

Appium Girls学习指南

WeiDong H

Published
with GitBook



目錄

Introduction	0
Appium 介绍	1
安装 Appium	2
配置IDE	3
启动Appium Server	4
学习 Desired Caps	5
元素定位	6
编写脚本	7
手势操作	8
进阶之UIAutomator	9
进阶之Hybrid	10
框架开发	11
附录 - 下载篇	12

Appium Girls 学习手册

本手册适用于刚接触Appium，初次把玩Appium的同学。 为了达到更好的学习效果，我们希望你能具备如下基础：

- Android开发（布局）基础认识
- Android SDK工具的熟悉了解
- 知道如何使用命令行
- 略Java或Python的语言基础

准备工作：

- 1、搭建Java环境、Android开发环境
- 2、安装Appium
- 3、安装Eclipse

学习范围：

通过本手册的学习，你能够学习到Appium的基本架构和实现，以及如何利用Appium进行Android UI自动化脚本编写。

Appium 介绍

Appium 是什么？

Appium介绍

Appium是一个移动端的自动化框架，可用于测试原生应用，移动网页应用和混合型应用，且是跨平台的。可用于IOS和Android以及firefox的操作系统。原生的应用是指用android或ios的sdk编写的应用，移动网页应用是指网页应用，类似于ios中safari应用或者Chrome应用或者类浏览器的应用。混合应用是指一种包裹webview的应用,原生应用于网页内容交互性的应用。重要的是Appium是跨平台的，何为跨平台，意思就是可以针对不同的平台用一套api来编写测试用例。

Appium的哲学

Appium遵循下面几个原则（其实也是appium的特点）：1.使用自动化来测试一个app，但是不需要重新编译它 2.写自动化case，不需要学习特定的语言 3.一个自动化框架不需要重复造轮子 4.一个自动化框架需要开源，在精神和实践上实现开源

Appium的设计

为了遵循上面的原则，appium的解决方法分别如下：第一条：采用底层驱动商提供的自动化框架。

IOS:苹果的UIAutomation Android 4.2+:谷歌的 UiAutomator Android 2.3+:谷歌的 Instrumentation（已被selendroid取 第二条：采用底层驱动商提供统一API，就是WebDriver API。

WebDriver(也称Selenium WebDriver)其实是一个C/S架构的协议，叫做JSON Wire Protocol。通过这个协议，用任何语言写成的客户端都可以发送HTTP请求给服务器。这就意味着你可以自由选择你想要使用的测试框架和执行器，也可以将任何包含HTTP客户端的库文件加入到你的代码中。换句话说，Appium的WebDriver不是一个技术上的测试框架，而是一个自动化库。第三条：因为WebDriver是一个非常**的网页协议且已经正在起草W3C的标准。我们为什么还要创造其他东西呢？相反，我们在WebDriver的基础上，扩展了一些适合移动端自动化协议的API。

第四条：你之所以能读到这篇文章，就是因为我们开源啦。

Appium概念

C/S 架构

Appium的核心是一个遵守REST设计风格的web 服务器，它接受客户端的连接，接收客户端的命令，在手机设备上执行命令，然后通过HTTP的响应收集命令执行的结果。这种架构给我们提供了很好的开放特性：只要某种语言有http 客户端的api，我们就可以通过这个语言写我们的测试代码，当然了为了方便大家使用，提供了如下的客户端库供使用：

这里写图片描述 我们还可以将服务器放到远端，比如云里，这样我们可以借助云服务来接受命令以及解析命令。

Session

自动化的过程通常在session上下文中执行。客户端初始化一个session会话，虽然不同的语言初始化的方式不同，但是他们都要发送POST/session 请求到服务器端，这些请求里面都会带有一个对象：desired capabilities ,这个时候服务器端会启动自动化session然后返回一个session ID，以后的命令都会用这个session ID去匹配。

Desired Capabilities

desired capabilities 这个对象其实是一个key-value的集合，里面包含了各种各样的信息，发送到服务器端后，服务器解析这些信息就知道了客户端对哪种session感兴趣，然后就会启动相应的session。这里面的信息会影响着服务器端启动session的类型。比如你platformName的值为ios,就是告诉服务器启动一个ios的session，而不是android session。如果safariAllowPopups的值为true,这是告诉safari类的自动化session，可以使用js打开新窗口。具体信息查看capabilities doc 详细了解。

Appium Server

Appium server使用node.js写的http服务器，遵守REST 风格

安装 Appium

在这一章，我们会学习如何搭建和安装Appium

安装步骤

- 安装JDK 并设置环境变量
 - 安装Android SDK 并设置环境变量
 - 安装Nodejs
 - 安装appium
 - 验证安装
-

安装JDK 并设置环境变量

到Java官网下载相应的JDK并安装

设置环境变量

添加 `JAVA_HOME` 对应的路径 `C:\Program Files\Java\jdk1.7.0_79`

在 `path` 变量添加 `;%JAVA_HOME%\bin;`

添加 `CLASSPATH` 设置值为 `%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar`

设置完毕后在command line输入 `java -version`

显示如下内容说明配置正确

```
java version "1.7.0_79"
```

```
Java(TM) SE Runtime Environment (build 1.7.0_79-b15)
```

```
Java HotSpot(TM) 64-Bit Server VM (build 24.79-b02, mixed mode)
```

安装Android SDK 并设置环境变量

注意：安装Android SDK需要翻墙并更新你的SDK repository

到[Android 官网](#)下载SDK并安装

```
设置环境变量，添加 ANDROID_HOME 设置值为：C:\你的安装路径\Android\sdk  
在 path 环境变量值中添加：  
加：;%ANDROID_HOME%\tools;%ANDROID_HOME%\platform-tools;
```

验证安装 在命令行输入 `adb devices`

如果结果如下，说明配置成功：

```
List of devices attached  
* daemon not running. starting it now on port 5037 *  
* daemon started successfully *
```

安装Nodejs

到[Nodejs官网](#)下载最新版本的NodeJs并直接安装即可。

安装完毕后，打开终端/CMD（命令行），输入 `node -v`，不报错说明安装成功。

安装appium

安装appium有两种方式：

- 使用 **npm(Node js 的管理与分发工具)**安装**Appium**
- 使用**Appium**官方安装包安装

使用NPM安装Appium

首先，npm默认的镜像地址已经被墙，我们需要将npm的下载地址更改为国内的地址。

打开终端（命令行），输入

```
npm config get registry
```

我们可以看到当前npm的镜像地址。

我们需要将这个地址替换为国内的地址，这里我们替换成淘宝的NPM镜像源：

```
npm config set registry=https://registry.npm.taobao.org/
```

敲击回车后，我们再次输入

```
npm config get registry
```

可以确认镜像源是否替换成功。

接下来我们就可以安装Appium了：

```
npm install -g appium
```

需要注意的是，最新版本的Appium在安装过程中，会去google拉取最新的chromedriver，因为google被墙的关系，你很可能无法下载。这里强烈建议各位学习如何翻墙。

使用Appium官方安装包安装

到[Appium 官网](#) 下载和你所使用系统一致的版本进行安装。

验证安装

当确认Appium安装完毕后，我们可以通过 `appium-doctor` 的命令来检查当前appium安装是否完善，当前的JDK、SDK等环境是否配置正确。

如果 `appium-doctor` 返回的内容是有错的，请根据返回的具体的提示，将你的环境搭建完善。

如果返回的结果类似如下，说明安装成功

```
...
...
Android Checks were successful.
All Checks were successful
```

需要注意的是，如果你是通过安装包安装的，使用 `appium-doctor` 命令时尽量切换到 `C:\Program Files (x86)\Appium\node_modules.bin` 目录或者将该路径放到环境变量 `path` 中

思考与总结

应该使用哪种安装方式，哪种方式比较好？

相关资料

Appium for Windows环境搭建：<http://www.cnblogs.com/tobecrazy/p/4562199.html>

Appium for iOS 环境搭建：<http://www.cnblogs.com/tobecrazy/p/4970188.html>

请参 [附录-下载篇](#)

配置IDE

注意：你可能已经完成了这一步，如果那样的话，你可以直接进入下一章节。

你马上就要开始写下你的代码了。对于编辑代码，有很多不同的编辑器，通常根据个人偏好选择。大部分 Java 程序员喜欢使用集成开发环境（IDE），它们针对代码这种特殊的纯文本有很好的优化，如代码高亮、自动补全。

下面是我们的建议，但是你可以随时咨询你的教练。那样会更容易得到他们的帮助。

Eclipse

Eclipse是 IBM 出品的著名的跨平台开源集成开发环境（IDE）。最初主要用来Java语言开发，目前亦有人通过插件使其作为C++、Python、PHP等其他语言的开发工具。

Android 开发工具 Android Developer Tools（ADT）就是基于 Eclipse 开发的。

下载方式：

[点此进入下载页面](#)，在页面右侧根据你的系统选择合适的版本即可。

IDEA

IDEA 是现在比较流行的 Java 集成开发环境。相比 Eclipse，稳定性、易用性方面会更加突出。

悄悄告诉你，现在最新的 Android 开发工具 Android Studio 就是基于 IDEA 开发的哦~

下载方式：

[点此进入下载页面](#)。其中 community 是社区版，是免费的。而 Ultimate 则是收费的。

为了保持统一，后续我们将以 Eclipse 作为主要的 IDE 工具。

安装 Eclipse

Eclipse 是使用 Java 编写的程序，而我们在前一章已经安装好 JDK 了，因此可以直接安装 Eclipse 了。

1. [进入下载页面](#)下载 Eclipse
2. 下载完成后你会得到一个压缩包，此时你需要做唯一一件事情就是：解压缩。

3. 解压缩后的文件夹里面会有一个名为 Eclipse 的文件。双击它就可以打开了。

第一次打开会弹出一个设置工作空间的弹窗。此时请选择你想存放接下来编写的脚本的目录位置并记住它。后面在 Eclipse 编辑的一切代码都将会存储在这里。

配置 Appium Java Client

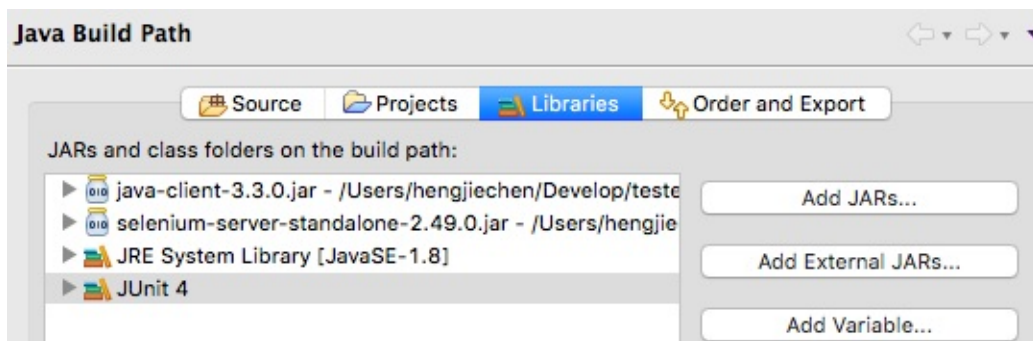
Java 本身是一门语言，它内置了很多的方法可以读取文件内容（Read）、写入文件内容（Write）等。然而它本身并不认识 Appium，因此需要一座桥梁来让我们的 Java 代码认识它，这座桥梁就是 Appium Java Client。

Appium Java Client 是基于 Selenium 的 Java Client 编写的一个专用于与 Appium 进行通讯的库。

具体安装步骤主要有3步：

1. 下载 [Appium java client](#) 及 [Selenium Java standalone server](#) 两个库对应的 jar 包
2. 在 Eclipse 中新建 Java 项目
3. 在项目的 build path 中通过“Add External Jars”添加第一步下载的两个 jar 文件，以及通过“Add Library”添加 JUnit4 这个 Library。我们后续会用到。

添加完成后的 build path 如下图：



本章节相关软件下载链接汇总

- Eclipse : <http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/mars1>
- Appium java client : <http://search.maven.org/remotecontent?filepath=io/appium/java-client/3.3.0/java-client-3.3.0.jar>
- Selenium Java standalone server : <http://npm.taobao.org/mirrors/selenium/2.49/selenium-server-standalone-2.49.0.jar>

启动 Appium Server

前面配置了这么多，是时候来动手启动你的 Appium Server 了。

根据前面的安装方式，一共有 2 种启动 appium server 的方式。

从命令行启动

无论你使用哪种安装方式，最终都是通过命令行启动的。因此，我们先来学习如何用命令行启动。

- 通过 npm 安装的 appium

对于这种方式，启动非常简单，只需要运行一个命令：

```
$ appium
info: Welcome to Appium v1.4.16 (REV ae6877eff263066b26328d457bd285c0cc62430d)
info: Appium REST http interface listener started on 0.0.0.0:4723
info: Console LogLevel: debug
```

此时 appium 就启动起来了！

现在，我们来解读一下这三行信息的意义：

```
info: Welcome to Appium v1.4.16 (REV ae6877eff263066b26328d457bd285c0cc62430d)
```

表明目前启动的 appium 版本。在这里我们可以看到使用的版本是 `1.4.16`，对应版本库里的 `reversion` 号是 `ae6877eff263066b26328d457bd285c0cc62430d`

此处的版本信息可能与你的不一致，没关系，只要是 1.4.0 以上的版本都可以继续进行下去。

```
info: Appium REST http interface listener started on 0.0.0.0:4723
```

说明监听的地址及端口。这个地址就是 Appium Server 的入口，`0.0.0.0` 表示监听本机所有 ip 地址。我们将会在后续编写脚本时用到它。

```
info: Console LogLevel: debug
```

表示我们当前的日志级别是 `debug`。这也是 appium 日志的默认级别。

若要退出，同时按下键盘的 `ctrl` 键和 `c` 键即可。

- 通过图形化界面安装的 appium

Windows:


假设 appium 安装在默认路径 `C:\Program Files (x86)\Appium`，那么可以通过下面的命令从命令行启动 appium：

```
C:\Users\AppData\Local>"C:\Program Files (x86)\Appium\node.exe" "C:\Program Files (x86)\Appium\node_modules\appium\bin\appium.js"
```

Mac：

假设 appium 安装在默认路径 `/Applications`，那么可以通过下面的命令从命令行启动 appium：

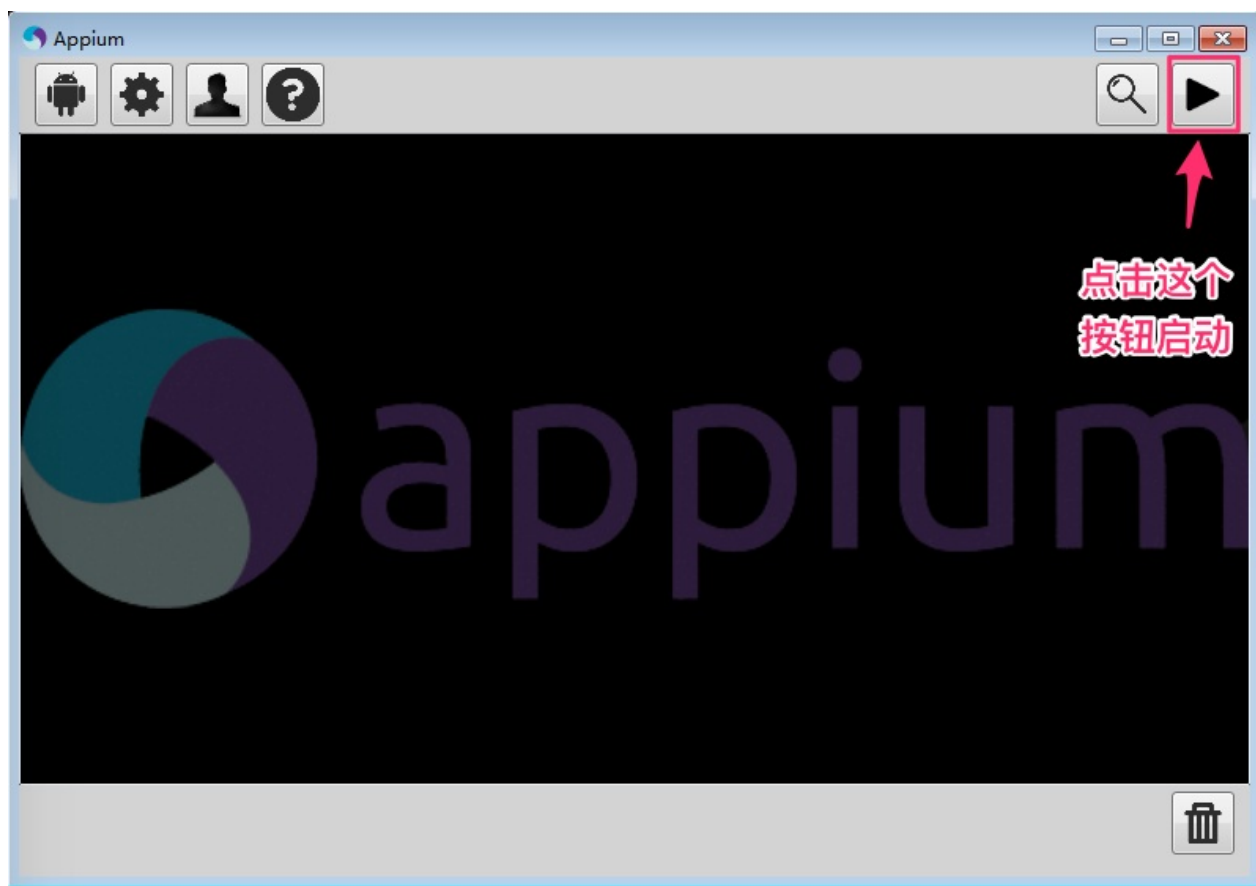
```
$ "/Applications/Appium.app/Contents/Resources/node/bin/node" "/Applications/Appium.app/C
```



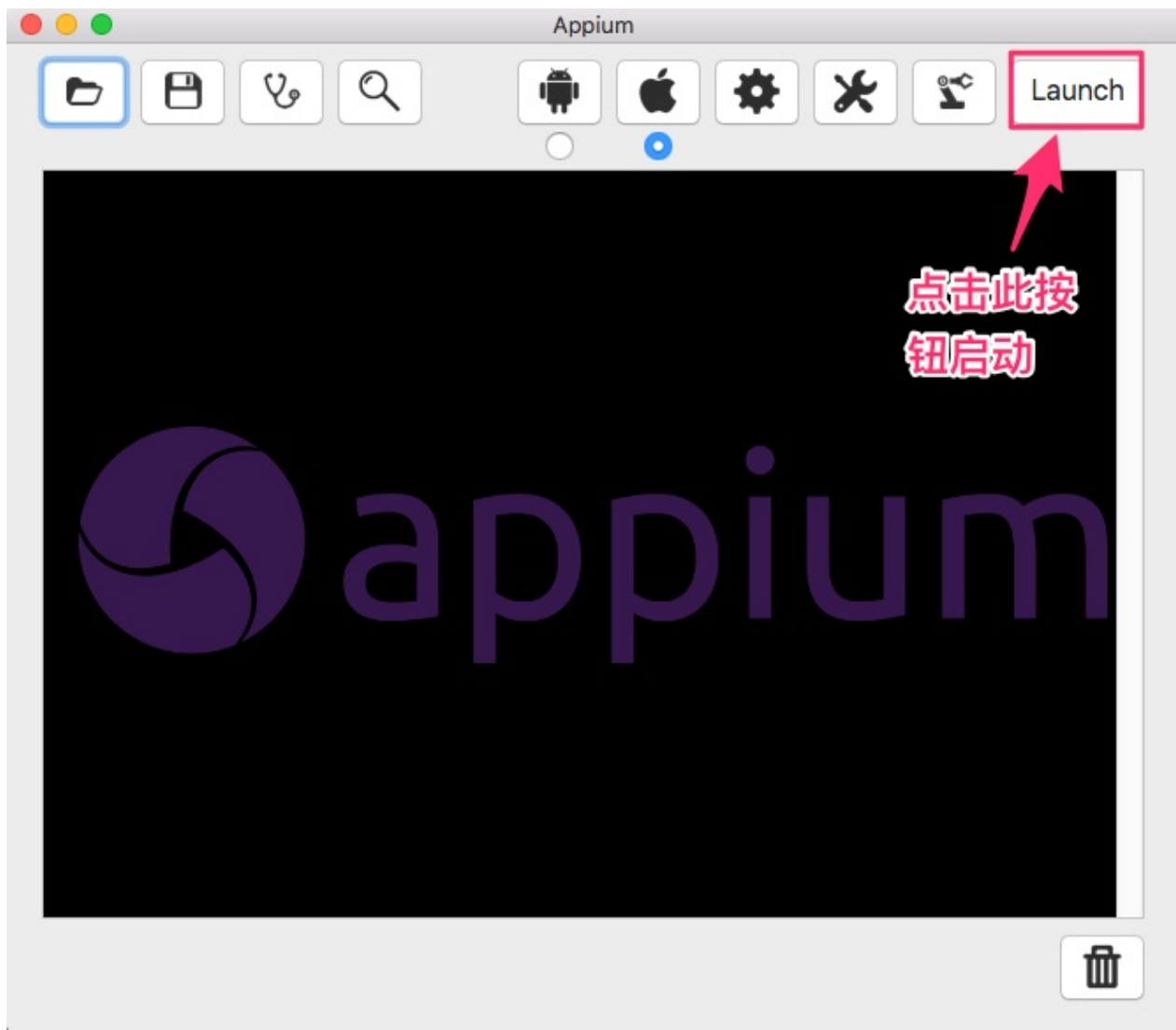
从图形化界面启动 appium

这个就简单多了。只需要点击一个按钮。

Windows：



Mac:



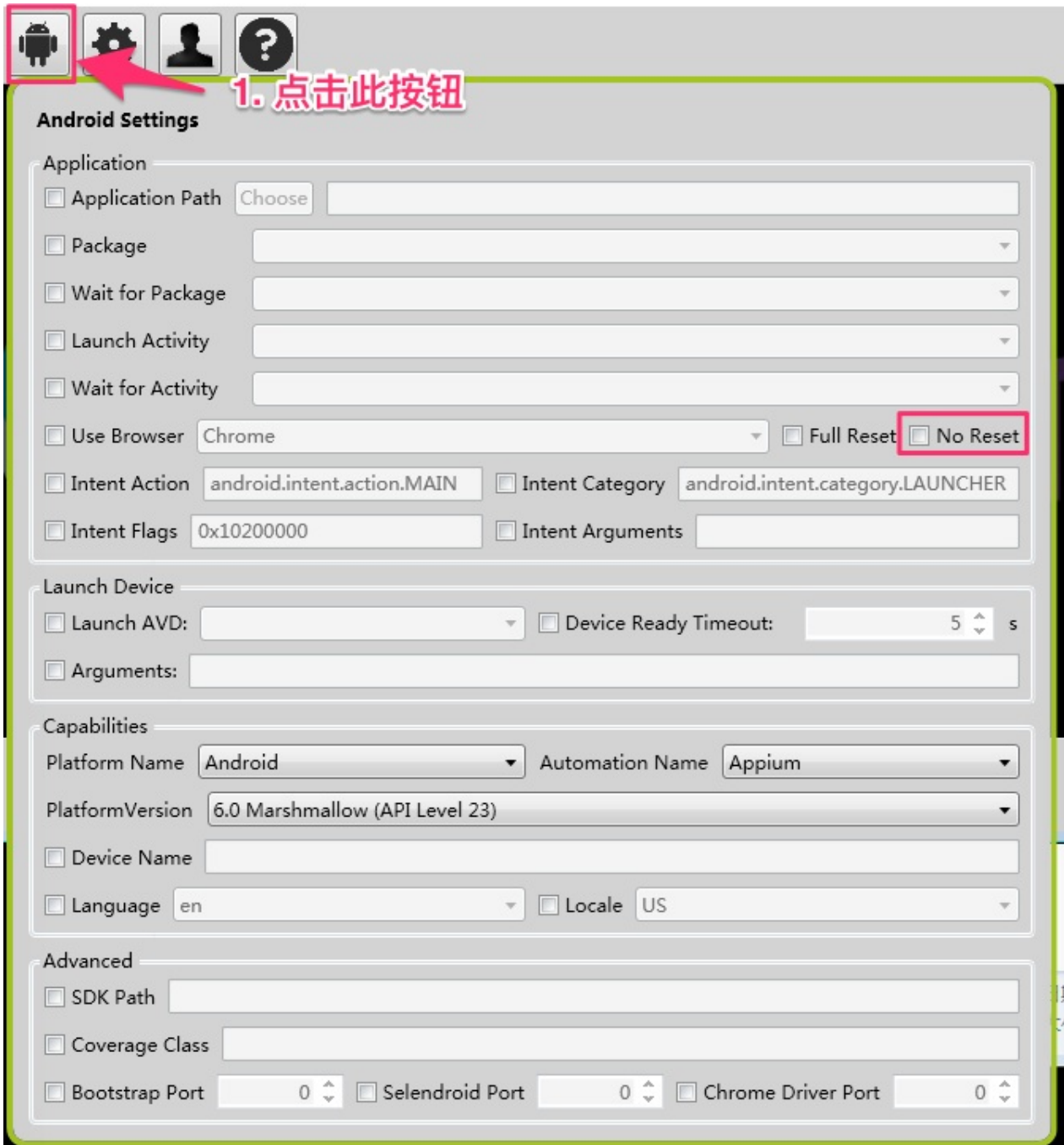
常用 Server Argument 讲解

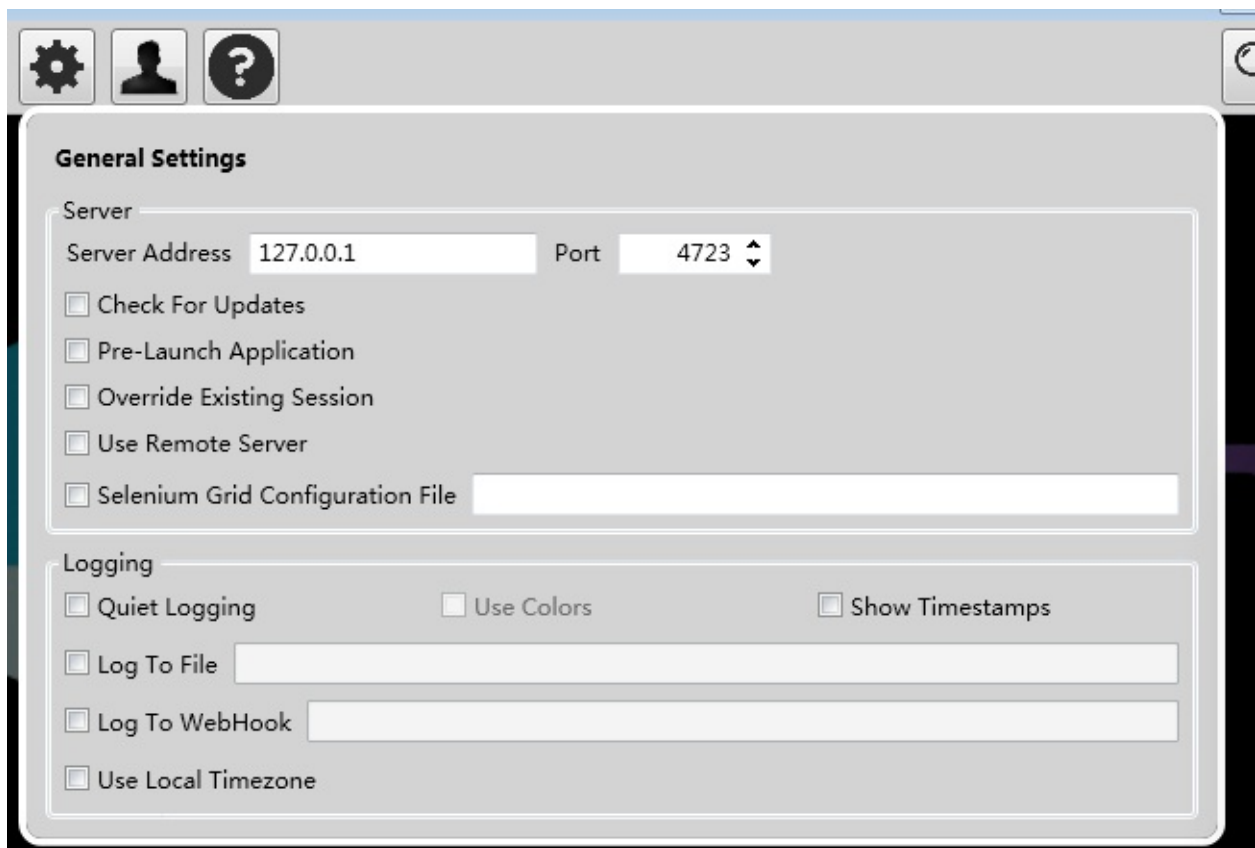
前面讲述的是采用默认参数启动 appium server。但实际使用中默认参数并不总能满足我们需要，因此我们需要手动配置这些参数。

由于时间关系，我们只讲述最常用的几个参数。若希望了解所有参数，可以查阅 [appium 官方文档](#)

标志	默认值	描述	例子
<code>-a, --address</code>	0.0.0.0	监听的 ip 地址。注意在图形化界面上默认值为 127.0.0.1	<code>--address 0.0.0.0</code>
<code>-p, --port</code>	4723	监听的端口。需要启动多个 appium server 进行并行测试时需要保证每个 server 的监听端口不一样。	<code>--port 4723</code>
<code>--log-timestamp</code>	false	在日志输出里显示时间戳	
<code>--local-timezone</code>	false	在日志输出的时间戳使用本地时间	
<code>-q, --log</code>	null	将日志输出到指定文件	<code>--log /path/to/appium.log</code>
<code>--session-override</code>	false	允许 session 被覆盖 (冲突的话)	
<code>--command-timeout</code>	60	默认所有会话的接收命令超时时间 (在超时时间内没有接收到新命令, 自动关闭会话)。会被新的超时时间覆盖	

这些参数同时也可以可以在图形化界面上配置：



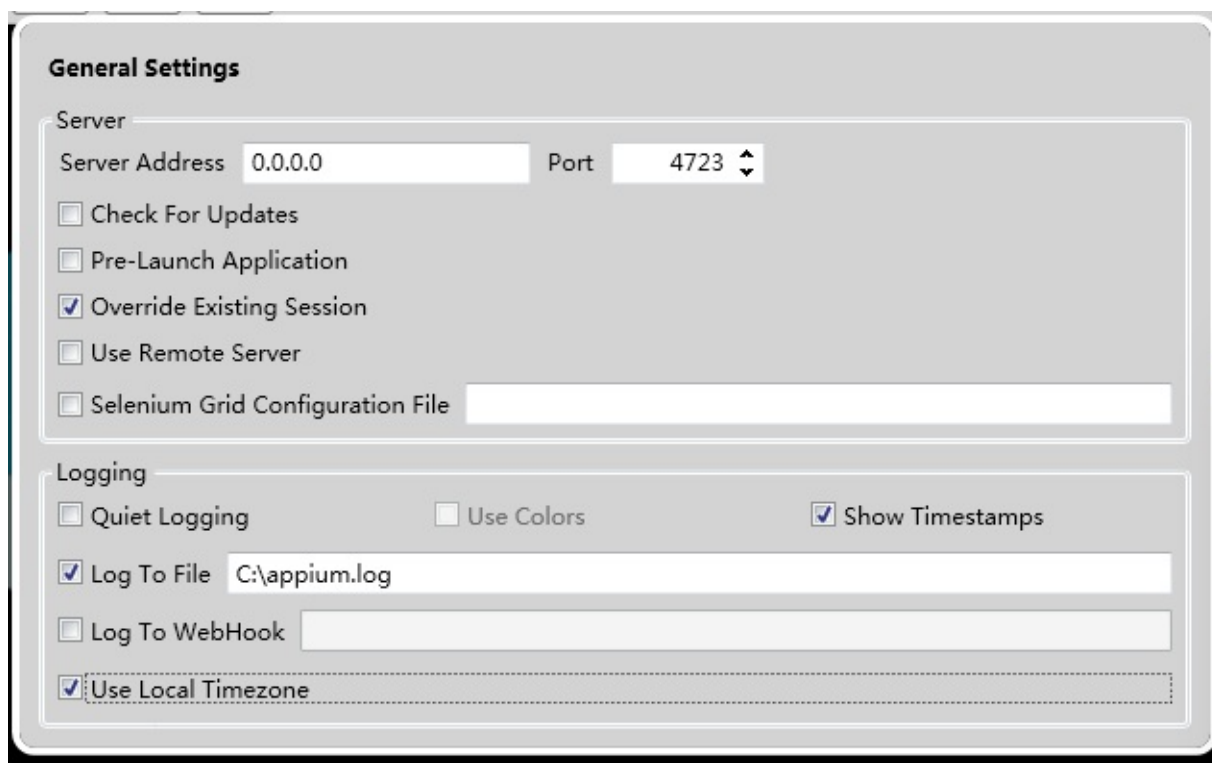


接下来，关闭你的 appium server，使用如下配置再次启动 appium server：

```
$ appium --address 0.0.0.0 --port 4723 --log "C:\appium.log" --log-timestamp --local-time
```

若你使用 mac，请把日志位置从 "C:\appium.log" 改为 "~/appium.log"

使用图形化界面的同学请参照下图修改你们界面的配置。它等价于上面命令行里的添加的参数：



小活动

使用图形化界面的同学，发动你的大脑，找一下怎么在图形化界面里面进行配置，使其等价于命令行参数 `--command-timeout 120` ？

学习 DesiredCapabilities

DesiredCapabilities 简介

Desired Capabilities 携带了一些配置信息，在启动session的时候是必须提供，如启动模式、apk/app配置、package/activity配置、浏览器配置、键盘配置等。从本质上讲是一组 key-value形式的对象,你可以理解成是java里的json对象。（你可以在selenium-api-2.49.0-sources.jar中查看CapabilityType的声明）

Desired Capabilities 关键字

Desired Capabilities的重要作用是在启动时传递信息给Appium Server，。可以粗略的分为两类：设备信息类和应用信息类。

- 设备信息：设备是真机还是模拟器，手机操作系统以及版本等。
- 应用信息：要进行浏览器测试还是移动端测试？如果是移动应用，安装文件apk或者app文件的位置？如果是浏览器测试，浏览器的类型是什么？

下表中列举了Appium常用的部分关键字。更多详情,可以搜索Appium服务关键字。

关键字	描述	实例
platformName	手机操作系统	iOS,Android,FirefoxOS
platformVersion	手机操作系统版本	例如：7.1, 4.4
deviceName	手机类型或模拟器类型	iPhone Simulator, iPad Simulator, Android Emulator, Galaxy S4等。在 iOS 上，这个关键字的值必须是使用 instruments -s devices 得到的可使用的设备名称之一。在 Android 上，这个关键字目前不起作用。
app	.ipa or .apk(也可以是包含他们的zip)文件所在的本地绝对路径或者远程路径	Appium会先尝试安装路径对应的应用在适当的真机或模拟器上。针对Android系统，如果你指定app-package和app-activity(具体见下面)的话，那么就可以不指定app。
browserName	需要进行自动化测试的手机 web 浏览器名称。如果是对应用进行自动化测试，这个关键字的值应为空。	iOS 系统上可以用 'Safari'，Android 系统上可以用 'Chrome'，'Chromium'，或 'Browser'。
automationName	自动化测试引擎	Appium,Selendroid
appActivity	要从应用包中启动的 Android Activity 名称。	它通常需要在前面添加 . (如：使用 .MainActivity 而不是 MainActivity) MainActivity, .Settings
appPackage	你想运行的Android应用的包名	比如com.example.android.myApp
appWaitActivity	你想要等待启动的 Android Activity 名称	SplashActivity
unicodeKeyboard	true	Appium 1.3.3以上的版本,支持中文输入
resetKeyboard	true	Appium 1.3.3以上的版本,支持中文输入

Session简介

Session 是指一个终端用户（从注册进入到注销退出）与交互系统进行通信的会话，用于保持状态的基于 Web服务器的方法。将Appium理解为Server端，客户端设备发起command的必须是在Session start后才可以进行的。一般来说，通过POST /session这个URL，然后传入Desired Capabilities就可以开启session了。

开启session后，会返回一个全局唯一的sessionid，以后几乎所有的请求都必须带上这个session id，这个session id代表了你所打开的浏览器或者是移动设备的模拟器。

Desired Capabilities使用

• 方法一：

参见appium-java-workshop中的代码，在 `setUp()` 方法中依次设置必须的Capabilities值，如下：

```
// set up appium
DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability("deviceName", "iOS Simulator");
capabilities.setCapability("platformName", "iOS");
capabilities.setCapability("platformVersion", "8.1");
capabilities.setCapability("app", "/Users/color/App/BOCOP.app");
// if no need install don't add this
```

• 方法二：

Capabilities理解为一组key-value形式的json对象,将不同设备系统的Capabilities保存到配置文件然后读取，可以避免在每个脚本 `setUp()` 方法中重复编写。（仅作为框架设计的一种思路，如果你已经熟悉测试框架，可以忽略该部分。）

```
@BeforeMethod
public void setUp(@Optional("ios") String PlatformName, @Optional("web") String AppTy
    super.setUp(PlatformName, AppTypeName);
}
```

将Session启动所需要的配置，用json字符串保存在配置文件appium.properties中。通过通过 `setUp()` 的参数，读取相应的配置信息。

```
//反斜杠\是续行符
IOS_Simulator_Web={"device":"iPhone Simulator",\
  "deviceName":"iPhone Simulator",\
  "platformName":"iOS","platformVersion": "7.1",\
  "browserName": "safari",\
  "app": "safari"}

IOS_Mobile_NATIVE={"device":"iPhone",\
  "deviceName":"iPhone Mobile",\
  "platformName": "iOS",\
  "app":"${user.dir}/src/main/resources/Apps/BOCOP.app"}

ANDROID_Mobile_NATIVE={\
  "device":"Android",\
  "deviceName":"Android Mobile",\
  "platformName": "Android",\
  "platform":"MAC",\
  "platformVersion": "4.4.2",\
  "app":"${user.dir}/src/main/resources/Apps/ContactManager.apk",\
  "appPackage":"com.example.android.contactmanager",\
  "appActivity": ".ContactManager"\
}
```

由配置文件中的json字符串，返回一个DesiredCapabilities对象，在启动将该对象前赋值给AppiumDriver。通过json字符串获取DesiredCapabilities对象的方法如下：

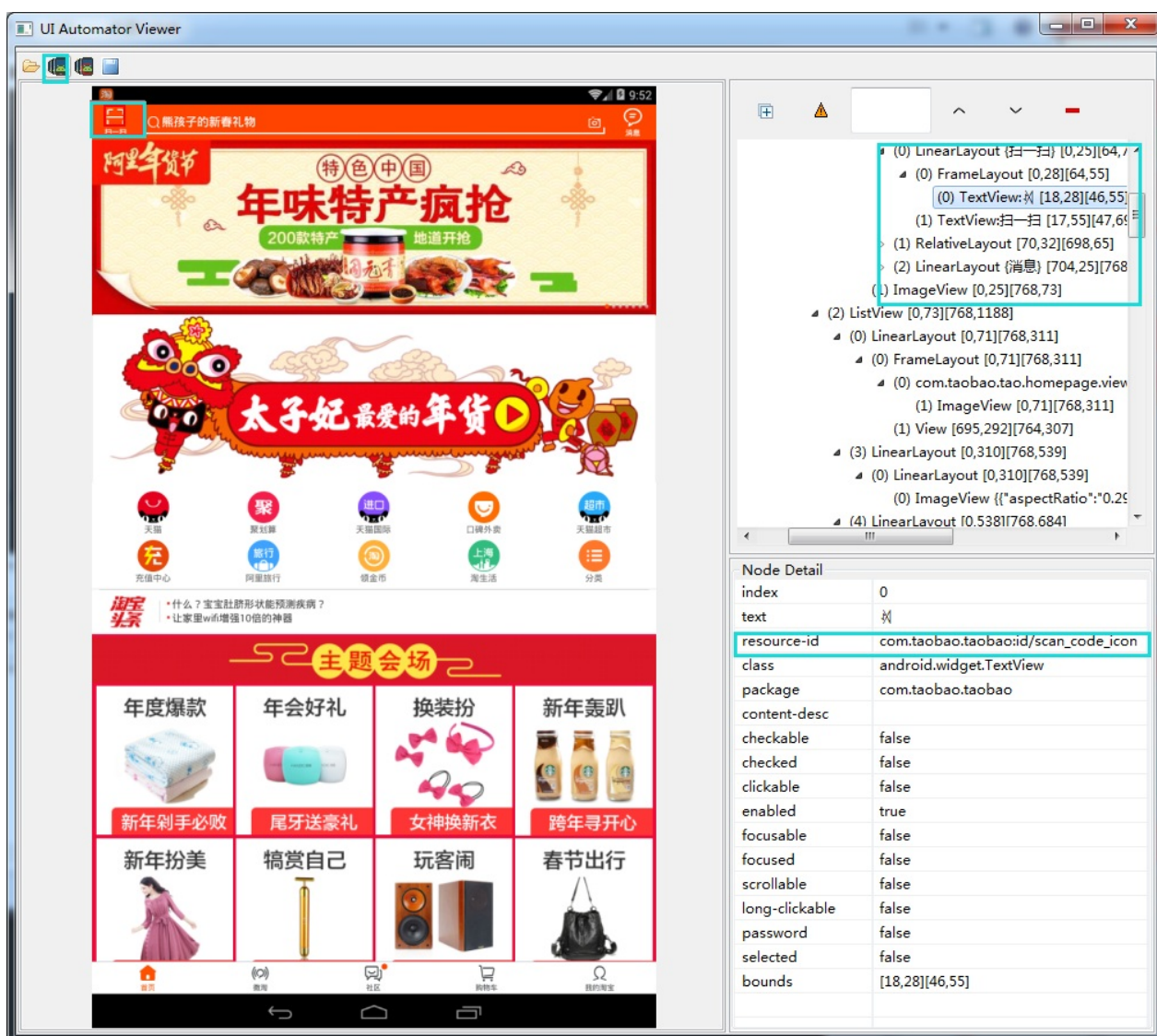
```
private DesiredCapabilities SetCapabilities(  
    DesiredCapabilities capabilities, String JsonCapabilities) {  
    if (JsonCapabilities == null || "".equals(JsonCapabilities))  
        return capabilities;  
    try {  
        JSONObject jsonObj = JSONObject.parseObject(JsonCapabilities);  
        if (!jsonObj.isEmpty()) {  
            if (capabilities == null) {  
                capabilities = new DesiredCapabilities();  
            }  
            for (Entry<String, Object> element : jsonObj.entrySet()) {  
                capabilities.setCapability(element.getKey(),  
                    element.getValue());  
                System.out.println(element.getKey() + "----"  
                    + element.getValue());  
            }  
        }  
    }  
    catch (Exception e) {  
        throw new Error("Please input a JSON format string");  
    }  
    return capabilities;  
}
```


元素定位

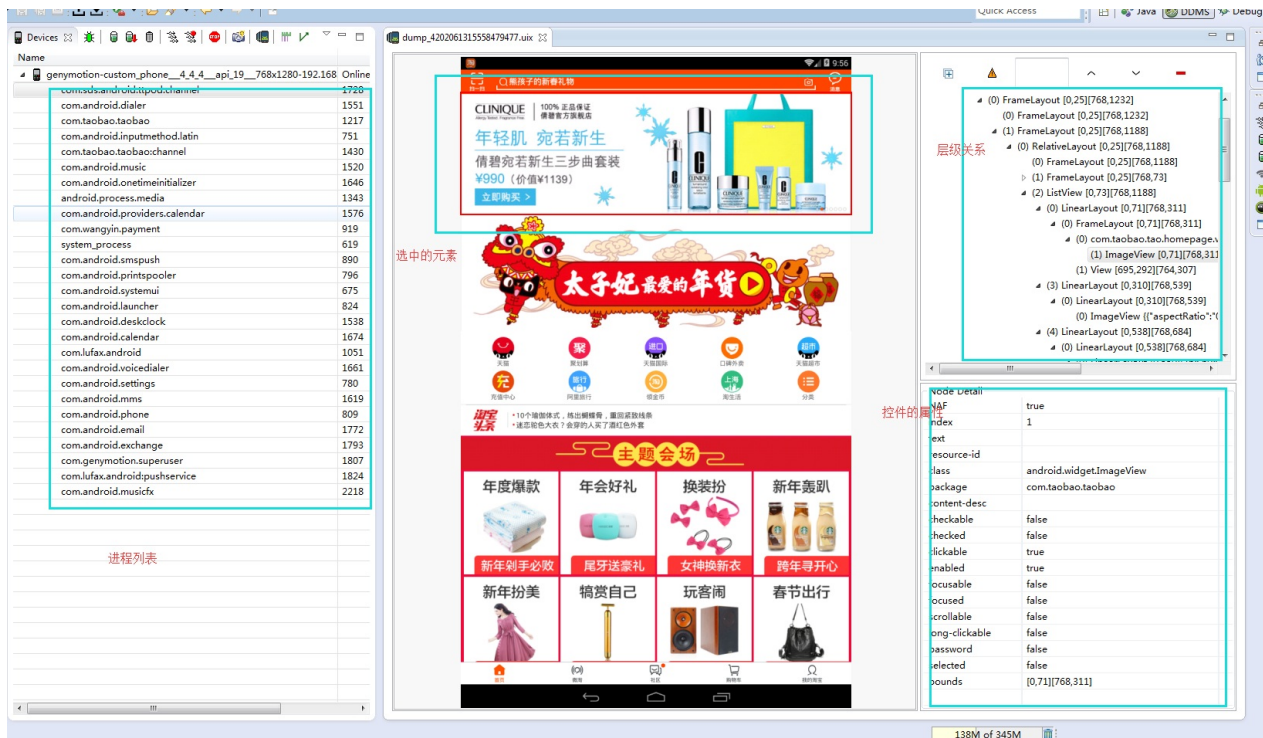
uiautomatorviewer是SDK自带的一个APP元素查看工具，通过这个工具我们可以查看一下App view上面的元素的属性，然后再利用Robotium solo的各种API来对元素进行定位操作。下面我们介绍一下

UIAutomatorViewer如何使用

- 将测试机连接到电脑上，并且将被测试的**App**签名后安装在的测试机上。
- 打开我们安装的**SDK**文件夹，在\sdk\tools下找到批处理文件uiautomatorviewer.bat，运行这个文件



还有一种方式是通过 ADT内的DDMS打开



find_elements方法讲解

常用的几种定位方法：

直接文本定位

```
driver.findElementByAndroidUIAutomator("new UiSelector().text(\"Add note\")");
```

元素定位方式介绍分析：

通过id定位

```
driver.findElement(By.id("id is me"));
```

通过name定位

```
driver.findElement(By.id("name is me"));
```

通过className定位

```
driver.findElement(By.id("className is me"));
```

通过xpath定位

```
driver.findElementByXPath("//android.widget.TextView[contains(@text,'is xpathname')]");
```

List遍历：

```
List<WebElement> textFieldsList = driver.findElementsByClassName("android.widget.EditTex  
textFieldsList.get(0).sendKeys("Some Name");  
textFieldsList.get(2).sendKeys("Some@example.com");  
driver.findElementByName("Save").click();
```

LinkText

```
driver.findElementByClassName("android.widget.TextView");
```

定位不到怎么办？

desc

```
driver.findElementByAndroidUIAutomator("new UiSelector().descriptionContains(\"\"+name+\"\"
```

组合定位：

```
driver.findElement(By.className(className)).findElements(By.tagName("tagname is me")).get  
driver.findElement(By.className(className)).findElements(By.id("id is me")).get(i)  
driver.findElement(By.className(className)).findElements(By.name("name is me")).get(i)
```

List遍历判断：

```
List<WebElement> textFieldsList = driver.findElementsByClassName("android.widget.Edit  
    for(int i=0;i<textFieldsList.size();i++)    {  
        if(textFieldsList.get(i).equals("value")){  
            textFieldsList.get(i).click();  
        }  
    }  
}
```

坐标：

```
driver.tap(1, 540, 960, 500)
```


编写脚本

恭喜你！来到这里就说明你已经知道怎么通过代码来打开和控制应用了！但如果需要让这些代码能稳定地运行上千遍，并有不错的测试报告告诉你用例是否通过，我们还需要加入一些额外的代码。

在这一章，我们将会学习下面的内容：

1. 编写你的第一个测试脚本
2. 使用 Junit 组织你的脚本
3. 在脚本中加入隐式等待，应对不稳定的网络环境

准备工作

首先，在我们之前在 Eclipse 里面创建的项目里面添加一个名为 `app` 的文件夹，并把 `ToDoList` 应用放到里面。我们接下来将会对这个应用编写自动化测试用例。

编写你的第一个用例

运用前面学到的 Desired Caps 以及元素定位方法，我们来编写一个添加待办事项的用例：

序号	执行步骤	预期结果
1	打开应用	
2	输入“使用 Appium 编写测试脚本”	
3	点击“添加”	添加成功

首先，我们在 Eclipse 里面新建一个带有 `main` 方法的类（勾选“`public static void main(String[] args)`”），类名为 `ToDoListTest`。如无意外，建立后的文件内容应该如下：

```
public class ToDoListTest {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
    }  
  
}
```

接着，动手时间到！大家运用前面学到的知识来编写自己的第一条用例吧！如果有问题，可以随时咨询你的教练。



好了，你应该写完自己的第一条用例了。下面是检查时间，给你的教练展示一下你跑起来的脚本吧！

为了方便后续描述，这里给出一个可运行的版本。注意，这不是唯一的编写方法，只要你的脚本能够跑起来，那么都是没问题的~

```
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.List;

import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.DesiredCapabilities;

public class ToDoListTest {

    public static void main(String[] args) throws MalformedURLException {

        // Install and open application
        File classpathRoot = new File(System.getProperty("user.dir"));
        File appDir = new File(classpathRoot, "app/");
        File app = new File(appDir, "ToDoList.apk");

        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability("deviceName", "Android Device");
        capabilities.setCapability("platformVersion", "4.4");
        capabilities.setCapability("app", app.getAbsolutePath());
        capabilities.setCapability("unicodeKeyboard", true);
        capabilities.setCapability("resetKeyboard", true);
        AndroidDriver driver = new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);

        // Add new item
        String itemText="使用 Appium 编写测试脚本";
        WebElement editText = driver.findElement(By.id("com.testershome.appiumgirl.todolist.editText"));
        editText.sendKeys(itemText);
        WebElement addItemBtn = driver.findElement(By.id("com.testershome.appiumgirl.todolist.addItemBtn"));
        addItemBtn.click();

        // Check if item is added
        List<AndroidElement> appiumItems = driver.findElements(By.xpath("//android.widget.TextView"));
        if (appiumItems.isEmpty()) {
            System.out.println("测试失败");
        }else{
            System.out.println("测试通过");
        }

        // exist
        driver.quit();
    }
}
```

用 Junit 组织你的用例

如果你已经在上面的脚本用上 Junit，请直接略过这一节

好了，我们的脚本编写好了，但总觉得缺了点什么？对的，我们缺少了报告！如果每一个测试用例都是通过 print 输出测试结果，那么当我们有100个用例的时候，岂不是看得眼花缭乱？不用急，Junit 来搭救你了！

Junit 是采用 Java 语言编写的一个单元测试框架。通过它，我们可以有效地组织我们的用例，把用例的不同部分区分开。

正常情况下，我们的测试用例总会有前提条件。它不属于测试范围，若无法创造此条件则测试用例无法进行，测试结果为 blocked。同样，自动化测试里面也有类似的概念，只是名字换成了 setUp 和 tearDown。其中 setUp 负责准备前提条件，它会在每个用例执行前被执行。tearDown 负责收尾，它会在每个用例执行后执行。值得注意的是，tearDown 无论在用例执行结果是什么的时候都会被执行。

现在，我们来重新整理一下我们前面的用例。它应该有两部分：

- 前提条件

ToDoList 应用已经装上手机并启动

- 执行步骤及预期结果

序号	步骤	预期结果
1	输入“使用 Appium 编写测试脚本”	
2	点击“添加”	添加成功

现在，我们用 Junit 改写我们的测试用例。

第一步，增加 setUp 方法，把 driver 的初始化放入其中：


```

...
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
...

public class ToDoListTest{
    private AndroidDriver driver;

    @Before
    public void setUp() throws Exception {
        // Install and open application
        File classpathRoot = new File(System.getProperty("user.dir"));
        File appDir = new File(classpathRoot, "app/");
        File app = new File(appDir, "ToDoList.apk");
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability("deviceName", "Android Device");
        capabilities.setCapability("platformVersion", "4.4");
        capabilities.setCapability("app", app.getAbsolutePath());
        capabilities.setCapability("unicodeKeyboard", true);
        capabilities.setCapability("resetKeyboard", true);
        driver = new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);
    }
    ...
}

```

第二步，把执行步骤及预期结果放到测试用例中，方法命名为 `addItem`：

```

...
@Test
public void addItem(){
    String itemText = "使用 Appium 编写测试脚本";

    // Add new item
    WebElement editText = driver.findElement(By.id("com.testahome.appiumgirl.todolist.editText"));
    editText.sendKeys(itemText);
    WebElement addItemBtn = driver.findElement(By.id("com.testahome.appiumgirl.todolist.addItemBtn"));
    addItemBtn.click();

    // Check if item is added
    List<AndroidElement> appiumItems = driver.findElements(By.xpath("//android.widget.TextView"));
    Assert.assertEquals("找不到待办事项 " + itemText, false, appiumItems.isEmpty());
}

```

第三步，把我们最后的关闭 session 操作放在 `tearDown`，防止后续的用例由于会话冲突无法启动：

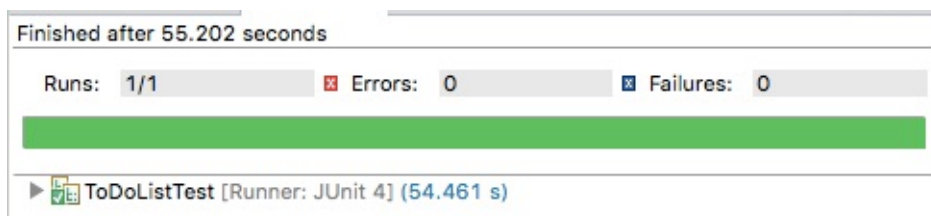
```
...
    driver = new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"), capabilities)
}

@After
public void tearDown() throws Exception {
    driver.quit();
}

@Test
public void addItem(){
    // Add new item
    ...
}
```

第四步，删除我们之前的 main 方法。现在我们已经不需要它了。

改造完成！现在，在 Eclipse 里面再运行一下这个用例，执行完毕后你会看到这样的结果：



是不是比之前好看多了！

隐式等待

由于我们的被测应用是纯本地操作，逻辑也比较简单，因此速度很快，添加后立即就出现了。但实际项目中大多数应用由于逻辑复杂、网络不稳定的因素，添加后会需要等待一段时间才能显示。此时，我们需要加入隐式等待。

隐式等待是指在所有查找元素方法中加入固定的等待时间。例如上面用例中我们只会查找一次添加后的待办事项“使用 Appium 编写测试脚本”，找不到元素就会直接执行失败。而加入隐式等待后，查找元素将会指定的等待时间中不断寻找，直到找到元素或者超时。

但需要注意，隐式等待一旦加入，直到修改隐式等待时间或 driver 退出，否则隐式等待将一直生效。

多说无用，Let's show code！

首先，我们添加一个新的用例。在这个用例中我们添加的事项内容改为“模拟弱网”。此时应用将会模拟弱网络下的行为，在点击添加按钮5秒后才出现待办事项：

```

@Test
public void addItemInWeekNetwork(){
    String itemText = "模拟弱网";

    // Add new item
    WebElement editText = driver.findElement(By.id("com.testershome.appiumgirl.todolis
    editText.sendKeys(itemText);
    WebElement addItemBtn = driver.findElement(By.id("com.testershome.appiumgirl.todol
    addItemBtn.click();

    // Check if item is added
    List<AndroidElement> appiumItems = driver.findElementsByXPath("//android.widget.T
    Assert.assertEquals("找不到待办事项 '"+itemText+"'", false, appiumItems.isEmpty());
}

```

此时你会发现，新的用例将会失败。此时，我们可以通过添加隐式等待来解决这个问题。设置隐式等待的方法是：`implicitlyWait()`。

同时，由于 `findElementsByXPath` 方法即使找不到元素也会立即返回，因此我们需要把它改为使用 `findElementByXPath`

```

@Test
public void addItemInWeekNetwork(){
    String itemText = "模拟弱网";

    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS); //--> Add implicit wait

    try{
        // Add new item
        ...
        // Check if item is added
        try{
            driver.findElementByXPath("//android.widget.TextView[@text='"+itemText+"'");
        }catch (Exception e){
            throw new AssertionError("找不到待办事项 '"+itemText+"");
        }
    }finally{
        driver.manage().timeouts().implicitlyWait(0, TimeUnit.SECONDS); //--> Remove implicit wait
    }
}

```

再次运行，你会发现脚本会自动等待直到出现“模拟弱网”这个待办事项啦！

试一试

试一下，把 `implicitlyWait` 的时间缩短到4秒，会发生什么？20秒呢？

拓展

除了隐式等待，其实对应还有显式等待。有兴趣的同学可以了解下：<http://www.cnblogs.com/shinhwa/p/3688184.html>

手势操作

通过上一章节的学习，我们已经掌握了编写脚本的基本技巧，接下来我们要通过本章节的学习做一个有意思的手势解锁，如果时间充足，可以尝试针对地图类APP来实践一下本节课内容。 预备知识：手机屏幕二维坐标系

- 你平时都使用过什么手势？
 - appium中提供的手势
 - 自定义手势
 - 一个手势解锁的demo
 - 总结
 - 相关资料
-

你平时都使用过什么手势？

聊起这个话题，相信你们在日常使用手机中都是有使用过如下的手势

上下左右滑屏（慢速）
快速滑动
滚动（滚动和滑动有什么区别？）
放大缩某个控件

相信大多数时间我们所使用的手势都是滑动，因为大多数APP基本都有滑动的手势，比如安装好的APP出现引导页，列表内容单页不够显示，而滑动是一个比较友好的方式。这样我们在自动化APP的时候就会遇到很多种滑动的情况，接下来让我们深入的了解一下这些手势。

appium中提供的手势

Java client 默认提供的API

- swipe --- 滑动
- zoom --- 放大
- pinch --- 缩小
- tap --- 点点（请思考tap和click的区别？）
- scroll ---- 滚动

swipe方法详解

滑动和滚动的区别你想清楚了吗？首先让我们了解一下appium 自带的滑动方法swipe, swipe方法的定义如下：

```
public void swipe(int startx,
                  int starty,
                  int endx,
                  int endy,
                  int duration)
```

swipe方法共有五个参数，其中参数依次代表起始点x、y坐标，终点x、y坐标和滑动时间，单位毫秒。如果想滑的快，就把时间设置小，反之。

swipe方法的本质是封装TouchAction,通过touchAction先按下起始点坐标，再间隔时间之后移动到终点坐标并释放。

具体实现方法如下：

```
TouchAction touchAction = new TouchAction(this);
// appium converts press-wait-moveto-release to a swipe action
touchAction.press(startx, starty).waitAction(duration)
              .moveTo(endx, endy).release();
touchAction.perform();
```

提示：swipe 的高级用法,swipe 可以进一步封装，比如一些MobileElement需要滑动屏幕才能出现，这个时候就可以封装一个滚动到这些MobileElement出现为止的方法。

zoom方法详解

通过了解swipe方法实际上zoom方法也类似 首先看一下该方法的定义：

```
void zoom(WebElement el);

void zoom(int x, int y);
```

zoom方法有两个，一个是传入一个WebElement，另外一个传入起始点坐标

zoom(x,y) 方法其实是由两个 MultiTouchAction实现的 首先根据你传入的坐标点确定偏移量，然后创建两个Action分别相反方向移动，同时时间释放。

```

MultiTouchAction multiTouch = new MultiTouchAction(this);
int scrHeight = manage().window().getSize().getHeight();
int yOffset = 100;

if (y - 100 < 0) {
    yOffset = y;
} else if (y + 100 > scrHeight) {
    yOffset = scrHeight - y;
}

TouchAction action0 = new TouchAction(this).press(x, y).moveTo(x, y - yOffset).release();
TouchAction action1 = new TouchAction(this).press(x, y).moveTo(x, y + yOffset).release();

multiTouch.add(action0).add(action1);

multiTouch.perform();

```

另外一个zoom方法也类似，可自己研究一下。

pinch方法详解

pinch 方法做的事情个zoom刚好相反但使用方法是一样的

```

void pinch(WebElement el);
void pinch(int x, int y);

```

tap方法详解

之前有提到过一个问题，tap和click的行为到底有什么不同？对于tap单一finger操作除了和click是通用的，但如不止一个finger那就不同咯。

先看一下tap方法

```

void tap(int fingers, int x, int y, int duration);
void tap(int fingers, WebElement element, int duration);

```

第一个的使用方法传入finger和要tap的坐标点还有间隔时间

第二个的使用方法是传finger和要tap的对象还有间隔时间

tap方法的实现代码如下：

```
MultiTouchAction multiTouch = new MultiTouchAction(this);

for (int i = 0; i < fingers; i++) {
    multiTouch.add(createTap(x, y, duration));
}

multiTouch.perform();
```

scroll方法详解

scroll是一个抽象的方法，iOS和Android各自实现

```
scrollTo(String text)
scrollToExact(String text)
```

那么问题就来了，这两者有什么样的区别？

首先两者传入的都是字符串，而这个字符串是对所要滚动到的对象的描述，不同之处在于scrollTo是包含contains，而scrollToExact>equals，精确匹配。

提示：scroll可以滚动到查找某个MobileElement出现，和swipe不同的是这个滚动只能是当前屏幕，不能跨页面

自定义手势

注意在早期的Appium版本中,可使用mobile command 自定义一些手势，比如swipe,flick 但是在最近的Appium中已经不再支持这些mobile command

```
Tried to execute non-existent mobile command 'swipe'. Most mobile commands have
been ported to official client library methods. Please check your Appium library for more
information and documentation
```

如果你在网上看到类似这样的代码,切记已经完全无效：


```
JavascriptExecutor js = (JavascriptExecutor) driver;
HashMap<String, String> swipeObject = new HashMap<String, String>();
swipeObject.put("startX", "100");
swipeObject.put("startY", "400");
swipeObject.put("endX", "100");
swipeObject.put("endY", "200");
swipeObject.put("duration", "400");
js.executeScript("mobile: swipe", swipeObject);
```

而这里我讲的自定义手势，比如向上向下滑动，是根据swipe的原理设计出来的

还可以进一步封装滑动到某MobileElement 出现 举个例子,向上滑动屏幕：

```
/**
 * This Method for swipe up
 *
 * @author Young
 * @param driver
 * @param during
 */
public void swipeToUp(AndroidDriver<MobileElement> driver, int during)
{
    int width = driver.manage().window().getSize().width;
    int height = driver.manage().window().getSize().height;
    driver.swipe(width / 2, height * 3 / 4, width / 2, height / 4, during);
}
```

首先获取你设备的宽度和高度，然后根据二维坐标系进行滑动，同理你可以试着封装一个手势。

一个手势解锁的demo

核心代码如下：

```
@Test
public void GGestureLockerTest() throws InterruptedException
{
    driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);

    MobileElement button = driver.findElementByAndroidUIAutomator("new UiSelector().t
button.tap(1, 1000);
    // get all the items of gesture locker
    List<MobileElement> items = driver.findElementsByClassName("android.widget.ImageView

    for (MobileElement item : items)
    {
        /**
        * 0 1 2 3 4 5 6 7 8
        */
        item.click();
    }

    // create a Z from 0->1->2->4->6->7->8
    TouchAction touches = new TouchAction(driver);
    touches.press(items.get(0)).waitAction(1000).moveTo(items.get(1)).waitAction(1000
        .waitAction(1000).moveTo(items.get(4)).moveTo(items.get(6)).waitAction(10
        .waitAction(1000).moveTo(items.get(8)).release();
    touches.perform();
    Thread.sleep(1000);
    touches.press(items.get(0)).waitAction(1000).moveTo(items.get(1)).waitAction(1000
        .waitAction(1000).moveTo(items.get(4)).release();
    touches.perform();
    Assert.assertTrue(driver.findElementByName("与上一次绘制不一致, 请重新绘制").isDisplay

}
```

总结

本章我们都学习了常见手势的使用方法，其中appium自带的

- swipe
- tap
- zoom
- pinch
- scroll

掌握了这些，可以针对一个地图类型的app实践一下

```
@Test
public void GustomerLockerTest() throws InterruptedException
{
    driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
    int width = driver.manage().window().getSize().width;
    int height = driver.manage().window().getSize().height;
    Thread.sleep(15000);

    // swipe to right
    driver.swipe(width / 4, height / 2, width * 3 / 4, height / 2, 300);
    // swipe to left
    driver.swipe(width * 3 / 4, height / 2, width / 4, height / 2, 300);

    Thread.sleep(5000);
    driver.pinch(width / 4, height / 4);
    Thread.sleep(5000);
    driver.zoom(width / 4, height / 4);
    Thread.sleep(5000);
    // tap
    driver.tap(2, width / 2, height / 2, 1000);
}
```

该app的下载地址：

<https://github.com/tobecrazy/LuoHe/blob/master/app/build/outputs/apk/app-debug.apk>

相关资料

demo所使用的APK

<https://github.com/tobecrazy/appiumDemo/blob/master/apps/Locker.apk>

官方资料：<http://appium.io/slate/en/master/?java#key-event>

进阶之UIAutomator

来到这里了，说明你已经学会了以下几个项目：

- Appium Server的启动
- App元素定位
- 脚本编写和执行

OK，我们接下来学习一下关于 `UIAutomator` 在Appium中的应用。

我们都知道，Appium的默认运行模式就是基于 `UIAutomator` 的，熟悉 `UIAutomator` 的同学们应该知道，本身就是一个测试框架，它有非常多实用的特性，那在我们的实际应用当中，我们需要了解和学习它的哪些东西呢？

UIAutomator

这里不具体展开 `UIAutomator` 的框架本身的内容。建议大家先去[testerhome-UIAutomator板块](#)了解学习一下。

其中主要有几个对象需要重点学习：

- `UIDevice`
- `UISelector`
- `UIScrollable`

本文主要针对两个业务场景来为大家介绍UIAutomator的使用实践。

核心讲解的是在 `AndroidDriver` 中，`findElementByAndroidUIAutomator` 这个方法的使用。

第一式 多属性联合查询定位

当我们遇到一个元素，它没有唯一的ID、Text、ClassName等明显标记可以唯一确定它的时候，往往需要联合该元素的多个属性来唯一确定它的位置。

常见场景：假设页面A为导航页，该页面全部的元素都是由Imageview组成，包括上面唯一可点击的按钮 `跳过`，都是一个Imageview空控件，这些IV控件没有任何ID、Text标记，那我们怎么点击到 `跳过` 这个IV控件呢？

有同学可能会说，获取这个页面的所有IV控件，然后通过序号的方式去点击。

ok，这是一种解决方案。但是并不完美。

我们打开 `uiautomatorviewer` 来分析一下这个场景，布局上全都是 `ImageView` 控件，没有ID，没有Text，没有Content-Desc。但是我们能够从上面发现，`跳过` 这个控件有另外一个特点，它的 `clickable` 属性是True，其他的IV却都是False。这说明它是这个页面中所有 `ImageView` 控件中唯一一个可点击的。

ok，这就是我们的切入点。我们现在也明确，我们要点击的元素的明确特征是：

- 1.是ImageView (`className='Android.widget.ImageView'`)
- 2.是可点击的 (`clickable=true`)

那么我们怎么将这两个属性组合起来查询？

我们可以直接使用UIAutomator的方式：

```
//UiAutomator原生的定位方式
UiObject iv = new UiObject(new UiSelector().className("android.widget.ImageView").clickable(true));
iv.click();
```

那在Appium里的实现就是：

```
WebElement iv = driver.findElementByAndroidUIAutomator("new UiSelector().className(\"android.widget.ImageView\").clickable(true)");
iv.click();
```

这里需要特别说明的是，`findElementByAndroidUIAutomator` 方法获取的对象就是 `UiObject` 本身，所以是写法如上。

第二式 ListView自动搜索查询

当我们碰到很长的ListView，且需要在这个ListView里面查询指定的元素的时候。我们如何做呢？生硬的swipe+findElement适用吗？

场景：在班级列表中找到带有 `上课中` 字样的选项，然后点击。

常见的 `swipe` 滑动List，然后 `findElement` 找到指定的元素，这个方式也是可用的，但是实在不稳定，因为我们不能确定到底滑动多少次，才进行元素点击，也不知道什么时候才滑动到了最后。所以这个方法是不可行的。

正确的方法：

```
//UiAutomator原生
```

```
//此方法的含义是先获取当前页面可滑动的元素，然后在这个元素的基础上，找到包含`上课中`这三个字的项目，再点击  
UiObject c1 = new UiScrollable(new UiSelector().scrollable(true)).scrollIntoView(new UiSe  
c1.clickt()
```

那么，在Appium中的写法就是：

```
WebElement c1 = driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()  
c1.click()
```

快去尝试一下吧。

相关资料：<http://developer.android.com/intl/zh-cn/tools/testing-support-library/index.html>

进阶之Hybrid

Hybrid应用，简而言之就是指Native应用里内嵌了WebView的应用。此类应用从14年开始普遍流行，从技术发展和业务契合度的角度来看，Hybrid带来了许多好处和变化，比如使App本身更加轻便，使许多热更技术变得可行等等。但这也给我们做App的自动化带了许多难度和瓶颈。

在Appium出现之前，能实现Hybrid自动化的框架并不多。而 `Selendroid` 就是其中一个。

而Appium所应用到的技术，其中一项也正是包含了 `Selendroid` 模式。

Appium下的Hybrid解决方案

- 1、基于 `Selendroid` 模式
- 2、基于 `UiAutomator` + `Chromedriver` 模式

这两种模式各有优劣：

Selendroid模式

- 支持Android2.3+
- 支持Hybrid自动化
- 运行速度快
- 无法使用许多手势API
- apk包需要重签名
- 无法跨进程

Appium模式

- 支持Android4.4以上设备
- 支持Hybrid自动化
- 可跨进程
- apk包无需重签名
- 源码中webview必须为debug模式（真机）

很明显，我们从以上两个分析可以看出，Appium模式其实是更加完善，对于运行稳定性和拓展性优势是要比较明显的。目前的Android市场份额，已经逐渐向高端转化，不出半年，也会变成4.4~6.0的天下了。所以，这个模式正在从劣势变为优势。

但是上面提到的webview必须为debug模式是什么意思呢？

基于UIAutomator+Chromedriver的实践

使用 `chromedriver` 来做hybrid的自动化我们有几个前提条件必须解决：

- 1、准备好Android4.4或以上的手机；
- 2、将Webview设置为debug模式；
 - 设置方法：在Android SDK API>=19的情况下，在源码中添加 `webview.setWebContentsDebuggingEnabled(true)` 这一段代码即可。（如果使用的是模拟器，则无需修改源码）

Android混合应用自动化的关键API

实际上，Appium模式下，实现混合应用的自动化的原理很简单，Native部分走UIAutomator，Webview部分走Chromedriver，两者结合混搭，从而实现Hybrid的自动化。

那我们到底如何做到在 `UIAutomator` 和 `Chromedriver` 之间灵活地游走呢？

除了准备工作要做好，还有几个关键的概念你需要清楚：

- `context`
- `window_handle`(一般很少用到)

我们通过切换当前 `context` 对象，来让 Appium 认识自己当前处于哪一个状态里面，对于 `webview` 的 `context` 对象，可能会打开多个网页，那我们还需要通过切换 `window_handle` 对象来让 `Chromedriver` 认识自己当前是打开的哪个页面。

那么，刚刚提到的两个概念，就是我们学习的重点。

解读Sample-Code

Talk is very cheap。相信看完上面的东西，许多同学还是一头雾水。还是从看源码解释要来得清楚一点。

我们从[sample-code-androidWebViewTest.java](#)可以看到：


```

public class AndroidWebViewTest {
    private AppiumDriver<WebElement> driver;

    @Before
    public void setUp() throws Exception {
        // set up appium
        File classpathRoot = new File(System.getProperty("user.dir"));
        File app = new File(classpathRoot, "../../../apps/selendroid-test-app.apk");
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability("deviceName", "Android Emulator");

        //automationName必须为Appium, 或者该参数不填
        capabilities.setCapability("automationName", "Appium");
        capabilities.setCapability("app", app.getAbsolutePath());
        capabilities.setCapability("appPackage", "io.selendroid.testapp");
        capabilities.setCapability("appActivity", ".HomeScreenActivity");
        driver = new AndroidDriver<WebElement>(new URL("http://127.0.0.1:4723/wd/hub"), c
    }

    @After
    public void tearDown() throws Exception {
        driver.quit();
    }

    @Test
    public void webView() throws InterruptedException {
        WebElement button = driver.findElement(By.id("buttonStartWebview"));
        button.click();
        Thread.sleep(6000);

        //获取当前页面的所有Context对象, 其中就会包含Native和Webview的对象。
        Set<String> contextNames = driver.getContextHandles();
        for (String contextName : contextNames) {
            System.out.println(contextName);

            //遍历获取下来的context对象, 若发现context对象包含WEBVIEW字样时, 就切换到该对象下
            if (contextName.contains("WEBVIEW")){
                driver.context(contextName);
            }
        }
        //然后就是webview中的操作, webview下的自动化定位方式与Selenium中web的定位是一样的。
        WebElement inputField = driver.findElement(By.id("name_input"));
        inputField.sendKeys("Some name");
        inputField.submit();
    }
}

```

我在上面标注了4个关键的注释, 请认真阅读。

其中，相比于单纯的Native测试，Hybrid测试会多两道工序，那就是：

- 1、获取当前contexts对象
- 2、切换到webview的context对象下

接下来后续的操作都是跟Native下相差无几。

注意点

- 当我们在Webview下操作完毕了以后，若想要操作Native下的元素，则需要重新切换context对象到Native下，就跟你切换到webview下是一样的。

App中WebView元素的定位方式

有同学会问，Native的所有元素都可以通过uiautomatorviewer获取到，那webview里的元素我们有办法或者有工具能够帮助我们定位吗？

答案肯定是有。

那就是chrome浏览器的 `inspector`。

使用方法：

- 1、手机连接上电脑，并打开App，打开需要定位的Webview的页面
- 2、电脑上打开Chrome，地址栏输入 `chrome://inspect`
- 3、点击devices标签，此时你会看到你设备上对应的App的包名
- 4、点击包名旁边的 `inspect`，就会进入chrome的调试工具，在这个调试工具就可以获取当前webview的所有元素了。

框架开发

附录 - 下载篇

Eclipse (IDE)

ADT版本_v23.0:

- [Windows 32位](#)
- [Windows 64位](#)
- [MacOSX 64位](#)

JDK

Windows :

- [JDK 1.7 u60 32位](#)
- [JDK 1.7 u60 64位](#)

MacOSX :

- [JDK 1.7 u60 64位](#)

Android SDK

SDK Tools :

- [SDK tools for Windows](#)
- [SDK tools for MacOSX](#)

SDK Platform Tools :

这是 adb, fastboot 等工具包。把解压出来的 platform-tools 文件夹放在 android sdk 根目录下，并把 adb所在的目录添加到系统 PATH 路径里，即可在命令行里直接访问了 adb, fastboot 等工具。

- [Platform Tools For Windows](#)
- [Platform Tools For MacOSX](#)

Build Tools:

这是Android开发所需的Build-Tools，下载并解压后，将解压出的整个文件夹复制或者移动到 your sdk 路径/build-tools 文件夹即可。

- [Build Tools For Windows](#)
- [Build Tools For MacOSX](#)

Android SDK :

- [Android 4.4W For Windows](#)
- [Android 5.0 For Windows](#)
- [Android 4.4W For MacOSX](#)
- [Android 5.0 For MacOSX](#)

Appium客户端

- [Appium For Windows](#)
- [Appium.dmg For MacOSX](#)

Appium Java Client

- [Java-Client-3.3.0.jar](#)
- [selenium-server-standalone-2.49.0.jar](#)

被测应用

- [ToDoList](#)
- [Hybrid](#)
- [for滑动解锁demo](#)
- [for手势操作的demo](#)