## 启动多个Appium服务

前面课程只是启动了单个appium服务，只能控制单台设备。如果需要针对多台设备测试那么该如何处理？

首先看下面两个启动appium服务案例。

启动appium服务1

C:\Users\Shuqing>appium -p 4723

[Appium] Welcome **to** Appium v1.7.2

[Appium] Appium REST http interface listener started on 0.0.0.0:4723

启动appium 服务2

C:\Users\Shuqing>appium -p 4725

[Appium] Welcome **to** Appium v1.7.2

[Appium] Non-default server args:

[Appium] port: 4725

[Appium] Appium REST http interface listener started on 0.0.0.0:4725

上面案例我们启动了2个不同的appium服务器，他们通过不同的端口来区分不同的服务；如同百米赛跑要给不同的运动员安排不同的赛道，每个运动员也只能在自己指定的赛道进行比赛。

#### Appium常用参数

| **参数** | **默认值** | **含义** |
| --- | --- | --- |
| -U, --udid | null | 连接物理设备的唯一设备标识符 |
| -a, --address | 0.0.0.0 | 监听的 ip 地址 |
| -p, --port | 4723 | 监听的端口 |
| -bp, --bootstrap-port | 4724 | 连接Android设备的端口号(Android-only) |
| -g, --log | null | 将日志输出到指定文件 |
| --no-reset | false | session 之间不重置应用状态 |
| --session-override | false | 允许 session 被覆盖 (冲突的话) |
| --app-activity | null | 打开Android应用时，启动的 Activity(Android-only) 的名字 |
| --app | null | 本地绝对路径\_或\_远程 http URL 所指向的一个安装包 |

更多参数请输入命令： appium -h

#### Appium安卓手机每次运行时都要安装 Unlock、Setting解决方案

首先通过如下命令找到appium的安装路径

C:\Users\Shuqing>where appium

C:\Users\Shuqing\AppData\Roaming\npm\appium

C:\Users\Shuqing\AppData\Roaming\npm\appium.cmd

1.打开 C:\Users\Shuqing\AppData\Roaming\npm\node\_modules\appium\node\_modules\appium-android-driver\lib 中的**android-helpers.js**

**#注释475行如下代码**

*//await helpers.pushSettingsApp(adb);*

**#注释486行如下代码**

*//await helpers.pushUnlock(adb);*

2.打开C:\Users\Shuqing\AppData\Roaming\npm\node\_modules\appium\node\_modules\appium-android-driver\build\lib中的**android-helpers.js**

**#注释1128行下面这行代码**

//**return** \_regeneratorRuntime.awrap(helpers.pushSettingsApp(adb))*;*

**#修改为如下：**

**return** context$1$0.abrupt('return', defaultIME)*;*

**#注释1163行下面这行代码**

//**return** \_regeneratorRuntime.awrap(helpers.pushUnlock(adb))*;*

**#修改如下：**

**return** context$1$0.abrupt('return', defaultIME)*;*

修改完成后重启Appium服务即可，如果新设备没有这个两个守护App可以手动安装这两个App到设备。

**Appium Setting路径** :

{appium安装路径}\node\_modules\\_io.appium.settings@2.4.0@io.appium.settings\app\build\outputs\apk\settings\_apk-debug.apk

eg:

C:\Users\Shuqing\AppData\Roaming\npm\node\_modules\appium\node\_modules\\_io.appium.settings@2.4.0@io.appium.settings\app\build\outputs\apk\settings\_apk-debug.apk

**Unlock app路径**:

appium安装路径\node\_modules\\_appium-unlock@2.0.0@appium-unlock\bin \unlock\_apk-debug.apk

eg：

C:\Users\Shuqing\AppData\Roaming\npm\node\_modules\appium\node\_modules\\_io.appium.settings@2.4.0@io.appium.settings\app\build\outputs\apk\unlock\_apk-debug.apk

## 多设备启动

前面我们已经启动了多个appium服务，那么接下来我们可以基于这些服务来启动不同的设备。

##### 测试场景

连接以下2台设备，然后分别启动考研帮App

* 设备1：127.0.0.1:62001
* 设备2：127.0.0.1:62025

##### 代码实现

multi\_device.py

from appium import webdriver

import yaml

from time import ctime

with open('desired\_caps.yaml','r')as file:

data=yaml.load(file)

devices\_list=['127.0.0.1:62001','127.0.0.1:62025']

def appium\_desire(udid,port):

desired\_caps={}

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

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

desired\_caps['deviceName']=data['deviceName']

desired\_caps['udid']=udid

desired\_caps['app']=data['app']

desired\_caps['appPackage']=data['appPackage']

desired\_caps['appActivity']=data['appActivity']

desired\_caps['noReset']=data['noReset']

print('appium port: %s start run %s at %s' *%(port,udid,ctime()))*

driver=webdriver.Remote('http://'+str(data['ip'])+':'+str(port)+'/wd/hub',desired\_caps)

return driver

if \_\_name\_\_ == '\_\_main\_\_':

appium\_desire(devices\_list[0],4723)

appium\_desire(devices\_list[1],4725)

## 多进程并发启动设备

上面的案例设备启动并不是并发进行的，而是先后执行。如何实现2台设备同时启动，并启动App呢？

#### 测试场景

同时启动2台设备：'127.0.0.1:62025'和'127.0.0.1:62001'并打开考研帮app

#### 实现思路

可以使用Python多线程或者多进程实现。这里我们推荐使用**多进程( multiprocessing)** 原因如下：

* 多进程中，同一个变量，各自有一份拷贝存在于每个进程中，互不影响。
* 而多线程中，所有变量都由所有线程共享，所以，任何一个变量都可以被任何一个线程修改，因此，线程之间共享数据最大的危险在于多个线程同时改一个变量，容易把内容给改乱了。

知识点补充：

* [线程与进程](http://www.51zxw.net/show.aspx?id=60446&cid=615)
* [Python多进程](http://www.51zxw.net/show.aspx?id=60449&cid=615)
* [python多线程](http://www.51zxw.net/show.aspx?id=60448&cid=615)

#### 代码实现

multi\_devices\_sync.py

from appium **import** webdriver

**import** yaml

from time **import** ctime

**import** multiprocessing

**with** open('desired\_caps.yaml','r') **as** file:

**data**=yaml.load(file)

devices\_list=['127.0.0.1:62001','127.0.0.1:62025']

def appium\_desired(udid,port):

desired\_caps={}

desired\_caps['platformName']=**data**['platformName']

desired\_caps['platformVersion']=**data**['platformVersion']

desired\_caps['deviceName']=**data**['deviceName']

desired\_caps['udid']=udid

desired\_caps['app']=**data**['app']

desired\_caps['appPackage']=**data**['appPackage']

desired\_caps['appActivity']=**data**['appActivity']

desired\_caps['noReset']=**data**['noReset']

print('appium port:%s start run %s at %s' %(port,udid,ctime()))

driver=webdriver.Remote('http://'+str(**data**['ip'])+':'+str(port)+'/wd/hub',desired\_caps)

driver.implicitly\_wait(5)

return driver

#构建desired进程租

desired\_process=[]

#加载desied进程

**for** i in range(len(devices\_list)):

port = 4723 + 2 \* i

desired=multiprocessing.Process(target=appium\_desired,args=(devices\_list[i],port))

desired\_process.append(desired)

**if** \_\_name\_\_ == '\_\_main\_\_':

# 启动多设备执行测试

**for** desired in desired\_process:

desired.start()

**for** desired in desired\_process:

desired.join()

## Python启动Appium 服务

目前我们已经实现了并发启动设备，但是我们的Appium服务启动还是手动档，比如使用Dos命令或者bat批处理来手动启动appium服务，启动效率低下。如何将启动Appium服务也实现自动化呢？

#### 方案分析

我们可以使用python启动appium服务，这里需要使用**subprocess**模块，该模块可以创建新的进程，并且连接到进程的输入、输出、错误等管道信息，并且可以获取进程的返回值。

[subprocess模块官方文档](https://docs.python.org/3.5/library/subprocess.html)

#### 测试场景

使用Python启动2台appium服务，端口配置如下：

* Appium服务器端口：4723，bp端口为4724
* Appium服务器端口：4725，bp端口为4726

说明：bp端口（ --bootstrap-port）是appium和设备之间通信的端口，如果不指定到时无法操作多台设备运行脚本。

#### 代码实现

首先我们使用Python脚本启动单个appium服务：

* host：127.0.0.1
* port：4723

multi\_appium.py

**import** subprocess

**from** time **import** ctime

**def** **appium\_start**(host,port):

'''启动appium server'''

bootstrap\_port = str(port + 1)

cmd = 'start /b appium -a ' + host + ' -p ' + str(port) + ' -bp ' + str(bootstrap\_port)

print('%s at %s' %(cmd,ctime()))

subprocess.Popen(cmd, shell=**True**,stdout=open('./appium\_log/'+str(port)+'.log','a'),stderr=subprocess.STDOUT)

**if** \_\_name\_\_ == '\_\_main\_\_':

host = '127.0.0.1'

port=4723

appium\_start(host,port)

#### 启动校验

启动后我们需要校验服务是否启动成功，校验方法如下：

1. 首先查看有没有生成对应的log文件，查看log里面的内容。
2. 使用如下命令来查看

netstat -ano |findstr 端口号

##### netstat 命令解释

netstat命令是一个监控TCP/IP网络的非常有用的工具，它可以显示路由表、实际的网络连接以及每一个网络接口设备的状态信息。输入 netstat -ano 回车.可以查看本机开放的全部端口；输入命令 netstat -h可以查看全部参数含义。

C:\Users\Shuqing>netstat -ano |findstr "4723"

TCP 127.0**.0.1**:4723 0.0**.0.0**:0 LISTENING 8224

#### 关闭Appium服务

关闭进程有2种方式，具体如下：

1. 通过netstat命令找到对应的Appium进程pid然后可以在系统任务管理器去关闭进程；

[win7系统任务管理器PID显示](https://jingyan.baidu.com/article/95c9d20d47b63bec4f756140.html)

1. 使用如下命令来关闭：

taskkill -f -pid appium进程id

#### 多个appium服务启动

多个appium服务启动非常简单，只需在执行环境使用循环调用即可。

**if** \_\_name\_\_ == '\_\_main\_\_':

host = '127.0.0.1'

**for** **i** **in** range(2):

port=4723+2\***i**

appium\_start(host,port)

## 多进程并发启动appium服务

上面的案例还不是并发执行启动appium,因此我们需要使用多进程来实现并发启动。 同样需要引入multiprocessing多进程模块。

muti\_appium\_sync.py

**import** multiprocessing

**import** subprocess

**from** time **import** ctime

**def** **appium\_start**(host,port):

'''启动appium server'''

bootstrap\_port = str(port + 1)

cmd = 'start /b appium -a ' + host + ' -p ' + str(port) + ' --bootstrap-port ' + str(bootstrap\_port)

print('%s at %s' %(cmd,ctime()))

subprocess.Popen(cmd, shell=**True**,stdout=open('./appium\_log/'+str(port)+'.log','a'),stderr=subprocess.STDOUT)

*#构建appium进程组*

appium\_process=[]

*#加载appium进程*

**for** i **in** range(2):

host='127.0.0.1'

port = 4723 + 2 \* i

appium=multiprocessing.Process(target=appium\_start,args=(host,port))

appium\_process.append(appium)

**if** \_\_name\_\_ == '\_\_main\_\_':

*#并发启动appium服务*

**for** appium **in** appium\_process:

appium.start()

**for** appium **in** appium\_process:

appium.join()

## Appium端口检测

#### 问题思考

经过前面学习，我们已经能够使用python启动appium服务，但是启动Appium服务之前必须保证对应的端口没有被占用，否则会出现如下报错：

C:\Users\Shuqing>appium -a 127.0.0.1 -p 4723

[Appium] Welcome **to** Appium v1.7.2

[Appium] Non-default server args:

[Appium] address: 127.0.0.1

[HTTP] Could **not** start REST http interface listener. The requested port may already be **in** use. Please make sure there is no other instance of this server running already.

uncaughtException: listen EADDRINUSE 127.0.0.1:4723

针对以上这种情况，我们在启动appium服务前该如何检测端口是否可用呢？对于被占用的端口我们又该如何释放？

#### 需求分析

1. 自动检测端口是否被占用
2. 如果端口被占用则自动关闭对应端口的进程

#### 端口检测

端口检测需要使用到**socket**模块来校验端口是否被占用。

[python socket模块官方文档](https://docs.python.org/3.5/library/socket.html)

**什么是socket？**

网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个socket。建立网络通信连接至少要一对端口号(socket)。

socket本质是编程接口(API)，对TCP/IP的封装，TCP/IP也要提供可供程序员做网络开发所用的接口，这就是Socket编程接口；HTTP是轿车，提供了封装或者显示数据的具体形式；Socket是发动机，提供了网络通信的能力。

例如当你用浏览器打开我要自学网主页时，你的浏览器会创建一个socket并命令它去连接 自学网的服务器主机，服务器也对客户端的请求创建一个socket进行监听。两端使用各自的socket来发送和接收信息。在socket通信的时候，每个socket都被绑定到一个特定的IP地址和端口。

补充资料： [网络工程师视频教程](http://www.51zxw.net/list.aspx?cid=417)

#### 代码实现

check\_port.py

**import** socket

**def** **check\_port**(host, port):

"""检测指定的端口是否被占用"""

*#创建socket对象*

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

**try**:

s.connect((host, port))

s.shutdown(2)

**except** OSError **as** msg:

print('port %s is available! ' %port)

print(msg)

**return** **True**

**else**:

print('port %s already be in use !' % port)

**return** **False**

**if** \_\_name\_\_ == '\_\_main\_\_':

host='127.0.0.1'

port=4723

check\_port(host,port)

**方法**

shutdown(self, flag)：禁止在一个Socket上进行数据的接收与发送。利用shutdown()函数使socket双向数据传输变为单向数据传输。shutdown()需要一个单独的参数， 该参数表示了如何关闭socket

**参数**

* 0表示禁止将来读；
* 1表示禁止将来写
* 2表示禁止将来读和写。

当端口可以使用时，控制台输出如下：此使说明服务端没有开启这个端口服务，所以可用。

C:\Python35\python.exe E:/AppiumScript/advance/appium\_cmd/appium\_multiProcess.py

port 4723 is available!

[WinError 10061] 由于目标计算机积极拒绝，无法连接。

Process finished with **exit** code 0

### 端口释放

如果端口被占用，则需要释放该端口。那么怎么样去释放被占用的端口呢？

#### 代码实现

check\_port.py

**import** os

**def** **release\_port**(port):

"""释放指定的端口"""

*#查找对应端口的pid*

cmd\_find='netstat -aon | findstr %s' %port

print(cmd\_find)

*#返回命令执行后的结果*

result = os.popen(cmd\_find).read()

print(result)

**if** str(port) **and** 'LISTENING' **in** result:

*#获取端口对应的pid进程*

i=result.index('LISTENING')

start=i+len('LISTENING')+7

end=result.index('\n')

pid=result[start:end]

*# 关闭被占用端口的pid*

cmd\_kill='taskkill -f -pid %s' %pid

print(cmd\_kill)

os.popen(cmd\_kill)

**else**:

print('port %s is available !' %port)

**if** \_\_name\_\_ == '\_\_main\_\_':

host='127.0.0.1'

port=4723

*# check\_port(host,port)*

release\_port(port)

控制台显示：

C:\Python35\python.exe E:/AppiumScript/advance/appium\_cmd/appium\_multiProcess.py

netstat -aon | findstr "4723"

TCP 127.0.0.1:4723 0.0.0.0:0 LISTENING 29532

taskkill -f -pid 29532

Process finished with **exit** code 0

## Appium并发测试综合实践

#### 测试场景

并发启动2个appium服务，再并发启动2台设备测试考研帮App

2个appium服务，端口配置如下：

* Appium服务器端口：4723，bp端口为4724
* Appium服务器端口：4725，bp端口为4726

2台设备：

* 127.0.0.1:62025
* 127.0.0.1:62001

测试app：考研帮Andriod版

#### 场景分析

其实就是将前面所讲的两部分组合起来，先启动appium服务，再分配设备启动app。

#### 代码实现

appium\_devices\_sync.py

**from** appium\_sync.multi\_appium **import** appium\_start

**from** appium\_sync.multi\_devices **import** appium\_desired

**from** appium\_sync.check\_port **import** \*

**from** time **import** sleep

**import** multiprocessing

devices\_list=['127.0.0.1:62025','127.0.0.1:62001']

**def** **start\_appium\_action**(host,port):

'''检测端口是否被占用，如果没有被占用则启动appium服务'''

**if** check\_port(host,port):

appium\_start(host,port)

**return** **True**

**else**:

print('appium %s start failed!' %port)

**return** **False**

**def** **start\_devices\_action**(udid,port):

'''先检测appium服务是否启动成功，启动成功则再启动App,否则释放端口'''

host='127.0.0.1'

**if** start\_appium\_action(host,port):

appium\_desired(udid,port)

**else**:

release\_port(port)

**def** **appium\_start\_sync**():

'''并发启动appium服务'''

print('====appium\_start\_sync=====')

*#构建appium进程组*

appium\_process=[]

*#加载appium进程*

**for** i **in** range(len(devices\_list)):

host='127.0.0.1'

port = 4723 + 2 \* i

appium=multiprocessing.Process(target=start\_appium\_action,args=(host,port))

appium\_process.append(appium)

*# 启动appium服务*

**for** appium **in** appium\_process:

appium.start()

**for** appium **in** appium\_process:

appium.join()

sleep(5)

**def** **devices\_start\_sync**():

'''并发启动设备'''

print('===devices\_start\_sync===')

*#定义desired进程组*

desired\_process = []

*#加载desired进程*

**for** i **in** range(len(devices\_list)):

port = 4723 + 2 \* i

desired = multiprocessing.Process(target=start\_devices\_action, args=(devices\_list[i], port))

desired\_process.append(desired)

*#并发启动App*

**for** desired **in** desired\_process:

desired.start()

**for** desired **in** desired\_process:

desired.join()

**if** \_\_name\_\_ == '\_\_main\_\_':

appium\_start\_sync()

devices\_start\_sync()

补充资料：[谈谈TCP中的TIME\_WAIT](https://blog.csdn.net/jewes/article/details/52654997)

netstat -ano |findstr 4723

TCP 127.0**.0.1**:4723 127.0**.0.1**:63255 TIME\_WAIT 0

TCP 127.0**.0.1**:4723 127.0**.0.1**:63257 TIME\_WAIT 0

TCP 127.0**.0.1**:4723 127.0**.0.1**:63260 TIME\_WAIT 0

TCP 127.0**.0.1**:62998 127.0**.0.1**:4723 TIME\_WAIT 0

port 4723 is available

### 并发用例执行

#### 测试场景

再上面的场景基础之上，并发启动设备后然后执行跳过引导页面操作。

#### 代码实现

kyb\_test.py

**from** selenium.common.exceptions **import** NoSuchElementException

**class** **KybTest**(object):

**def** **\_\_init\_\_**(self,driver):

self.driver=driver

**def** **check\_cancelBtn**(self):

print('check cancelBtn')

**try**:

cancelBtn = self.driver.find\_element\_by\_id('android:id/button2')

**except** NoSuchElementException:

print('no cancelBtn')

**else**:

cancelBtn.click()

**def** **check\_skipBtn**(self):

print('check skipBtn')

**try**:

skipBtn = self.driver.find\_element\_by\_id('com.tal.kaoyan:id/tv\_skip')

**except** NoSuchElementException:

print('no skipBtn')

**else**:

skipBtn.click()

**def** **skip\_update\_guide**(self):

self.check\_cancelBtn()

self.check\_skipBtn()

将执行的用例集成到 multi\_devices.py

from appium **import** webdriver

**import** yaml

from time **import** ctime

from appium\_sync.kyb\_test **import** KybTest

**with** open('desired\_caps.yaml','r') **as** file:

**data**=yaml.load(file)

devices\_list=['127.0.0.1:62001','127.0.0.1:62025']

def appium\_desired(udid,port):

desired\_caps={}

desired\_caps['platformName']=**data**['platformName']

desired\_caps['platformVersion']=**data**['platformVersion']

desired\_caps['deviceName']=**data**['deviceName']

desired\_caps['udid']=udid

desired\_caps['app']=**data**['app']

desired\_caps['appPackage']=**data**['appPackage']

desired\_caps['appActivity']=**data**['appActivity']

desired\_caps['noReset']=**data**['noReset']

print('appium port:%s start run %s at %s' %(port,udid,ctime()))

driver=webdriver.Remote('http://'+str(**data**['ip'])+':'+str(port)+'/wd/hub',desired\_caps)

driver.implicitly\_wait(5)

k=KybTest(driver)

k.skip\_update\_guide()

return driver

**if** \_\_name\_\_ == '\_\_main\_\_':

appium\_desired(devices\_list[0],4723)

appium\_desired(devices\_list[1],4725)

## 基于Docker+STF Appium并发测试

[Docker](https://docs.docker.com)

[STF](https://openstf.io/)

实践案例：<https://github.com/haifengrundadi/DisCartierEJ>

#### 参考资料

* <https://testerhome.com/topics/2918>
* <https://testerhome.com/topics/7917>
* <https://baike.baidu.com/item/socket/281150?fr=aladdin>
* <http://www.runoob.com/python/os-popen.html>
* <https://blog.csdn.net/jewes/article/details/52654997>