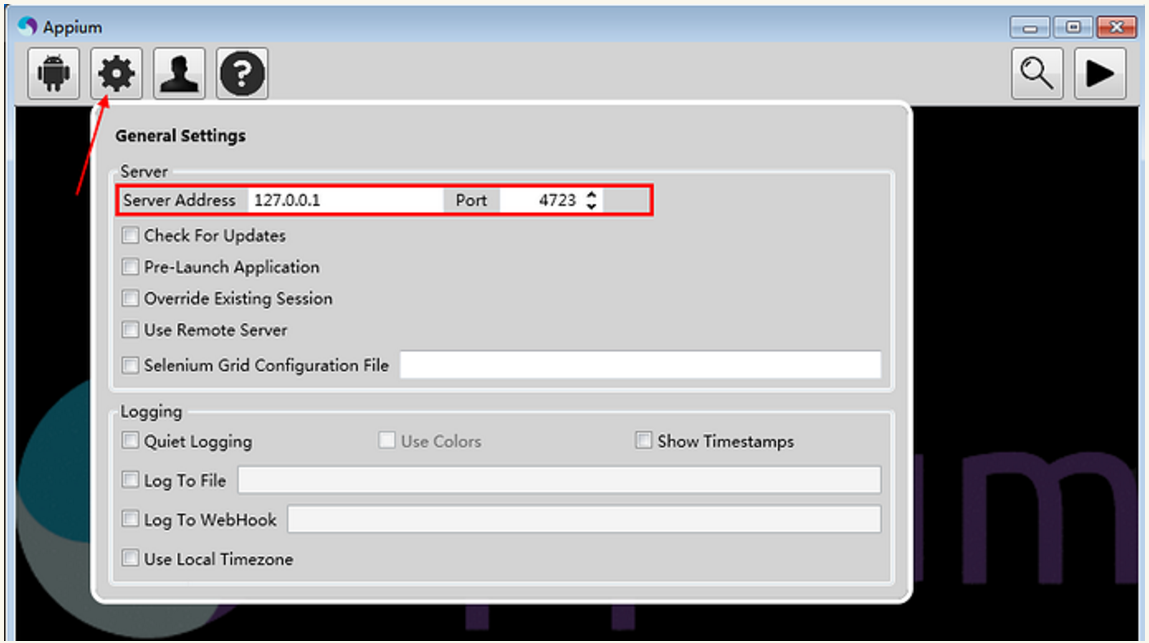
    3.platformVersion：android系统的版本号

    4.appPackage：apk包名

5.appActivity：apk的launcherActivity

|  |
| --- |
| # coding:utf-8 **from** appium **import** webdriver desired\_caps = {  # 这里是声明android还是ios的环境  'platformName': 'android',  # 手机设备名称，通过adb devices查看  'deviceName': '11642f40',  # android系统的版本号  'platformVersion': '6.0',  # apk包名  'appPackage': 'com.taobao.taobao',  # apk的launcherActivity  'appActivity': 'com.taobao.tao.welcome.Welcome'  } driver = webdriver.Remote('http;//127.0.0.1:4723/wd/hub', desired\_caps) |

这里的地址是下图中配置来的



### 运行appium

1.启动appium,右上角点三角形按钮，变成正方形，就是启动状态。

2.接下来准备一台android手机，一根数据线，确认手机连上电脑(注：手机上要安装淘宝的APP)

3.在pycharm运行脚本，随后在手机上会弹出安装下面两个软件的提示，安装后，桌面上多两个图标。那么恭喜你启动成功！

4.接着会看到淘宝app已经启动啦，有木有小激动~~

## 元素定位uiautomatorviewer

本篇主要介绍如何使用uiautomatorviewer，通过定位到页面上的元素，然后进行相应的点击等操作.uiautomatorviewer是android-sdk自带的一个元素定位工具，非常简单好用，使用uiautomatorviewer，你可以检查一个应用的UI来查看应用的布局和组件以及相关的属性。

### 启动uiautomatorviewer.bat

1.双击打开

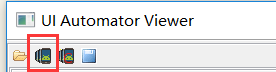
D:\ly\python\Appium\android-sdk\_r24.3.4-windows\android-sdk-windows\tools\**uiautomatorviewer.bat**

### 连接手机

1.cmd打开输入adb devices,确认手机已连上

    2.打开手机淘宝页面，让屏幕处于点亮状态

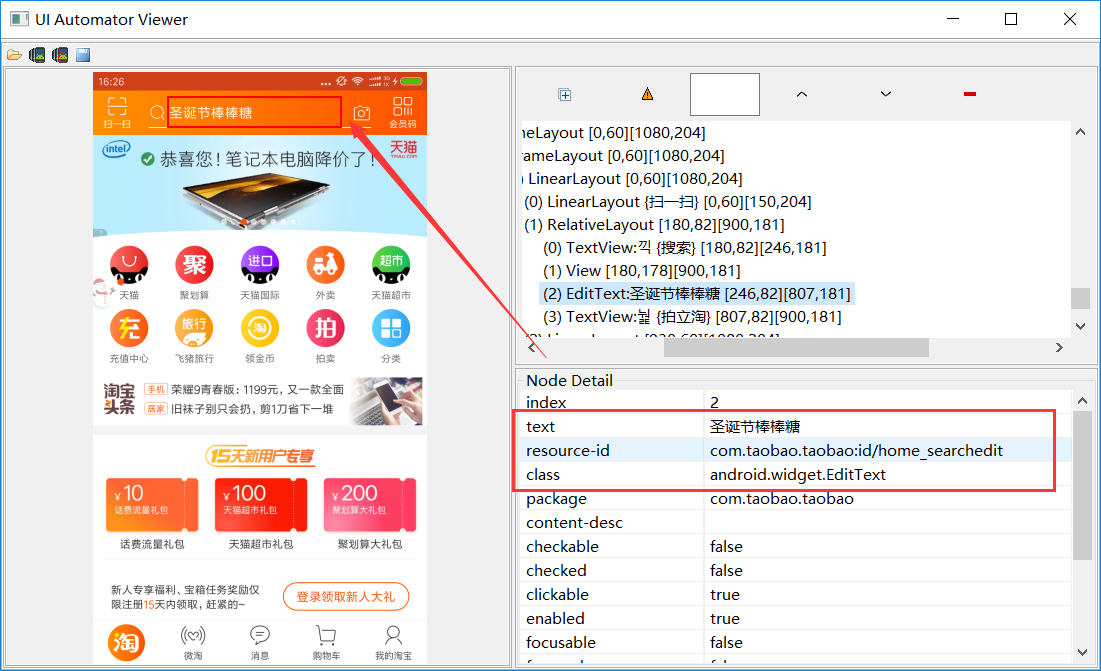
    3.点左上角安卓机器人按钮Devices Screenshot按钮刷新页面



### 定位元素

1.移动鼠标到需要定位的元素上，如搜索输入框

2.右下角可以看到元素对应的属性



text:圣诞节棒棒糖

resource-id: com.taobao.taobao:id/home\_searchedit

class: android.widget.EditText

### 点搜索框

    1.前面一骗启动app后，休眠五秒，等待页面加载完成

    2.通过id来定位到搜索框，然后点击

### 代码

|  |
| --- |
| # coding:utf-8 **from** appium **import** webdriver **import** time desired\_caps = {  # 这里是声明android还是ios的环境  'platformName': 'android',  # 手机设备名称，通过adb devices查看  'deviceName': '11642f40',  # android系统的版本号  'platformVersion': '6.0',  # apk包名  'appPackage': 'com.taobao.taobao',  # apk的launcherActivity  'appActivity': 'com.taobao.tao.welcome.Welcome'  } driver = webdriver.Remote('http://127.0.0.1:4723/wd/hub', desired\_caps) # 休眠5秒等待页面加载完成 time.sleep(5) driver.find\_element('id', 'com.taobao.taobao:id/home\_searchedit').click() |

### 元素定位

1.appium的webdriver提供了11种常规元素定位方法，在selenium的基础上扩展了三个，可以在pycharm里面输入driver.find\_element\_by然后会自动匹配出来

2.多的三种如下，在后面的会详细介绍

driver.find\_element\_by\_accessibility\_id()

driver.find\_element\_by\_android\_uiautomator()

driver.find\_element\_by\_ios\_uiautomation()

（第三个是ios的可以暂时不用管）

uiautomatorviewer是android sdk自带的，下篇介绍appium自带个元素定位工具：Inspector

## --appium Inspector（mac）

appium Inspector在windows上不太好用，问题比较多，一般在Mac上用的比较多，Mac版的功能强大一些！

## Remote远程控制

### 设置IP

1.打开appium>General Setting界面

2.这里用个appium默认的服务端地址127.0.0.1端口4723，一般在自己机器上调试，无需修改

### 访问地址

    1.代码里面的那个地址到底指向的是哪里呢？

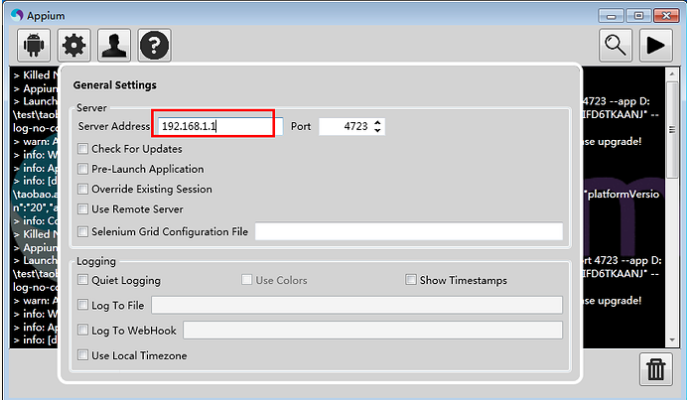
启动appium服务后，在浏览器输入：http://127.0.0.1:4723/wd/hub。出现如下图所示，说明服务启动成功，可以把appium看出是一个服务端。



### 配置测试机

    1.一台工作电脑，一台跑自动化测试的电脑，如何用工作的电脑远程控制自动化测试的那台电脑呢？

    2.测试电脑上的appium服务地址改成本机IP地址如：192.168.1.1（敲黑板，记重点！这里要是本机的ip地址）



3.启动测试电脑上的appium。

### 远程操作

    1.在自己工作电脑上打开浏览器输入：192.168.1.1:4723/wd/hub。这一步很重要，一定要看到如下界面，确认远程链接成功。



2.脚本里面的代码修改下地址



3.测试机上环境准备好后，在本机上运行脚本，于是测试机上可以自动运行了。

（其实没多大用处，了解下就行了，不同机器运行，一般最后用jenkins集成，远程客户端运行了，可以用git同步代码）

## 输入中文

本篇通过屏蔽软键盘，绕过手机的软键盘方法，解决中文输入问题。

一、定位搜索

    1.打开淘宝点搜索按钮，进入到搜索页面

    2.然后定位到搜索框后用sendkeys方法输入“hao”,这里定位元素用前面讲的uiautomatorviewer工具就可以了

    3.脚本如下图

# unicodeKeyboard是使用unicode编码方式发送字符串  
 'unicodeKeyboard': True,  
 # resetKeyboard是将键盘隐藏起来  
 'resetKeyboard': True

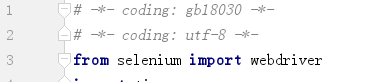
输入改为中文

driver.find\_element\_by\_id("com.taobao.taobao:id/searchEdit").send\_keys(u"测试部落")

|  |
| --- |
| # coding:utf-8 # coding:gb18030 **from** appium **import** webdriver **import** time desired\_caps = {  # 这里是声明android还是ios的环境  'platformName': 'Android',  # 手机设备名称，通过adb devices查看  'deviceName': '11642f40',  # android系统的版本号  'platformVersion': '6.0.1',  # apk包名  'appPackage': 'com.taobao.taobao',  # apk的launcherActivity  'appActivity': 'com.taobao.tao.welcome.Welcome',  # unicodeKeyboard是使用unicode编码方式发送字符串  'unicodeKeyboard': True,  # resetKeyboard是将键盘隐藏起来  'resetKeyboard': True  } driver = webdriver.Remote('http://127.0.0.1:4723/wd/hub', desired\_caps) # 休眠15秒等待页面加载完成 time.sleep(10) driver.find\_element\_by\_id("com.taobao.taobao:id/home\_searchedit").click() time.sleep(2) driver.find\_element\_by\_id("com.taobao.taobao:id/searchEdit").click() driver.find\_element\_by\_id("com.taobao.taobao:id/searchEdit").send\_keys(u"测试部落") driver.quit() |

.如果显示中文字符为乱码时：

前面加上“#--coding:gb18030--”   如下图所示：



### 手动输入法还原设置

    1.当运行上面的脚本后，发现手动去输入时候，无法调出软键盘了，如何恢复呢？

    2.打开手机设置，找到输入法设置选项，会发现默认的输入法被改成appium的输入法了。所以只需把这里的设置，恢复成原来输入法就OK拉。

## appium\_API(带翻译)

### contexts

contexts(self):

Returns the contexts within the current session.

返回当前会话中的上下文，使用后可以识别H5页面的控件

:Usage:

driver.contexts

用法 driver.contexts

### current\_context

current\_context(self):

Returns the current context of the current session.

返回当前会话的当前上下文

:Usage:

driver.current\_context

用法driver. current\_context

### context

context(self):

Returns the current context of the current session.

返回当前会话的当前上下文。

:Usage:

driver.context

用法driver. Context

### find\_element\_by\_ios\_uiautomation

find\_element\_by\_ios\_uiautomation(self, uia\_string):

Finds an element by uiautomation in iOS.

通过iOS uiautomation查找元素

:Args:

- uia\_string - The element name in the iOS UIAutomation library

:Usage:

driver.find\_element\_by\_ios\_uiautomation('.elements()[1].cells()[2]')

用法dr. find\_element\_by\_ios\_uiautomation(‘elements’)

### find\_element\_by\_accessibility\_id

find\_element\_by\_accessibility\_id(self, id):

Finds an element by accessibility id.

通过accessibility id查找元素

:Args:

- id - a string corresponding to a recursive element search using the

Id/Name that the native Accessibility options utilize

:Usage:

driver.find\_element\_by\_accessibility\_id()

用法driver.find\_element\_by\_accessibility\_id(‘id’)

### scroll

scroll(self, origin\_el, destination\_el):

Scrolls from one element to another

从元素origin\_el滚动至元素destination\_el

:Args:

- originalEl - the element from which to being scrolling

- destinationEl - the element to scroll to

:Usage:

driver.scroll(el1, el2)

用法 driver.scroll(el1,el2)

### drag\_and\_drop

drag\_and\_drop(self, origin\_el, destination\_el):

Drag the origin element to the destination element

将元素origin\_el拖到目标元素destination\_el

:Args:

- originEl - the element to drag

- destinationEl - the element to drag to

用法 driver.drag\_and\_drop(el1,el2)

### tap

tap(self, positions, duration=None):

Taps on an particular place with up to five fingers, holding for a certain time

模拟手指点击（最多五个手指），可设置按住时间长度（毫秒）

:Args:

- positions - an array of tuples representing the x/y coordinates of

the fingers to tap. Length can be up to five.

- duration - (optional) length of time to tap, in ms

:Usage:

driver.tap([(100, 20), (100, 60), (100, 100)], 500)

用法 driver.tap([(x,y),(x1,y1)],500)

### swipe

swipe(self, start\_x, start\_y, end\_x, end\_y, duration=None):

Swipe from one point to another point, for an optional duration.

从A点滑动至B点，滑动时间为毫秒

:Args:

- start\_x - x-coordinate at which to start

- start\_y - y-coordinate at which to start

- end\_x - x-coordinate at which to stop

- end\_y - y-coordinate at which to stop

- duration - (optional) time to take the swipe, in ms.

:Usage:

driver.swipe(100, 100, 100, 400)

用法 driver.swipe(x1,y1,x2,y2,500)

### flick

flick(self, start\_x, start\_y, end\_x, end\_y):

Flick from one point to another point.

按住A点后快速滑动至B点

:Args:

- start\_x - x-coordinate at which to start

- start\_y - y-coordinate at which to start

- end\_x - x-coordinate at which to stop

- end\_y - y-coordinate at which to stop

:Usage:

driver.flick(100, 100, 100, 400)

用法 driver.flick(x1,y1,x2,y2)

### pinch

pinch(self, element=None, percent=200, steps=50):

Pinch on an element a certain amount

在元素上执行模拟双指捏（缩小操作）

:Args:

- element - the element to pinch

- percent - (optional) amount to pinch. Defaults to 200%

- steps - (optional) number of steps in the pinch action

:Usage:

driver.pinch(element)

用法 driver.pinch(element)

### zoom

zoom(self, element=None, percent=200, steps=50):

Zooms in on an element a certain amount

在元素上执行放大操作

:Args:

- element - the element to zoom

- percent - (optional) amount to zoom. Defaults to 200%

- steps - (optional) number of steps in the zoom action

:Usage:

driver.zoom(element)

用法 driver.zoom(element)

### reset

reset(self):

Resets the current application on the device.

重置应用(类似删除应用数据)

用法 driver.reset()

### hide\_keyboard

hide\_keyboard(self, key\_name=None, key=None, strategy=None):

Hides the software keyboard on the device. In iOS, use `key\_name` to press a particular key, or `strategy`. In Android, no parameters are used.

隐藏键盘,iOS使用key\_name隐藏，安卓不使用参数

:Args:

- key\_name - key to press

- strategy - strategy for closing the keyboard (e.g., `tapOutside`)

driver.hide\_keyboard()

### keyevent

keyevent(self, keycode, metastate=None):

Sends a keycode to the device. Android only. Possible keycodes can be found in http://developer.android.com/reference/android/view/KeyEvent.html.

发送按键码（安卓仅有），按键码可以上网址中找到

:Args:

- keycode - the keycode to be sent to the device

- metastate - meta information about the keycode being sent

用法 dr.keyevent(‘4’)

### press\_keycode

press\_keycode(self, keycode, metastate=None):

Sends a keycode to the device. Android only. Possible keycodes can be found in http://developer.android.com/reference/android/view/KeyEvent.html.

发送按键码（安卓仅有），按键码可以上网址中找到

:Args:

- keycode - the keycode to be sent to the device

- metastate - meta information about the keycode being sent

用法 driver.press\_ keycode(‘4’)

dr.keyevent(‘4’)与driver.press\_ keycode(‘4’) 功能实现上一样的，都是按了返回键

### long\_press\_keycode

long\_press\_keycode(self, keycode, metastate=None):

Sends a long press of keycode to the device. Android only. Possible keycodes can be

found in http://developer.android.com/reference/android/view/KeyEvent.html.

发送一个长按的按键码（长按某键）

:Args:

- keycode - the keycode to be sent to the device

- metastate - meta information about the keycode being sent

用法 driver.long\_press\_keycode(‘4’)

### current\_activity

current\_activity(self):

Retrieves the current activity on the device.

获取当前的activity

用法 print(driver.current\_activity())

### wait\_activity

wait\_activity(self, activity, timeout, interval=1):

Wait for an activity: block until target activity presents or time out.

This is an Android-only method.

等待指定的activity出现直到超时，interval为扫描间隔1秒

即每隔几秒获取一次当前的activity

返回的True 或 False

:Agrs:

- activity - target activity

- timeout - max wait time, in seconds

- interval - sleep interval between retries, in seconds

用法driver.wait\_activity(‘.activity.xxx’,5,2)

### background\_app

background\_app(self, seconds):

Puts the application in the background on the device for a certain duration.

后台运行app多少秒

:Args:

- seconds - the duration for the application to remain in the background

用法 driver.background\_app(5) 置后台5秒后再运行

### is\_app\_installed

is\_app\_installed(self, bundle\_id):

Checks whether the application specified by `bundle\_id` is installed on the device.

检查app是否有安装

返回 True or False

:Args:

- bundle\_id - the id of the application to query

用法 driver.is\_app\_installed(“com.xxxx”)

### install\_app

install\_app(self, app\_path):

Install the application found at `app\_path` on the device.

安装app,app\_path为安装包路径

:Args:

- app\_path - the local or remote path to the application to install

用法 driver.install\_app(app\_path)

### remove\_app

remove\_app(self, app\_id):

Remove the specified application from the device.

删除app

:Args:

- app\_id - the application id to be removed

用法 driver.remove\_app(“com.xxx.”)

### launch\_app

launch\_app(self):

Start on the device the application specified in the desired capabilities.

启动app

用法 driver.launch\_app()

### close\_app

close\_app(self):

Stop the running application, specified in the desired capabilities, on the device.

关闭app

用法 driver.close\_app()

启动和关闭app运行好像会出错

### start\_activity

start\_activity(self, app\_package, app\_activity, \*\*opts):

Opens an arbitrary activity during a test. If the activity belongs to

another application, that application is started and the activity is opened.

This is an Android-only method.

在测试过程中打开任意活动。如果活动属于另一个应用程序，该应用程序的启动和活动被打开。

这是一个安卓的方法

:Args:

- app\_package - The package containing the activity to start.

- app\_activity - The activity to start.

- app\_wait\_package - Begin automation after this package starts (optional).

- app\_wait\_activity - Begin automation after this activity starts (optional).

- intent\_action - Intent to start (optional).

- intent\_category - Intent category to start (optional).

- intent\_flags - Flags to send to the intent (optional).

- optional\_intent\_arguments - Optional arguments to the intent (optional).

- stop\_app\_on\_reset - Should the app be stopped on reset (optional)?

用法 driver.start\_activity(app\_package, app\_activity)

### lock

lock(self, seconds):

Lock the device for a certain period of time. iOS only.

锁屏一段时间 iOS专有

:Args:

- the duration to lock the device, in seconds

用法 driver.lock()

### shake

shake(self):

Shake the device.

摇一摇手机

用法 driver.shake()

### open\_notifications

open\_notifications(self):

Open notification shade in Android (API Level 18 and above)

打系统通知栏（仅支持API 18 以上的安卓系统）

用法 driver.open\_notifications()

### network\_connection

network\_connection(self):

Returns an integer bitmask specifying the network connection type.

Android only.

返回网络类型 数值

Possible values are available through the enumeration `appium.webdriver.ConnectionType`

用法 driver.network\_connection

### set\_network\_connection

set\_network\_connection(self, connectionType):

Sets the network connection type. Android only.

Possible values:

Value (Alias) | Data | Wifi | Airplane Mode

-------------------------------------------------

0 (None) | 0 | 0 | 0

1 (Airplane Mode) | 0 | 0 | 1

2 (Wifi only) | 0 | 1 | 0

4 (Data only) | 1 | 0 | 0

6 (All network on) | 1 | 1 | 0

These are available through the enumeration appium.webdriver.ConnectionType`

设置网络类型

:Args:

- connectionType - a member of the enum appium.webdriver.ConnectionType

用法 先加载from appium.webdriver.connectiontype import ConnectionType

dr.set\_network\_connection(ConnectionType.WIFI\_ONLY)

ConnectionType的类型有

NO\_CONNECTION = 0

AIRPLANE\_MODE = 1

WIFI\_ONLY = 2

DATA\_ONLY = 4

ALL\_NETWORK\_ON = 6

### available\_ime\_engines

available\_ime\_engines(self):

Get the available input methods for an Android device. Package and activity are returned (e.g., ['com.android.inputmethod.latin/.LatinIME'])

Android only.

返回安卓设备可用的输入法

用法print(driver.available\_ime\_engines)

### is\_ime\_active

is\_ime\_active(self):

Checks whether the device has IME service active. Returns True/False.

Android only.

检查设备是否有输入法服务活动。返回真/假。

安卓

用法 print(driver.is\_ime\_active())

### activate\_ime\_engine

activate\_ime\_engine(self, engine):

Activates the given IME engine on the device.

Android only.

激活安卓设备中的指定输入法，设备可用输入法可以从“available\_ime\_engines”获取

:Args:

- engine - the package and activity of the IME engine to activate (e.g.,

'com.android.inputmethod.latin/.LatinIME')

用法 driver.activate\_ime\_engine(“com.android.inputmethod.latin/.LatinIME”)

### deactivate\_ime\_engine

deactivate\_ime\_engine(self):

Deactivates the currently active IME engine on the device.

Android only.

关闭安卓设备当前的输入法

用法 driver.deactivate\_ime\_engine()

### active\_ime\_engine

active\_ime\_engine(self):

Returns the activity and package of the currently active IME engine (e.g.,

'com.android.inputmethod.latin/.LatinIME').

Android only.

返回当前输入法的包名

用法 driver.active\_ime\_engine

### toggle\_location\_services

toggle\_location\_services(self):

Toggle the location services on the device. Android only.

打开安卓设备上的位置定位设置

用法 driver.toggle\_location\_services()

### set\_location

set\_location(self, latitude, longitude, altitude):

Set the location of the device

设置设备的经纬度

:Args:

- latitude纬度 - String or numeric value between -90.0 and 90.00

- longitude经度 - String or numeric value between -180.0 and 180.0

- altitude海拔高度- String or numeric value

用法 driver.set\_location(纬度，经度，高度)

### tag\_name

tag\_name(self):

This element's ``tagName`` property.

返回元素的tagName属性

经实践返回的是class name

用法 element.tag\_name()

### text

text(self):

The text of the element.

返回元素的文本值

用法 element.text()

### click

click(self):

Clicks the element.

点击元素

用法 element.click()

### submit

submit(self):

Submits a form.

提交表单

用法 暂无

### clear

clear(self):

Clears the text if it's a text entry element.

清除输入的内容

用法 element.clear()

### get\_attribute

get\_attribute(self, name):

详见[@chenhengjie123](https://testerhome.com/chenhengjie123) 的[超级链接](https://testerhome.com/topics/2606)

Gets the given attribute or property of the element.

1、获取 content-desc 的方法为 get\_attribute("name") ，而且还不能保证返回的一定是 content-desc （content-desc 为空时会返回 text 属性值）

2、get\_attribute 方法不是我们在 uiautomatorviewer 看到的所有属性都能获取的（此处的名称均为使用 get\_attribute 时使用的属性名称）：

可获取的：

字符串类型：

name(返回 content-desc 或 text)

text(返回 text)

className(返回 class，只有 API=>18 才能支持)

resourceId(返回 resource-id，只有 API=>18 才能支持)

This method will first try to return the value of a property with the

given name. If a property with that name doesn't exist, it returns the

value of the attribute with the same name. If there's no attribute with

that name, ``None`` is returned.

Values which are considered truthy, that is equals "true" or "false",

are returned as booleans. All other non-``None`` values are returned

as strings. For attributes or properties which do not exist, ``None``

is returned.

:Args:

- name - Name of the attribute/property to retrieve.

Example::

# Check if the "active" CSS class is applied to an element.

is\_active = "active" in target\_element.get\_attribute("class")

用法 暂无

### is\_selected

is\_selected(self):

Returns whether the element is selected.

Can be used to check if a checkbox or radio button is selected.

返回元素是否选择。

可以用来检查一个复选框或单选按钮被选中。

用法 element.is\_slected()

### is\_enabled

is\_enabled(self):

Returns whether the element is enabled.

返回元素是否可用True of False

用法 element.is\_enabled()

### find\_element\_by\_id

find\_element\_by\_id(self, id\_):

Finds element within this element's children by ID.

通过元素的ID定位元素

:Args:

- id\_ - ID of child element to locate.

用法 driver. find\_element\_by\_id(“id”)

### find\_elements\_by\_id

find\_elements\_by\_id(self, id\_):

Finds a list of elements within this element's children by ID.

通过元素ID定位,含有该属性的所有元素

:Args:

- id\_ - Id of child element to find.

用法 driver. find\_elements\_by\_id(“id”)

### find\_element\_by\_name

find\_element\_by\_name(self, name):

Finds element within this element's children by name.

通过元素Name定位（元素的名称属性text）

:Args:

- name - name property of the element to find.

用法 driver.find\_element\_by\_name(“name”)

### find\_elements\_by\_name

find\_elements\_by\_name(self, name):

Finds a list of elements within this element's children by name.

通过元素Name定位（元素的名称属性text），含有该属性的所有元素

:Args:

- name - name property to search for.

用法 driver.find\_element\_by\_name(“name”)

### find\_element\_by\_link\_text

find\_element\_by\_link\_text(self, link\_text):

Finds element within this element's children by visible link text.

通过元素可见链接文本定位

:Args:

- link\_text - Link text string to search for.

用法 driver.find\_element\_by\_link\_text(“text”)

### find\_elements\_by\_link\_text

find\_element\_by\_link\_text(self, link\_text):

Finds a list of elements within this element's children by visible link text

通过元素可见链接文本定位,含有该属性的所有元素

:Args:

- link\_text - Link text string to search for.

用法 driver.find\_elements\_by\_link\_text(“text”)

### find\_element\_by\_partial\_link\_text

find\_element\_by\_partial\_link\_text(self, link\_text):

Finds element within this element's children by partially visible link text.

通过元素部分可见链接文本定位

:Args:

- link\_text - Link text string to search for.

driver. find\_element\_by\_partial\_link\_text(“text”)

### find\_elements\_by\_partial\_link\_text

find\_elements\_by\_partial\_link\_text(self, link\_text):

Finds a list of elements within this element's children by link text.

通过元素部分可见链接文本定位,含有该属性的所有元素

:Args:

- link\_text - Link text string to search for.

driver. find\_elements\_by\_partial\_link\_text(“text”)

### find\_element\_by\_tag\_name

find\_element\_by\_tag\_name(self, name):

Finds element within this element's children by tag name.

通过查找html的标签名称定位元素

:Args:

- name - name of html tag (eg: h1, a, span)

用法 driver.find\_element\_by\_tag\_name(“name”)

### find\_elements\_by\_tag\_name

find\_elements\_by\_tag\_name(self, name):

Finds a list of elements within this element's children by tag name.

通过查找html的标签名称定位所有元素

:Args:

- name - name of html tag (eg: h1, a, span)

用法driver.find\_elements\_by\_tag\_name(“name”)

### find\_element\_by\_xpath

find\_element\_by\_xpath(self, xpath):

Finds element by xpath.

通过Xpath定位元素，详细方法可参阅http://www.w3school.com.cn/xpath/

:Args:

xpath - xpath of element to locate. "//input[@class='myelement']"

Note: The base path will be relative to this element's location.

This will select the first link under this element.

::

myelement.find\_elements\_by\_xpath(".//a")

However, this will select the first link on the page.

::

myelement.find\_elements\_by\_xpath("//a")

用法 find\_element\_by\_xpath(“//\*”)

### find\_elements\_by\_xpath

find\_elements\_by\_xpath(self, xpath):

Finds elements within the element by xpath.

:Args:

- xpath - xpath locator string.

Note: The base path will be relative to this element's location.

This will select all links under this element.

::

myelement.find\_elements\_by\_xpath(".//a")

However, this will select all links in the page itself.

::

myelement.find\_elements\_by\_xpath("//a")

用法find\_elements\_by\_xpath(“//\*”)

### find\_element\_by\_class\_name

find\_element\_by\_class\_name(self, name):

Finds element within this element's children by class name.

通过元素class name属性定位元素

:Args:

- name - class name to search for.

用法 driver. find\_element\_by\_class\_name(“android.widget.LinearLayout”)

### find\_elements\_by\_class\_name

find\_elements\_by\_class\_name(self, name):

Finds a list of elements within this element's children by class name.

通过元素class name属性定位所有含有该属性的元素

:Args:

- name - class name to search for.

用法 driver. find\_elements\_by\_class\_name(“android.widget.LinearLayout”)

### find\_element\_by\_css\_selector

find\_element\_by\_css\_selector(self, css\_selector):

Finds element within this element's children by CSS selector.

通过CSS选择器定位元素

:Args:

- css\_selector - CSS selctor string, ex: 'a.nav#home'

### send\_keys

send\_keys(self, \*value):

Simulates typing into the element.

在元素中模拟输入（开启appium自带的输入法并配置了appium输入法后，可以输入中英文）

:Args:

- value - A string for typing, or setting form fields. For setting

file inputs, this could be a local file path.

Use this to send simple key events or to fill out form fields::

form\_textfield = driver.find\_element\_by\_name('username')

form\_textfield.send\_keys("admin")

This can also be used to set file inputs.

::

file\_input = driver.find\_element\_by\_name('profilePic')

file\_input.send\_keys("path/to/profilepic.gif")

# Generally it's better to wrap the file path in one of the methods

# in os.path to return the actual path to support cross OS testing.

# file\_input.send\_keys(os.path.abspath("path/to/profilepic.gif"))

driver.element.send\_keys(“中英”)

### is\_displayed

is\_displayed(self):

Whether the element is visible to a user.

此元素用户是否可见。简单地说就是隐藏元素和被控件挡住无法操作的元素（仅限 Selenium，appium是否实现了类似功能不是太确定）这一项都会返回 False

用法 driver.element.is\_displayed()

### location\_once\_scrolled\_into\_view

location\_once\_scrolled\_into\_view(self):

"""THIS PROPERTY MAY CHANGE WITHOUT WARNING. Use this to discover

where on the screen an element is so that we can click it. This method

should cause the element to be scrolled into view.

Returns the top lefthand corner location on the screen, or ``None`` if

the element is not visible.

暂不知道用法

"""

### size

size(self):

The size of the element.

获取元素的大小（高和宽）

new\_size["height"] = size["height"]

new\_size["width"] = size["width"]

用法 driver.element.size

### value\_of\_css\_property

value\_of\_css\_property(self, property\_name):

The value of a CSS property.

CSS属性

用法 暂不知

### location

location(self):

The location of the element in the renderable canvas.

获取元素左上角的坐标

用法 driver.element.location

'''返回element的x坐标, int类型'''

driver.element.location.get('x')

'''返回element的y坐标, int类型'''

driver.element.location.get('y')

### rect

rect(self):

A dictionary with the size and location of the element.

元素的大小和位置的字典

### screenshot\_as\_base64

screenshot\_as\_base64(self):

Gets the screenshot of the current element as a base64 encoded string.

获取当前元素的截图为Base64编码的字符串

:Usage:

img\_b64 = element.screenshot\_as\_base64

### execute\_script

execute\_script(self, script, \*args):

Synchronously Executes JavaScript in the current window/frame.

在当前窗口/框架（特指 Html 的 iframe ）同步执行 javascript 代码。你可以理解为如果这段代码是睡眠5秒，这五秒内主线程的 javascript 不会执行

:Args:

- script: The JavaScript to execute.

- \\*args: Any applicable arguments for your JavaScript.

:Usage:

driver.execute\_script('document.title')

### execute\_async\_script

execute\_async\_script(self, script, \*args):

Asynchronously Executes JavaScript in the current window/frame.

插入 javascript 代码，只是这个是异步的，也就是如果你的代码是睡眠5秒，那么你只是自己在睡，页面的其他 javascript 代码还是照常执行

:Args:

- script: The JavaScript to execute.

- \\*args: Any applicable arguments for your JavaScript.

:Usage:

driver.execute\_async\_script('document.title')

### current\_url

current\_url(self):

Gets the URL of the current page.

获取当前页面的网址。

:Usage:

driver.current\_url

用法 driver.current\_url

### page\_source

page\_source(self):

Gets the source of the current page.

获取当前页面的源。

:Usage:

driver.page\_source

### close

close(self):

Closes the current window.

关闭当前窗口

:Usage:

driver.close()

### quit

quit(self):

Quits the driver and closes every associated window.

退出脚本运行并关闭每个相关的窗口连接

:Usage:

driver.quit()

## Native和webview切换

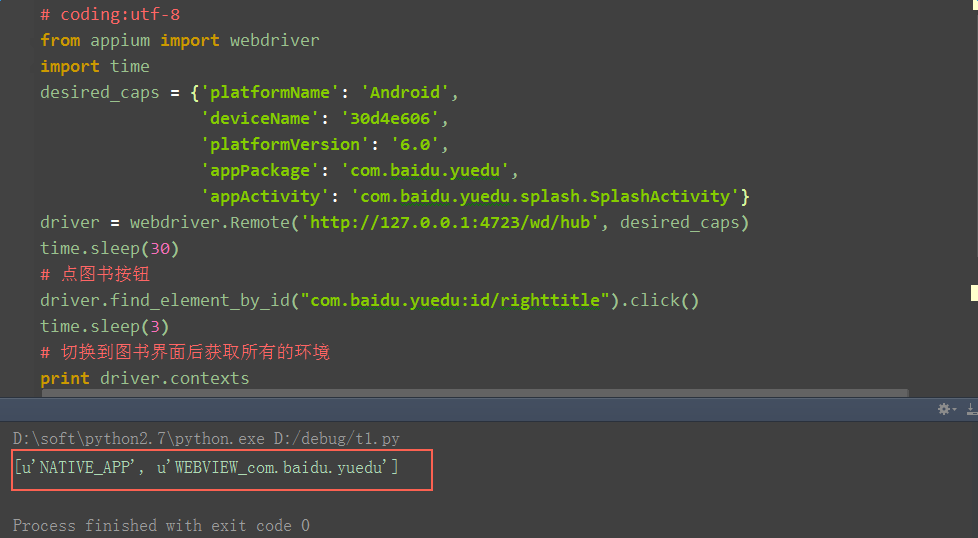
二、contexts

2.先获取页面是contexts环境，如下图红色区域，获取的是一个list列表：

NATIVE\_APP:这个就是native，也就是原生的

WEBVIEW\_com.xxxx :这个就是webview

3.当看到打印出来有下图两个，就说明获取到webview的context了（当然也有的app有坑，可能明明有webview，却通过contexts获取不到，这种需要特殊处理了）

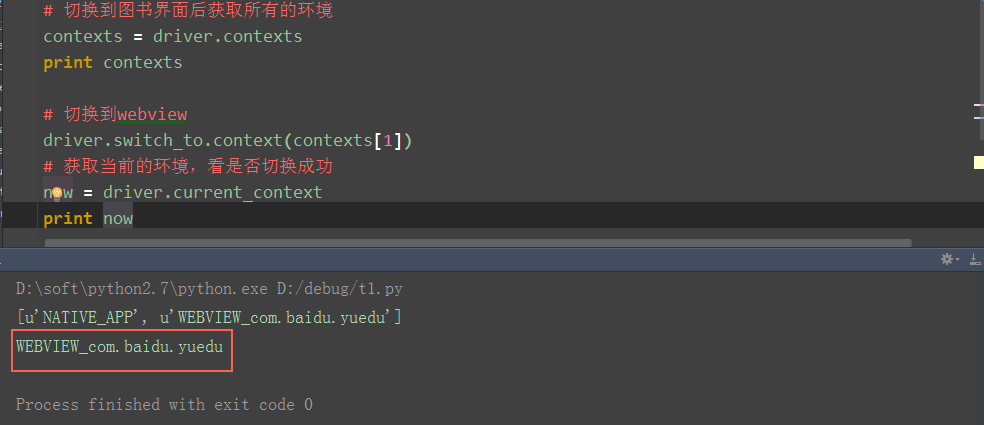


三、切换到webview

1.要想操作webview上的元素，第一步需要切换环境（跟selenium的切换iframe,切换handle思路是一样）

2.切换方法：switch\_to.context(参数是webview的context)

由于第二步已经获取到contexts是一个list对象，取这个list的第二个参数就行，也就是contexts[1]



四、切回native

1.webview上操作完后，想回到native上操作，这时候需要先切回来，切回native有两个方法：

方法一：driver.switch\_to.context("NATIVE\_APP")   # 这个NATIVE\_APP是固定的参数

方法二：driver.switch\_to.context(contexts[0])      # 从contexts里取第一个参数

参考代码

|  |
| --- |
| # coding:utf-8  from appium import webdriver  import time  desired\_caps = {'platformName': 'Android',                  'deviceName': '30d4e606',                  'platformVersion': '6.0',                  'appPackage': 'com.baidu.yuedu',                  'appActivity': 'com.baidu.yuedu.splash.SplashActivity'}  driver = webdriver.Remote('http://127.0.0.1:4723/wd/hub', desired\_caps)  time.sleep(30)  # 点图书按钮  driver.find\_element\_by\_id("com.baidu.yuedu:id/righttitle").click()  time.sleep(3)  # 切换到图书界面后获取所有的环境  contexts = driver.contexts  print contexts  # 切换到webview  driver.switch\_to.context(contexts[1])  # 获取当前的环境，看是否切换成功  now = driver.current\_context  print now  # 切回native  driver.switch\_to.context(contexts[0])  # driver.switch\_to.context("NATIVE\_APP") # 这样也是可以的  # 获取当前的环境，看是否切换成功  now = driver.current\_context  print now |

## 九宫格解锁（TouchAction）

### TouchAction

1.源码可以在这个路径找到：Lib\site-packages\appium\webdriver\common\touch\_action.py

```

class TouchAction(object):

    def \_\_init\_\_(self, driver=None):

        self.\_driver = driver

        self.\_actions = []

**def tap**(self, element=None, x=None, y=None, count=1):

        模拟手指触摸屏

**def press(**self, el=None, x=None, y=None):

        短按：模拟手指按住一个元素，或者坐标

**def long\_press**(self, el=None, x=None, y=None, duration=1000):

        长按：模拟按住一个元素，或者坐标

**def wait**(self, ms=0):

        按住元素后的等待时间

**def move\_to**(self, el=None, x=None, y=None):

        移动手指到另外一个元素，或者坐标，注意这里坐标不是绝对坐标，是偏移量

**def release(**self):

        释放手指

**def perform**(self):

        执行前面的动作

```

2.TouchAction里面有这几个动作：

- 触摸 (tap)

- 短按 (press)

- 长按 (long\_press)

- 等待 (wait)

- 移动到 (moveTo)

- 释放 (release)

- 执行 (perform)

### 九宫格解锁

# 定位九宫格元素

jiu = 'resourceId("com.tencent.mobileqq:id/name").index(6)'

loc = driver.find\_element\_by\_android\_uiautomator(jiu).location

print("获取九宫格坐标位置：%s"%loc)

s = driver.find\_element\_by\_android\_uiautomator(jiu).size

print("获取九宫格宽和高：%s"%s)

```

3.给每个圆圈编号从左到右1，2，3依次第二行4，5，6第三行7，8，9

```

gongge = {}

gongge[1] = (None, loc["x"]+s["width"]/6, loc["y"]+s["height"]/6)

gongge[2] = (None, loc["x"]+s["width"]/6\*3, loc["y"]+s["height"]/6)

gongge[3] = (None, loc["x"]+s["width"]/6\*5, loc["y"]+s["height"]/6)

gongge[4] = (None, loc["x"]+s["width"]/6, loc["y"]+s["height"]/6\*3)

gongge[5] = (None, loc["x"]+s["width"]/6\*3, loc["y"]+s["height"]/6\*3)

gongge[6] = (None, loc["x"]+s["width"]/6\*5, loc["y"]+s["height"]/6\*3)

gongge[7] = (None, loc["x"]+s["width"]/6, loc["y"]+s["height"]/6\*5)

gongge[8] = (None, loc["x"]+s["width"]/6\*3, loc["y"]+s["height"]/6\*5)

gongge[9] = (None, loc["x"]+s["width"]/6\*5, loc["y"]+s["height"]/6\*5)

print gongge

```

### 偏移量

1.这里有个坑，press里面的参数是元素的坐标位置，但是move\_to里面的是相对于前面一个元素的偏移位置。所以需要单独写一个函数，计算偏移量。

```

def pianyi(a=1,b=2):

    '''计算从a点到b点的偏移量'''

    g1 = gongge[a]

    g2 = gongge[b]

    r = (None, g2[1]-g1[1], g2[2]-g1[2])

    return r

```

2.另外press和move\_to里面都有三个参数，第一个参数默认为None，所以我返回的参数里面第一个写None.

### 移动手指

1.解锁思路：先press按住第一个点，再wait等待,接着移动带第二个点，再wait，最后release释放手指，perform执行

2.比如我要画出一个Z形状，依次经过的点1，2，3，5，7，8，9

代码

# coding:utf-8

from appium import webdriver

from appium.webdriver.common.touch\_action import TouchAction

from time import sleep

desired\_caps = {

                'platformName': 'Android',

                'deviceName': '127.0.0.1:62001',

                'platformVersion': '4.4.2',

                'appPackage': 'com.tencent.mobileqq',

                'appActivity': 'com.tencent.mobileqq.activity.SplashActivity',

                'noReset': "true"

                }

driver = webdriver.Remote('http://127.0.0.1:4723/wd/hub', desired\_caps)

sleep(5)

jiu = 'resourceId("com.tencent.mobileqq:id/name").index(6)'

loc = driver.find\_element\_by\_android\_uiautomator(jiu).location

print("获取九宫格坐标位置：%s"%loc)

s = driver.find\_element\_by\_android\_uiautomator(jiu).size

print("获取九宫格宽和高：%s"%s)

# 获取九个点的坐标

gongge = {}

gongge[1] = (None, loc["x"]+s["width"]/6, loc["y"]+s["height"]/6)

gongge[2] = (None, loc["x"]+s["width"]/6\*3, loc["y"]+s["height"]/6)

gongge[3] = (None, loc["x"]+s["width"]/6\*5, loc["y"]+s["height"]/6)

gongge[4] = (None, loc["x"]+s["width"]/6, loc["y"]+s["height"]/6\*3)

gongge[5] = (None, loc["x"]+s["width"]/6\*3, loc["y"]+s["height"]/6\*3)

gongge[6] = (None, loc["x"]+s["width"]/6\*5, loc["y"]+s["height"]/6\*3)

gongge[7] = (None, loc["x"]+s["width"]/6, loc["y"]+s["height"]/6\*5)

gongge[8] = (None, loc["x"]+s["width"]/6\*3, loc["y"]+s["height"]/6\*5)

gongge[9] = (None, loc["x"]+s["width"]/6\*5, loc["y"]+s["height"]/6\*5)

print gongge

def pianyi(a=1,b=2):

    '''计算从a点到b点的偏移量'''

    g1 = gongge[a]

    g2 = gongge[b]

    r = (None, g2[1]-g1[1], g2[2]-g1[2])

    return r

# 执行解锁

TouchAction(driver).press(\*gongge[1]).wait(300).move\_to(\*pianyi(1,2)).wait(300).move\_to(\*pianyi(2,3)).wait(300).move\_to(\*pianyi(3,5)).wait(300).move\_to(\*pianyi(5,7)).wait(300).move\_to(\*pianyi(7,8)).wait(300).move\_to(\*pianyi(8,9)).wait(

    300).release().perform()

# 操作

## appium元素定位

### Native APP

| **API** | **方法描述** |
| --- | --- |
| find\_element\_by\_id(self,id) | 通过控件的resource id来查找控件 |
| find\_element\_by\_name(self,name) | Native APP中，name就是控件的Text |
| find\_element\_by\_class\_name(self,name) | 控件的class name，网页测试也可以用此 |
| find\_element\_by\_accessibility\_id(self,id) | 控件的accessibility\_id就是Content Description |
| find\_element\_by\_android\_uiautomator(self,uia\_string) | 根据UIAutomator的语法查找控件，是WebDriver在兼容Appium时才新加的语法 |

### Web&Hybrid APP

| **API** | **方法描述** |
| --- | --- |
| find\_element\_by\_xpath(self,xpath) | 通过控件的xpath来查找控件 |
| find\_element\_by\_css\_selector(self,css\_selector) | 通过控件的css\_selector来查找控件 |
| find\_element\_by\_link\_text(self,link\_text) | 通过链接的text来查找控件 |
| find\_element\_by\_partiallink\_text(self,link\_text) | 通过链接的部分文本来查找控件 |
| find\_element\_by\_tag\_name(self,tag\_name) | 通过网页元素的Tag查找控件 |

## 获取控件信息（部分）

| **API** | **方法描述** |
| --- | --- |
| text(self) | 获取控件显示的文本信息 |
| is\_enabled(self) | 判断是否可用了，可用返回true |
| is\_selected(self) | 是否被选中了，是的话返回true |
| id\_displayed(self) | 判断控件是否显示，是的话返回true |
| get\_attribute(self,name) | 获取控件某项信息，如element.get\_attribute(“displayed”)等同于id\_displayed方法 |
| parent(self) | 返回控件的父控件，返回值为一个控件对象 |

## 手势操作（部分）

主要有点击、滑动、拖拽、放缩等常用的操作。

| **API** | **方法描述** |
| --- | --- |
| click(self) | 点击控件 |
| clear(self) | 清楚文本框控件的文本 |
| send\_keys(self,\*value) | 发送文本到控件中 |
| tap(self，positions，duration=None) | positions是一个列表，每个列表是一个二元组最多可以同时点击5个点；duration为时间长短，给参数的话则是长按操作 |
| swipe(self,start\_x,start\_y,end\_x,end\_y,duration=None) | 从一点滑动到另一点，时长为毫秒 |
| flick(self,start\_x,start\_y,end\_x,end\_y) | 两点快速的滑动 |
| scroll(self,origin\_ele,destination\_ele) | 从origin\_ele控件滚动到destination\_ele控件 |
| drag\_and\_drop(self,origin\_ele,destination\_ele) | 把origin\_ele控件拖拽到destination\_ele控件的位置 |
| pinch(self,element=None,percent=200,steps=50) | 在指定控件上执行缩小操作，默认缩放比例为2，分50步完成 |
| zoom(self,element=None,percent=200,steps=50) | 在指定控件上执行放大操作，默认缩放比例为2，分50步完成 |

## 系统操作API（部分）

系统操作用于模拟硬件操作、设置网络环境、获取系统信息等，下表简单的介绍一下常用的方法。

| **API** | **方法描述** |
| --- | --- |
| launch\_app(self) | 启动Capability中指定的APP |
| is\_app\_installed(self,package\_name) | 判断应用程序是否安装 |
| install\_app(self,app\_path) | 安装APP，app\_path指的是电脑上的apk路径 |
| close\_app(self) | 如果Capability指定的APP在运行，则关闭它 |
| background\_app(self,seconds) | 将APP放到后台运行一段时间 |
| reset(self) | 重置当前被测APP到初始状态 |
| current\_activity(self) | 获取当前正在显示的Activity |
| start\_activity(self,app\_package,app\_activity,\*\*opts) | 启动某个Activity |
| pull\_file(self,path) | 拉取手机上的一个文件，并以base64格式编码返回数据，path为手机文件路径 |
| pull\_folder(self,path) | 拉取手机上的一个文件夹，打包后以base64格式编码返回数据，path为手机上的文件夹路径 |
| push\_file(self,path，base64data) | 将一个base64编码格式的文件从电脑推送到手机上的路径path上 |
| press\_keycode(self,keycode,metastate=None) | 模拟发送一个硬件码到手机，如返回等 |
| open\_notification(self) | 打开通知栏 |
| network\_connection(self) | 返回当前网络连接的类型 |
| set\_network\_connection(self,connectionType) | 设置网络，值为：0 未设置，1 飞行模式，2 WiFi only， 4 Data only, 6 WiFi& Data |
| get\_screenshot\_as\_file(self,filename) | 截图并保存在电脑上，filename为路径及截图名称 |
| save\_screenshot(filename) | 截图并保存在电脑上，filename为路径及截图名称 |

## 控件信息验证

这里我们要说的是查找并操作控件后，怎么确定我们的操作起了作用。在实际的测试中也把它叫做检查点，检查点的划分和验证是UI自动化中的一个重点也是难点。常用的有以下方法：

1. 判断某个控件是否显示（操作之后新出现的控件）
2. 判断某个控件是否被选中
3. 判断某个开关控件是否处于check状态
4. 判断某个控件是否enabled
5. 截图之后和正确的进行比对   
   …

可以将以上判断的方式进行封装，便于我们在if语句和assert中使用。关于截图对比的方式，首先要有正确的操作截图，然后再进行对比得出结论看看是否一致，会涉及一些算法相关的知识。

# 遇到的问题

## 能启动app，但不能点击的问题

小米手机要登录账号，在开发者模式下面

设置-更多设置-开发者选项-

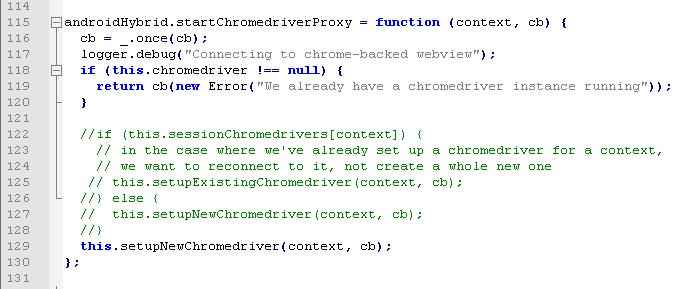
开启“USB调试（安全设置）”【允许通过USB调试修改权限或模拟点击】

## [解决Chromedriver报错Original error: chrome not reachable](http://www.cnblogs.com/ycyzharry/p/7003074.html)

解决办法：修改Appium安装目录下的文件，例如：

D:\Program Files\Appium\node\_modules\appium\lib\devices\android\android-hybrid.js

找到androidHybrid.startChromedriverProxy 函数，注释掉相关内容。



问题原因：

Appium第一次切换到Html页面时，会新生成一个Chromedriver；当第二次切换到Html时，会使用已经存在的Chromedriver。但其实在我们的应用里面每次打开一个Activity时一般都是会重新创建一个WebChromeClient，所以这里就把它改成无论如何都生成一个新的Chromedriver。

## [解决Android 6.0 NoSuchContextException 和WEBVIEW\_undefined 的问题](http://blog.csdn.net/xuecancan/article/details/69526086)

在 Android 4.4 操作系统上测试混合应用时，可以直接顺利的在native模式和webview模式之间切换，但是在Android6.0 操作系统上却报NoSuchContextException 的异常，

百度了一下原因：   
产生问题的原因：   
在Android 5.0中有9个字段在引导时，但在Android 6.0中有8字段在引导时：   
Android 5.0   
USER PID PPID VSIZE RSS WCHAN PC NAME   
u0\_a611 23802 188 1169748 133272 ffffffff 00000000 S com.XXX.XXX.XXX

Android 6.0   
User PID PPID VSIZE RSS WCHAN PC NAME   
u0\_a178 6346 211 1724936 123260 00000000 R com.XXX.XXX.XXX

但几秒钟后有9个字段   
USER PID PPID VSIZE RSS WCHAN PC NAME   
u0\_a182 10156 211 1721024 111224 sys\_epoll\_ 00000000 S com.XXX.XXX.XXX

sys\_epoll\_字段在几秒钟后才可见

所以，当我们尝试获取webview中9个字段但是只有8个时，我们就得到一个“undefined”   
解决方案：   
只针对Android 6.0+的解决办法：   
修改android-hybrid.js文件，在 pkg = line [pkgColumn]; 前面加上 pkgColumn = line.length - 1; 可以解决   
路径：C:、Program Files (x86)\Appium\node\_modules\appium\lib\devices\android\android-hybrid.js   
改完之后，重启了一下apium 服务，果然好了。

也有一些其他原因造成上述的错误，比如context的名字确实错了等，还要具体分析。