**1.4.ADB命令简介**

ADB

ADB全名Andorid Debug Bridge。 是一个Debug工具。为何称之为Bridge呢?

因为adb是一个标准的C/S结构的工具, 是要连接开发电脑和调试手机的

包含如下几个部分:

1.Client端，运行在开发机器中，即你的开发PC机。用来发送adb命令。

2.Daemon守护进程, 运行在调试设备中, 即的调试手机或模拟器。

3.Server端, 作为一个后台进程运行在开发机器中, 即你的开发PC机. 用来管理PC中的Client端和手机的Daemon之间的通信。

三者之间的通信

Client<—>Server<—>Daemon

adb帮助  adb --help

启动adb server  adb start-server

关闭adb server  adb kill-server

获取设备号 adb devices

获取系统版本

adb -s 设备号 shell getprop ro.build.version.release

发送文件到手机 adb push 电脑端文件路径/需要发送的文件 手机端存储的路径

示例：

将桌面的xx.png发送到手机sdcard目录下

adb push C:\Users\win\Desktop\xx.png /sdcard

从手机拉取文件 adb pull 手机端的路径/拉取文件名 电脑端存储文件路径

示例：

将手机/sdcard目录中的xx.png文件，发送到电脑桌面

adb pull /sdcard/xx.png C:\Users\win\Desktop

查看手机运行日志 adb logcat

手机shell命令行 adb shell

获取app启动包名和启动名(⚠️ 手机需要先打开对应app)

1.Mac/Linux: 'adb shell dumpsys window windows | grep mFocusedApp’

2.在 Windows 终端运行 'adb shell dumpsys window windows | findstr mFocusedApp’

安装app到手机 adb install 路径/xx.apk

卸载手机手机app adb uninstall 包名

获取app启动时间 adb shell am start -W 包名/启动名

示例：

adb shell am start -W com.yly.drawpic/.MainActivity

解释：

TotalTime：app自身启动时间

WaitTime：系统启动应用时间



手机驱动对象

驱动对象会帮助我们完成手机和脚本之间交互.

导入driver对象

from appium import webdriver

声明手机驱动对象

只有声明驱动对象我们才可以让手机完成脚本的操作,声明的方法如下:

driver = webdriver.Remote('http://127.0.0.1:4723/wd/hub', desired\_caps) # 声明对象后会直接启动参数中的应用

desired\_caps: ？？？

手机启动参数

desired\_caps：负责启动服务端时的参数设置，appium server 与手机端建立会话关系时，根据这些参数服务端可以做出相应的处理

desired\_caps常用参数:

platformName 平台的名称：iOS, Android, or FirefoxOS

platformVersion 设备系统版本号

deviceName 设备号 IOS：instruments -s devices，Android: adb devices

app 安装文件路径：/abs/path/to/my.apk or http://myapp.com/app

appActivity 启动的Activity

appPackage 启动的包

unicodeKeyboard unicode设置(允许中文输入)

resetKeyboard 键盘设置(允许中文输入)

# server 启动参数

desired\_caps = {}

desired\_caps['platformName'] = 'Android'

desired\_caps['platformVersion'] = '5.1'

desired\_caps['deviceName'] = '192.168.56.101:5555'

desired\_caps['appPackage'] = 'com.android.settings'

desired\_caps['appActivity'] = '.Settings'

desired\_caps['unicodeKeyboard'] = True

desired\_caps['resetKeyboard'] = True

脚本内启动其他app

driver.start\_activity(appPackage,appActivity)

关闭app

driver.close\_app() # 关闭当前操作的app，不会关闭驱动对象

关闭驱动对象

driver.quit() # 关闭驱动对象，同时关闭所有关联的app

实例代码

from appium import webdriver

import time

# server 启动参数

desired\_caps = {}

# 设备信息

desired\_caps['platformName'] = 'Android'

desired\_caps['platformVersion'] = '5.1'

desired\_caps['deviceName'] = '192.168.56.101:5555'

# app信息

desired\_caps['appPackage'] = 'com.android.settings'

desired\_caps['appActivity'] = '.Settings'

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

time.sleep(5)

driver.quit()

**3.1.App基础操作API**

完成app自动化需要一些基础条件的支持，本节将讲解APP初始化API.

3.1.1前置代码

# server 启动参数

desired\_caps = {}

desired\_caps['platformName'] = 'Android'

desired\_caps['platformVersion'] = '5.1'

desired\_caps['deviceName'] = '192.168.56.101:5555'

desired\_caps['appPackage'] = 'com.android.settings'

desired\_caps['appActivity'] = '.Settings'

desired\_caps['unicodeKeyboard'] = True

desired\_caps['resetKeyboard'] = True

# 声明driver对象

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

3.1.2安装APK到手机

driver.install\_app(app\_path)

参数：

app\_path：脚本机器中APK文件路径

3.1.3手机中移除APP

driver.remove\_app(app\_id)

参数：

app\_id：需要卸载的app包名

3.1.4判断APP是否已安装

driver.is\_app\_installed(bundle\_id)

参数：

bundle\_id: 可以传入app包名,返回结果为True(已安装) / False(未安装)

3.1.5发送文件到手机

import base64

data = str(base64.b64encode(data.encode('utf-8')),'utf-8')

driver.push\_file(path,data)

参数：

path：手机设备上的路径(例如：/sdcard/a.txt)

data：文件内数据,要求base64编码

Python3.x中字符都为unicode编码，而b64encode函数的参数为byte类型，需要先转码；

生成的数据为byte类型，需要将byte转换回去。

3.1.6从手机中拉取文件

import base64

data = driver.pull\_file(path) # 返回数据为base64编码

print(str(base64.b64decode(data),'utf-8')) # base64解码

参数：

path: 手机设备上的路径

3.1.7获取当前屏幕内元素结构

driver.page\_source

作用：返回当前页面的文档结构，判断特定的元素是否存在

**3.2.手机控件查看工具uiautomatorviewer：**用来扫描和分析Android应用程序的UI控件的工具.

如何使用

进入SDK目录下的tools目录，打开uiautomatorviewer

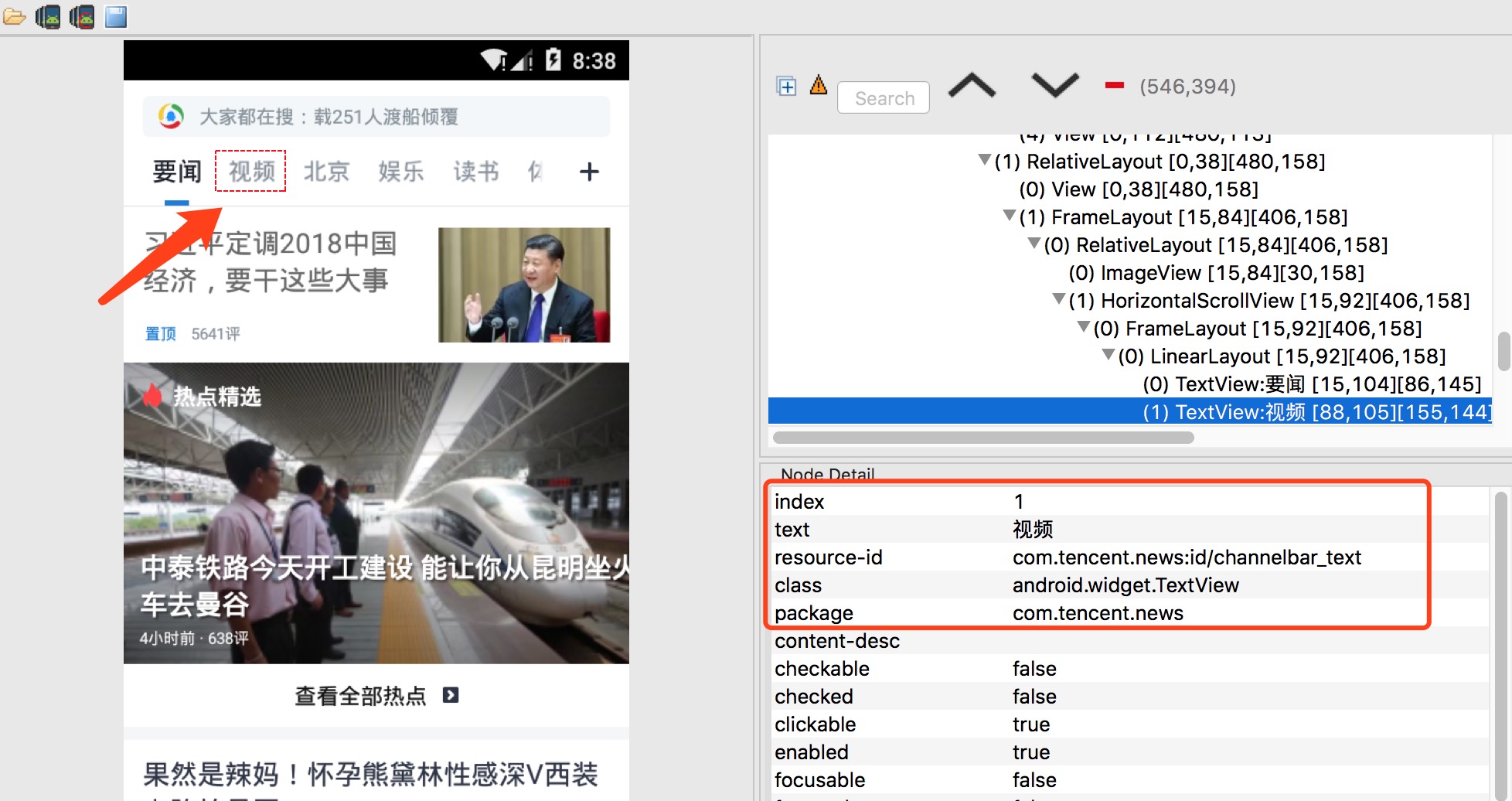
电脑连接真机或打开android模拟器

启动待测试app

点击uiautomatorviewer的左上角Device Screenshot,会生成app当前页面的UI控件截图



5.选择截图上需要查看的控件，即可浏览该控件的id,class,text,坐标等信息



**3.3.APP元素定位操作**

手工测试主要通过可见按钮操作，而自动化是通过元素进行交互操作.

⚠️⚠️⚠️ 元素的基本定位基于当前屏幕范围内展示的可见元素。

* 3.3.1Appium常用元素定位方式

| **name** | **value** |
| --- | --- |
| id | id属性值 |
| class | class属性值 |
| xpath | xpath表达式 |

* 3.3.2.前置代码

from appium import webdriver

# server 启动参数

desired\_caps = {}

# 设备信息

desired\_caps['platformName'] = 'Android'

desired\_caps['platformVersion'] = '5.1'

desired\_caps['deviceName'] = '192.168.56.101:5555'

# app的信息

desired\_caps['appPackage'] = 'com.android.settings'

desired\_caps['appActivity'] = '.Settings'

# 声明我们的driver对象

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

* 3.3.3.通过id定位

方法：find\_element\_by\_id(id\_value) # id\_value:为元素的id属性值

业务场景:

1.进入设置页面

2.通过ID定位方式点击搜索按钮

代码实现：

driver.find\_element\_by\_id("com.android.settings:id/search").click()

driver.quit()

* 3.3.4.通过class定位

方法：find\_element\_by\_class\_name(class\_value) # class\_value:为元素的class属性值

业务场景:

1.进入设置页面

2.点击搜索按钮

3.通过class定位方式点击输入框的返回按钮

代码实现：

# id 点击搜索按钮driver.find\_element\_by\_id("com.android.settings:id/search").click()

# class 点击输入框返回按钮driver.find\_element\_by\_class\_name('android.widget.ImageButton').click()

driver.quit()

* 3.3.5.通过xpath定位
* 方法：find\_element\_by\_xpath(xpath\_value) # xpath\_value:为可以定位到元素的xpath语句
* android端xptah常用属性定位：
* 1. id ://\*[contains(@resource-id,'com.android.settings:id/search')]
* 2. class ://\*[contains(@class,'android.widget.ImageButton')]
* 3. text ://\*[contains(@text,'WLA')]
* 模糊定位 contains(@key,value): value可以是部分值

业务场景:

1.进入设置页面

2.点击WLAN菜单栏

代码实现：

# xpath 点击WLAN按钮

driver.find\_element\_by\_xpath("//\*[contains(@text,'WLA')]").click()

* 3.3.6.定位一组元素,注意element -> elements

应用场景为元素值重复，无法通过元素属性直接定位到某个元素，只能通过elements方式来选择，返回一个定位对象的列表.

* 3.3.7.通过id方式定位一组元素

方法： find\_elements\_by\_id(id\_value) # id\_value:为元素的id属性值

业务场景:

1.进入设置页面

2.点击WLAN菜单栏(id定位对象列表中第1个)

代码实现：

# 定位到一组元素

title = driver.find\_elements\_by\_id("com.android.settings:id/title")

# 打印title类型，预期为list

print(type(title))

# 取title返回列表中的第一个定位对象，执行点击操作

title[0].click()

* 3.3.8.通过class方式定位一组元素

方法：find\_elements\_by\_class\_name(class\_value) # class\_value:为元素的class属性值

业务场景:

1.进入设置页面

2.点击WLAN菜单栏(class定位对象列表中第3个)

代码实现：

# 定位到一组元素

title = driver.find\_elements\_by\_class\_name("android.widget.TextView")

# 打印title类型，预期为list

print(type(title))

# 取title返回列表中的第一个定位对象，执行点击操作

title[3].click()

* 3.3.9.通过xpath方式定位一组元素

方法:find\_elements\_by\_xpath(xpath\_value) # xpath\_value:为可以定位到元素的xpath语句

业务场景:

1.进入设置页面

2.点击WLAN菜单栏(xpath中class属性定位对象列表中第3个)

代码实现：

# 定位到一组元素

title = driver.find\_elements\_by\_xpath("//\*[contains(@class,'widget.TextView')]")

# 打印title类型，预期为list

print(type(title))

# 取title返回列表中的第一个定位对象，执行点击操作

title[3].click()

* WebDriverWait 显示等待操作

在一个超时时间范围内，每隔一段时间去搜索一次元素是否存在，

如果存在返回定位对象，如果不存在直到超时时间到达，报超时异常错误。

方法:WebDriverWait(driver, timeout, poll\_frequency).until(method)

参数：

1.driver：手机驱动对象

2.timeout：搜索超时时间

3.poll\_frequency：每次搜索间隔时间，默认时间为0.5s

4.method：定位方法(匿名函数)

匿名函数:

lambda x: x

等价于python函数：

def test(x):

return x

使用示例：

WebDriverWait(driver, timeout, poll\_frequency).until(lambda x: x.find\_elements\_by\_id(id\_value))

# 解释：

1.x传入值为：driver，所以才可以使用定位方法.

函数运行过程：

1.实例化WebDriverWait类，传入driver对象，之后driver对象被赋值给WebDriverWait的一个类变量：self.\_driver

2.until为WebDriverWait类的方法，until传入method方法(即匿名函数)，之后method方法会被传入self.\_driver

3.搜索到元素后until返回定位对象，没有搜索到函数until返回超时异常错误.

业务场景:

1.进入设置页面

2.通过ID定位方式点击搜索按钮

# 代码实现：

from selenium.webdriver.support.wait import WebDriverWait

# 导入WebDriverWait类

# 超时时间为30s，每隔1秒搜索一次元素是否存在，如果元素存在返回定位对象并退出

search\_button = WebDriverWait(driver, 30, 1).until(lambda x: x.find\_elements\_by\_id(com.android.settings:id/search))

search\_button.click()

driver.quit()

**3.4.APP元素信息操作API**

* 前置代码

from appium import webdriver

# server 启动参数

desired\_caps = {}

# 设备信息

desired\_caps['platformName'] = 'Android'

desired\_caps['platformVersion'] = '5.1'

desired\_caps['deviceName'] = '192.168.56.101:5555'

# app的信息

desired\_caps['appPackage'] = 'com.android.settings'

desired\_caps['appActivity'] = '.Settings'

# 声明我们的driver对象

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

* 3.4.1.点击元素
* 3.4.2.发送数据到输入框
* 方法：send\_keys(vaue) # value：需要发送到输入框内的文本
* 业务场景:
* 1.打开设置
* 2.点击搜索按钮
* 3.输入内容abc

代码实现：

# 点击搜索按钮

driver.find\_element\_by\_id("com.android.settings:id/search").click()

# 定位到输入框并输入abc

driver.find\_element\_by\_id("android:id/search\_src\_text").send\_keys("abc")

重点:

大家可以将输入的abc 改成 输入中文，得到的结果:输入框无任何值输入且程序不会抱错

解决输入中文问题：

1.server 启动参数增加两个参数配置

desired\_caps['unicodeKeyboard'] = True

desired\_caps['resetKeyboard'] = True

2.再次运行会发现运行成功

# 点击搜索按钮

driver.find\_element\_by\_id("com.android.settings:id/search").click()

# 定位到输入框并输入abc

driver.find\_element\_by\_id("android:id/search\_src\_text").send\_keys("传智播客")

* 3.4.3.清空输入框内容
* 方法：clear()
* 业务场景:
* 1.打开设置
* 2.点击搜索按钮
* 3.输入内容abc
* 4.删除已输入abc
* 代码实现：
* # 点击搜索按钮
* driver.find\_element\_by\_id("com.android.settings:id/search").click()
* # 定位到输入框并输入abc
* input\_text = driver.find\_element\_by\_id("android:id/search\_src\_text")
* # 输入abc
* input\_text.send\_keys("abc")
* time.sleep(1)
* # 删除abc
* input\_text.clear()
* 3.4.4.获取元素的文本内容
* 方法: text
* 业务场景：
* 1.进入设置
* 2.获取所有元素class属性为“android.widget.TextView”的文本内容
* 代码实现：
* text\_vlaue = driver.find\_elements\_by\_class\_name("android.widget.TextView")
* for i in text\_vlaue:
* print(i.text)
* 执行结果：
* 设置
* 无线和网络
* WLAN
* 更多
* 设备
* 显示
* 提示音和通知
* 存储
* 3.4.5.获取元素的属性值
* 方法: get\_attribute(value) # value:元素的属性
* ⚠️ value='name' 返回content-desc / text属性值
* ⚠️ value='text' 返回text的属性值
* ⚠️ value='className' 返回 class属性值，只有 API=>18 才能支持
* ⚠️ value='resourceId' 返回 resource-id属性值，只有 API=>18 才能支持
* 业务场景：
* 1.进入设置
* 2.获取搜索按钮的content-desc属性值

**3.5.APP元素事件操作API**

* 3.5.1.前置代码

from appium import webdriver

# server 启动参数

desired\_caps = {}

# 设备信息

desired\_caps['platformName'] = 'Android'

desired\_caps['platformVersion'] = '5.1'

desired\_caps['deviceName'] = '192.168.56.101:5555'

# app的信息

desired\_caps['appPackage'] = 'com.android.settings'

desired\_caps['appActivity'] = '.Settings'

# 声明我们的driver对象

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

* 3.5.2.swip滑动事件
* ⚠️从一个坐标位置滑动到另一个坐标位置,只能是两个点之间的滑动
* 方法：swipe(start\_x, start\_y, end\_x, end\_y, duration=None)
* 参数：
* 1.start\_x：起点X轴坐标
* 2.start\_y：起点Y轴坐标
* 3.end\_x： 终点X轴坐标
* 4.end\_y,： 终点Y轴坐标
* 5.duration： 滑动这个操作一共持续的时间长度，单位：ms
* 业务场景：
* 1.进入设置
* 2.从坐标(148,659)滑动到坐标(148,248)
* 代码实现：
* # 滑动没有持续时间
* driver.swipe(188,659,148,248)
* # 滑动持续5秒的时间
* driver.swipe(188,659,148,248,5000)
* 3.5.3.scroll滑动事件
* ⚠️ 从一个元素滑动到另一个元素，直到页面自动停止
* 方法：scroll(origin\_el, destination\_el)
* 参数：
* 1.origin\_el：滑动开始的元素
* 2.destination\_el：滑动结束的元素
* 业务场景：
* 1.进入设置页
* 2.模拟手指从存储菜单位置 到 WLAN菜单位置的上滑操作
* 代码实现：
* # 定位到存储菜单栏
* el1 = driver.find\_element\_by\_xpath("//\*[contains(@text,'存储')]")
* # 定位到WLAN菜单栏
* el2 = driver.find\_element\_by\_xpath("//\*[contains(@text,'WLAN')]")
* # 执行滑动操作
* driver.scroll(el1,el2)
* 3.5.4.drag拖拽事件
* ⚠️ 从一个元素滑动到另一个元素,第二个元素替代第一个元素原本屏幕上的位置
* 方法：drag\_and\_drop(origin\_el, destination\_el)
* 参数：
* 1.origin\_el：滑动开始的元素
* 2.destination\_el：滑动结束的元素
* 业务场景：
* 1.进入设置页
* 2.模拟手指将存储菜单 滑动到 WLAN菜单栏位置
* 代码实现：
* # 定位到存储菜单栏
* el1 = driver.find\_element\_by\_xpath("//\*[contains(@text,'存储')]")
* # 定位到WLAN菜单栏
* el2 = driver.find\_element\_by\_xpath("//\*[contains(@text,'WLAN')]")
* # 执行滑动操作
* driver.drag\_and\_drop(el1,el2)
* 3.5.5.应用置于后台事件
* APP放置后台，模拟热启动
* 方法：background\_app(seconds)
* 参数：
* 1.seconds:停留在后台的时间，单位：秒
* 业务场景：
* 1.进入设置页
* 2.将APP置于后台5s
* 代码实现：
* driver.background\_app(5)
* 效果：
* app置于后台5s后，再次展示当前页面

**3.6.APP模拟手势高级操作**

TouchAction是AppiumDriver的辅助类，主要针对手势操作，比如滑动、长按、拖动等，

原理是将一系列的动作放在一个链条中发送到服务器，服务器接受到该链条后，解析各个动作，逐个执行。

* 3.6.1.前置代码

from appium import webdriver

# server 启动参数

desired\_caps = {}

# 设备信息

desired\_caps['platformName'] = 'Android'

desired\_caps['platformVersion'] = '5.1'

desired\_caps['deviceName'] = '192.168.56.101:5555'

# app的信息

desired\_caps['appPackage'] = 'com.android.settings'

desired\_caps['appActivity'] = '.Settings'

# 声明我们的driver对象

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

* ⚠️ 所有手势都要通过执行函数才会运行.
* 3.6.2.手指轻敲操作
* 模拟手指轻敲一下屏幕操作
* 方法：tap(element=None, x=None, y=None)
* 方法：perform() # 发送命令到服务器执行操作
* 参数：
* 1.element：被定位到的元素
* 2.x：相对于元素左上角的坐标，通常会使用元素的X轴坐标
* 3.y：通常会使用元素的Y轴坐标
* 业务场景：
* 1.进入设置
* 2.点击WLAN选项
* 代码实现：
* # 通过元素定位方式敲击屏幕
* el = driver.find\_element\_by\_xpath("//\*[contains(@text,'WLAN')]")
* TouchAction(driver).tap(el).perform()
* # 通过坐标方式敲击屏幕，WLAN坐标:x=155,y=250
* # TouchAction(driver).tap(x=155,y=250).perform()
* 3.6.3.手指按操作
* 模拟手指按下屏幕,按就要对应着离开.
* 方法:press(el=None, x=None, y=None)
* 方法：release() # 结束动作，手指离开屏幕
* 参数：
* 1.element：被定位到的元素
* 2.x：通常会使用元素的X轴坐标
* 3.y：通常会使用元素的Y轴坐标
* 业务场景:
* 1.进入设置
* 2.点击WLAN选项
* 代码实现：
* # 通过元素定位方式按下屏幕
* el = driver.find\_element\_by\_xpath("//\*[contains(@text,'WLAN')]")
* TouchAction(driver).press(el).release().perform()
* # 通过坐标方式按下屏幕，WLAN坐标:x=155,y=250
* # TouchAction(driver).tap(x=155,y=250).release().perform()
* 3.6.4.等待操作
* 方法：wait(ms=0)
* 参数：
* ms：暂停的毫秒数
* 业务场景:
* 1.进入设置
* 2.点击WLAN选项
* 3.长按WiredSSID选项5秒
* 代码实现：
* # 点击WLAN
* driver.find\_element\_by\_xpath("//\*[contains(@text,'WLAN')]").click()
* # 定位到WiredSSID
* el =driver.find\_element\_by\_id("android:id/title")
* # 通过元素定位方式长按元素
* TouchAction(driver).press(el).wait(5000).perform()
* # 通过坐标方式模拟长按元素
* # 添加等待(有长按)／不添加等待(无长按效果)
* # TouchAction(driver).press(x=770,y=667).wait(5000).release().perform()
* 3.6.5.手指长按操作
* 模拟手机按下屏幕一段时间,按就要对应着离开.
* 方法：long\_press(el=None, x=None, y=None, duration=1000)
* 参数：
* 1.element：被定位到的元素
* 2.x：通常会使用元素的X轴坐标
* 3.y：通常会使用元素的Y轴坐标
* 4.duration：持续时间，默认为1000ms
* 业务场景:
* 1.进入设置
* 2.点击WLAN选项
* 3.长按WiredSSID选项5秒
* 代码实现：
* # 点击WLAN
* driver.find\_element\_by\_xpath("//\*[contains(@text,'WLAN')]").click()
* # 定位到WiredSSID
* el =driver.find\_element\_by\_id("android:id/title")
* # 通过元素定位方式长按元素
* TouchAction(driver).long\_press(el,duration=5000).release().perform()
* # 通过坐标方式长按元素，WiredSSID坐标:x=770,y=667
* # 添加等待(有长按)／不添加等待(无长按效果)
* # TouchAction(driver).long\_press(x=770,y=667).perform()

3.6.6.手指移动操作

* 模拟手机的滑动操作
* 方法：move\_to(el=None, x=None, y=None)
* 参数:
* 1.el:定位的元素
* 2.x:相对于前一个元素的X轴偏移量
* 3.y:相对于前一个元素的Y轴偏移量
* 业务场景1：
* 1.进入设置
* 2.向上滑动屏幕

代码实现：

# 定位到存储

el = driver.find\_element\_by\_xpath("//\*[contains(@text,'存储')]")

# 定位到更多

el1 = driver.find\_element\_by\_xpath("//\*[contains(@text,'更多')]")

# 元素方式滑动

TouchAction(driver).press(el).move\_to(el1).release().perform()

# 坐标的方式向上滑动

# TouchAction(driver).press(x=240,y=1000).move\_to(x=0,y=-400).release().perform()

# ⚠️ press().move\_to() 实际使用的是TouchAction方法，需要给相对坐标.

# TouchAction(driver).press(x=240,y=600).wait(100).move\_to(x=240,y=100).release().perform()

# ⚠️ press().wait().move\_to()实际调用的是swip方法，会向下拉，感觉属于bug，可在log中查询swip，不建议这么用.

业务场景2：

1.进入设置

2.向上滑动屏幕到可见"安全"选项

3.进入到安全

4.点击屏幕锁定方式

5.点击图案

6.绘制图案

代码实现：

# 定位到WLAN

el1 = driver.find\_element\_by\_xpath("//\*[contains(@text,'WLAN')]")

# 定位到存储

el2 = driver.find\_element\_by\_xpath("//\*[contains(@text,'存储')]")

# 存储上滑到WLAN

driver.drag\_and\_drop(el2,el1)

# 定位到用户

el3 = driver.find\_element\_by\_xpath("//\*[contains(@text,'用户')]")

# 注意 这次使用drag\_and\_drop方法，传入的"存储定位"仍使用其原始在屏幕上的位置，所以是由存储滑动到用户才可以上滑，否则需要重新"定位存储"

# 存储上滑倒用户位置

driver.drag\_and\_drop(el2,el3)

# 点击安全按钮

driver.find\_element\_by\_xpath("//\*[contains(@text,'安全')]").click()

# 点击屏幕锁定方式按钮

driver.find\_element\_by\_xpath("//\*[contains(@text,'屏幕锁定')]").click()

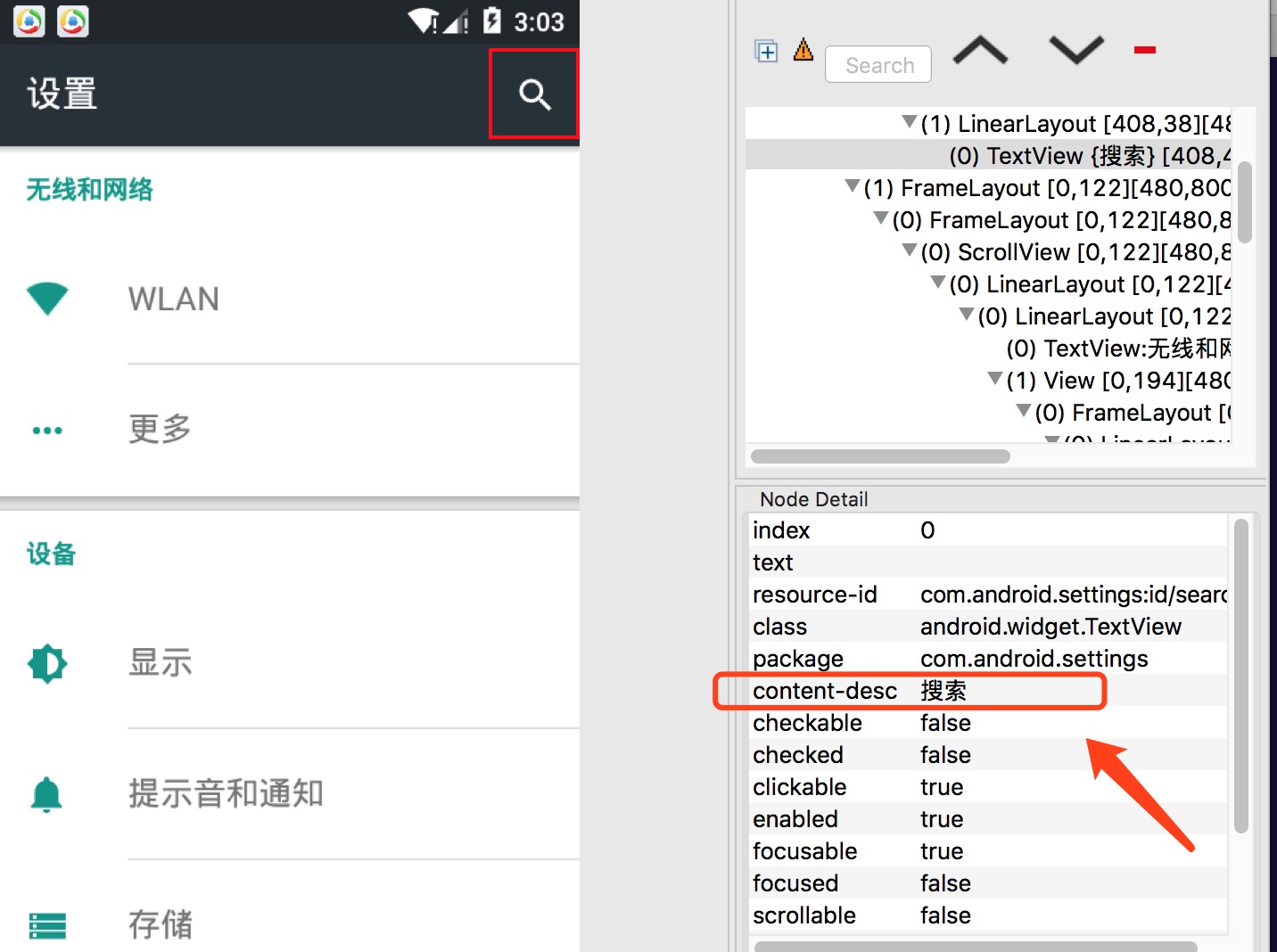
# 点击图案按钮

driver.find\_element\_by\_xpath("//\*[contains(@text,'图案')]").click()

# 绘制图案四个坐标 1:(244,967) 2:(723,967) 3:(723,1442) 4:(244,1916)

TouchAction(driver).press(x=244,y=967).wait(100).move\_to(x=479,y=0).wait(100)\

.move\_to(x=0,y=475).wait(100).move\_to(x=-479,y=474).release().perform()



**3.7.手机操作API**

针对手机的一些常用设置功能进行操作.

* 3.7.1.前置代码

from appium import webdriver

# server 启动参数

desired\_caps = {}

# 设备信息

desired\_caps['platformName'] = 'Android'

desired\_caps['platformVersion'] = '5.1'

desired\_caps['deviceName'] = '192.168.56.101:5555'

# app的信息

desired\_caps['appPackage'] = 'com.android.settings'

desired\_caps['appActivity'] = '.Settings'

# 声明我们的driver对象

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

* 3.7.2.获取手机时间
* 方法：device\_time
* 代码实现：
* # 获取当前手机的时间
* print(driver.device\_time)
* 执行结果：
* Wed Dec 27 08:52:45 EST 2017
* 3.7.3.获取手机的宽高
* 获取手机的宽高，可以根据宽高做一些坐标的操作
* 方法：get\_window\_size()
* 代码实现：
* print(driver.get\_window\_size())
* 执行结果：
* {'height': 800, 'width': 480}
* 3.7.4.发送键到设备
* 模拟系统键值的操作，比如操作honme键，音量键,返回键等。
* 方法：keyevent(keycode, metastate=None):
* 参数：
* keycode：发送给设备的关键代码
* metastate：关于被发送的关键代码的元信息，一般为默认值
* 业务场景:
* 1.打开设置
* 2.按多次音量增加键
* 代码实现：
* for i in range(3):
* driver.keyevent(24)