

Table of Contents

UI自动化测试课程	1.1
移动自动化测试基础	1.2
移动自动化测试框架	1.2.1
环境搭建	1.2.2
ADB调试工具	1.2.3
UIAutomatorViewer工具	1.2.4
入门示例	1.2.5
Appium基础操作	1.2.6
元素定位	1.2.7
元素操作	1.2.8
滑动和拖拽事件	1.2.9
高级手势TouchAction	1.2.10
手机操作	1.2.11
扩展	1.2.12

UI自动化测试课程

序号	章节	知识点
1	UI自动化测试介绍	1. UI自动化测试
2	Web自动化测试基础	1. Web自动化测试框架 2. 环境搭建 3. 元素定位和元素操作 4. 鼠标和键盘操作 5. 元素等待 6. HTML特殊元素处理 7. 验证码处理
3	移动自动化测试基础	1. 移动自动化测试框架 2. ADB调试工具 3. UIAutomatorViewer工具 4. 元素定位和元素操作 5. 滑动和拖拽事件 6. 高级手势TouchAction 7. 手机操作
4	PyTest框架	1. PyTest基本使用 2. PyTest常用插件 3. PyTest高级用法
5	PO模式	1. 方法封装 2. PO模式介绍 3. PO模式实战
6	数据驱动	1. 数据驱动介绍 2. 数据驱动实战
7	日志收集	1. 日志相关概念 2. 日志的基本方法 3. 日志的高级方法
8	黑马头条项目实战	1. 自动化测试流程 2. 项目实战演练

课程目标

1. 掌握使用Selenium实现Web自动化测试的流程和方法，并且能够完成自动化测试脚本的编写。
2. 掌握使用Appium实现移动自动化测试的流程和方法，并且能够完成自动化测试脚本的编写。
3. 掌握如何通过PyTest管理用例脚本，并使用Allure生成HTML测试报告。
4. 掌握使用PO模式来设计自动化测试代码的架构。
5. 掌握使用数据驱动来实现自动化测试代码和测试数据的分离。
6. 掌握使用logging来实现日志的收集。

移动自动化测试基础

目标

1. 熟悉Appium的特点和工作原理
2. 熟悉Appium相关环境的搭建过程
3. 掌握常见的adb命令
4. 能够使用 UIAutomatorViewer 获取元素的特征信息
5. 掌握Appium基础操作
6. 掌握元素定位的方法
7. 掌握滑动和拖拽事件
8. 掌握高级手势TouchAction
9. 掌握常用的手机操作
10. 掌握如何获取toast内容
11. 能够使用 Monkey 对 app 进行压力测试

移动自动化测试框架

目标

1. 了解主流的移动自动化测试框架
2. 熟悉Appium的特点
3. 掌握Appium的工作原理

1. 主流的移动自动化测试框架

- Robotium
 - 是一款国外开源的Android自动化测试框架
 - 适用平台：Android
 - 支持语言：Java
 - 不支持跨应用
- macaca
 - 是由阿里巴巴公司开源的一套自动化解决方案
 - 适用平台：PC端、Android、iOS
 - 支持语言：Java、Python、Node.js
 - 支持跨应用
- Appium
 - 是一款国外开源的自动化测试工具
 - 适用平台：Android、iOS
 - 支持语言：Java、Javascript、PHP、Python、C#、Ruby等主流语言
 - 支持跨应用
 - 社区活跃、资料丰富

2. Appium介绍

2.1 特点

- 开源；
- 支持Native App、Web App、Hybird App；
- 支持Android、iOS；
- Server也是跨平台的，你可以使用Mac OS X、Windows或者Linux；
- 用Appium自动化测试不需要重新编译App；
- 支持很多语言来编写测试脚本，Java、Javascript、PHP、Python、C#、Ruby等主流语言；

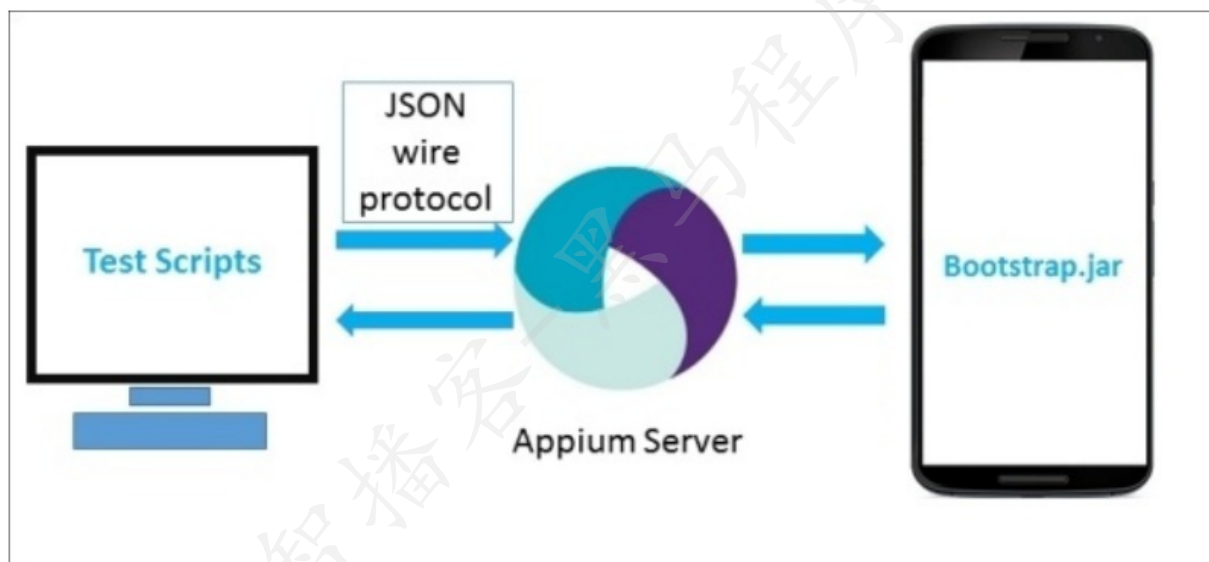
注：

- **Native App**: 原生应用，使用Android或iOS的标准SDK编写的应用；
- **Web App**: 移动浏览器应用，使用移动平台的浏览器访问的应用；
- **Hybrid App**: 混合应用，把一个基于webview实现的功能进行包装的应用。

2.2 设计理念

- C/S架构，appium的核心是一个web服务器，提供了一套接口。他会接收客户端发送过来的命令，然后在移动设备上运行命令，最后把运行结果通过HTTP响应包返回给客户端。
- **session**，每个client连接到server以后都会创建一个**session**，自动化始终围绕一个**session**进行。

2.3 工作流程



环境搭建

目标

1. 熟悉Appium相关环境的搭建过程

1. 安装JDK

1.1 版本

JDK1.8

1.2 下载地址

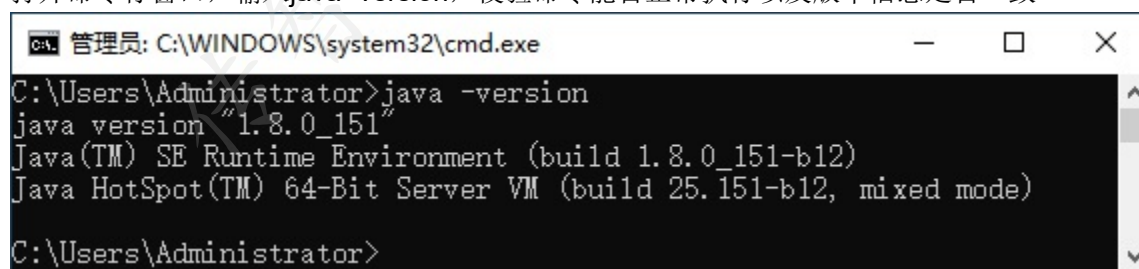
<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

1.3 配置环境变量

- JAVA_HOME=C:\Program Files\Java\jdk1.8.0_151
- 在Path中添加:%JAVA_HOME%\bin;

1.4 校验

打开命令行窗口，输入java -version，校验命令能否正常执行以及版本信息是否一致



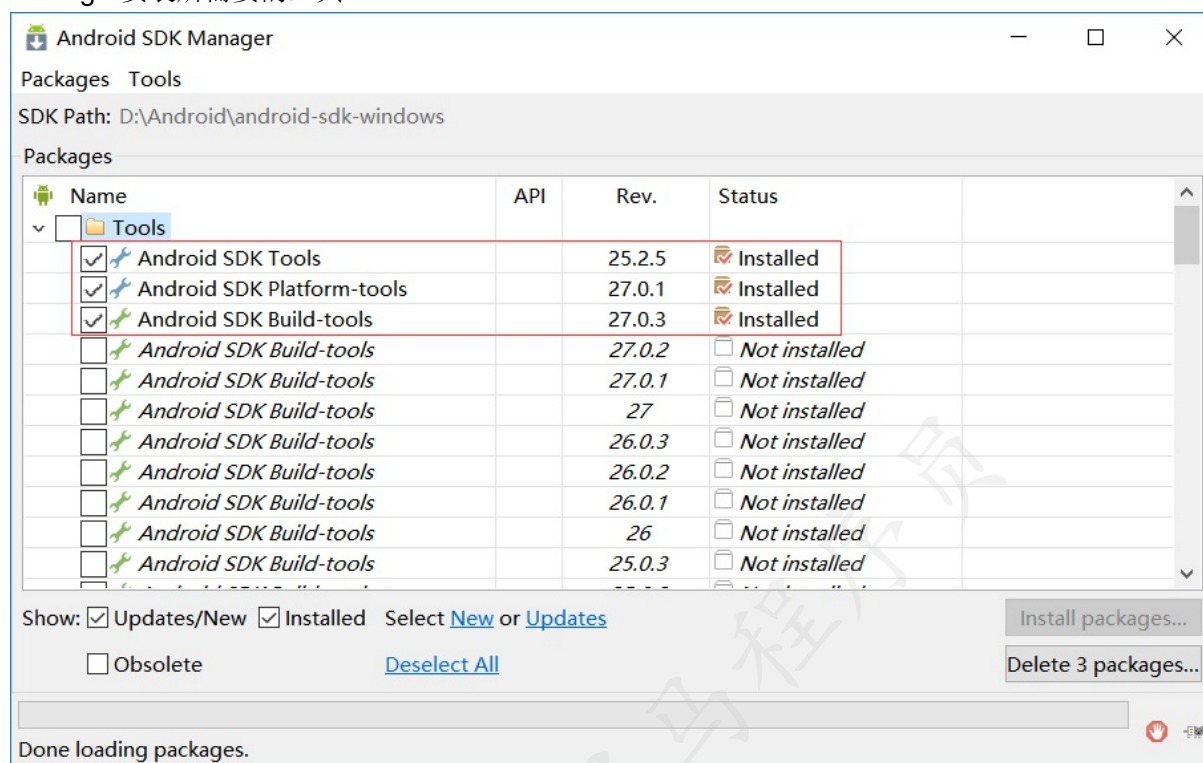
```
管理员: C:\WINDOWS\system32\cmd.exe
C:\Users\Administrator>java -version
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
C:\Users\Administrator>
```

2. 安装Android SDK

2.1 下载地址

http://dl.google.com/android/android-sdk_r24.4.1-windows.zip

解压之后的安装包只包含基本的SDK工具，它不包含Android平台或任何第三方库。需要使用SDK Manager安装所需要的工具。

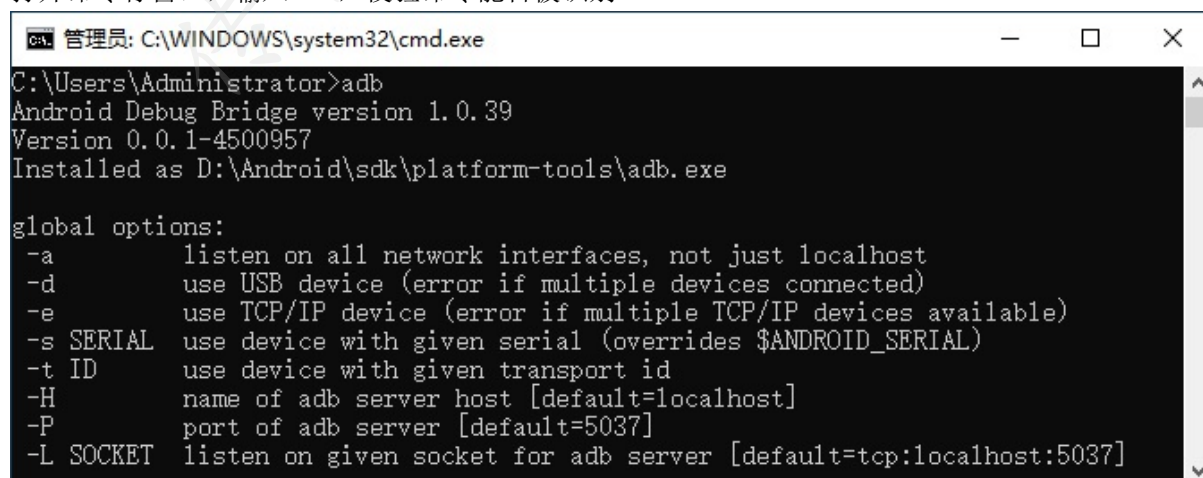


2.2 配置环境变量

- ANDROID_HOME=D:\Android\sdk
- 在Path中添加:%ANDROID_HOME%\tools;%ANDROID_HOME%\platform-tools;

2.3 校验

打开命令行窗口，输入adb，校验命令能否被识别



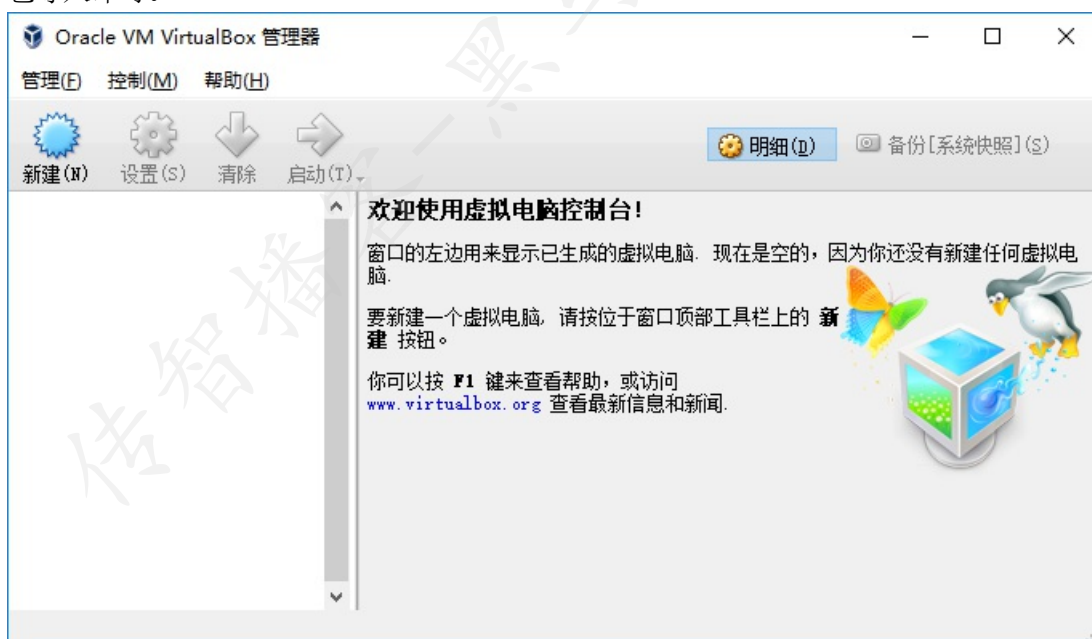
3. 安装Genymotion

3.1 安装步骤

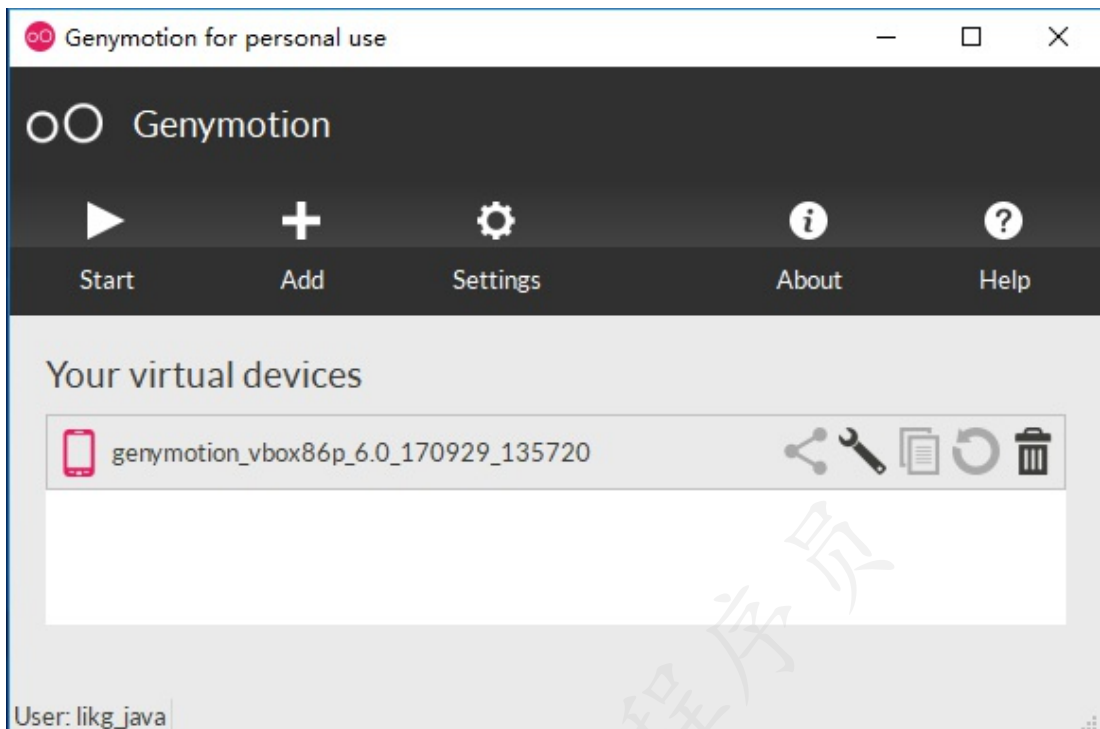
1. 打开Genymotion官网: <https://www.genymotion.com/>
2. 注册账号
3. 下载Genymotion安装包并安装, 下载地址: <https://www.genymotion.com/download/>
4. 下载VirtualBox安装包并安装, 下载地址: <http://download.virtualbox.org/virtualbox/5.1.30/VirtualBox-5.1.30-118389-Win.exe>

3.2 添加模拟器

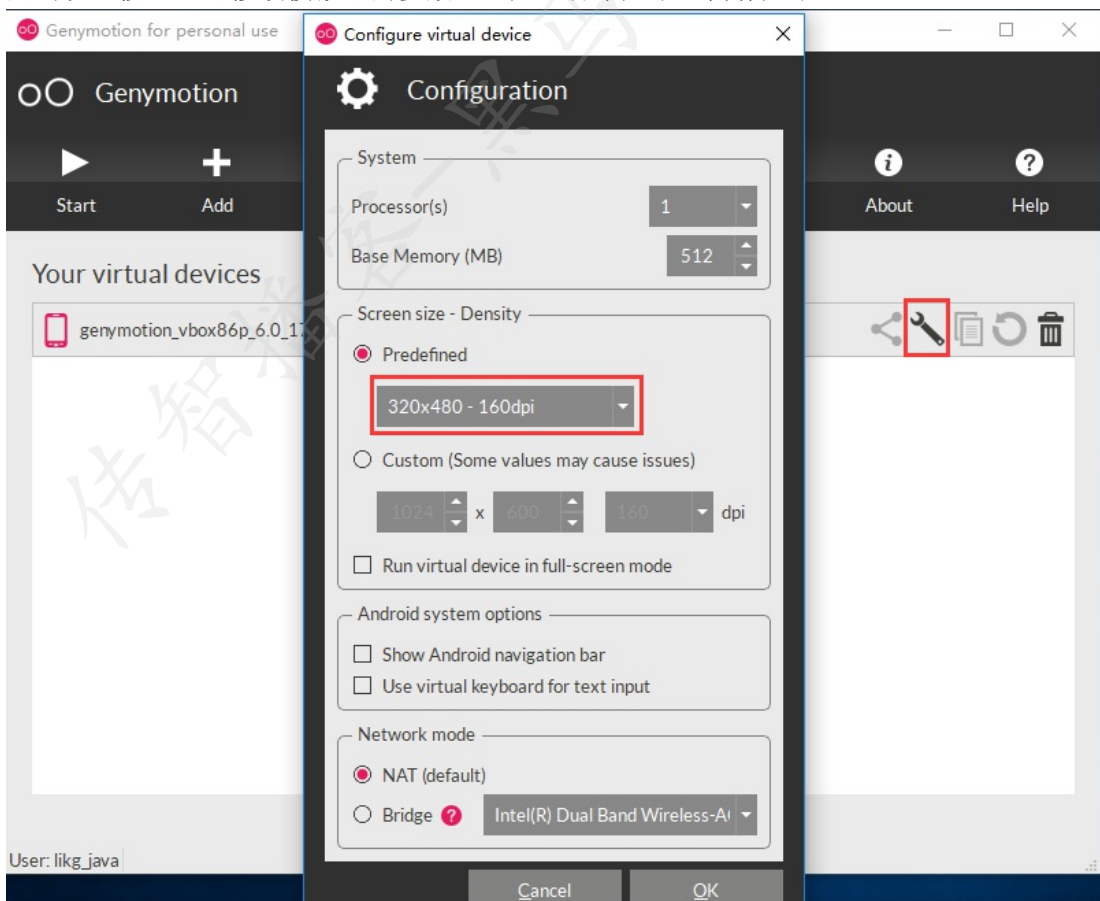
- 在线添加
 1. 打开Genymotion客户端, 点击添加按钮, 选择一款虚拟设备安装即可。
- 离线导入
 1. 下载模拟器镜像文件
(如: http://dl.genymotion.com/dists/6.0.0/ova/genymotion_vbox86p_6.0_170929_135720.ova)
 2. 打开Oracle VM VirtualBox客户端, 选择‘管理’->‘导入虚拟电脑...’菜单, 选择下载的镜像包导入即可。



3. 打开Genymotion客户端即可看到导入的模拟器设备。

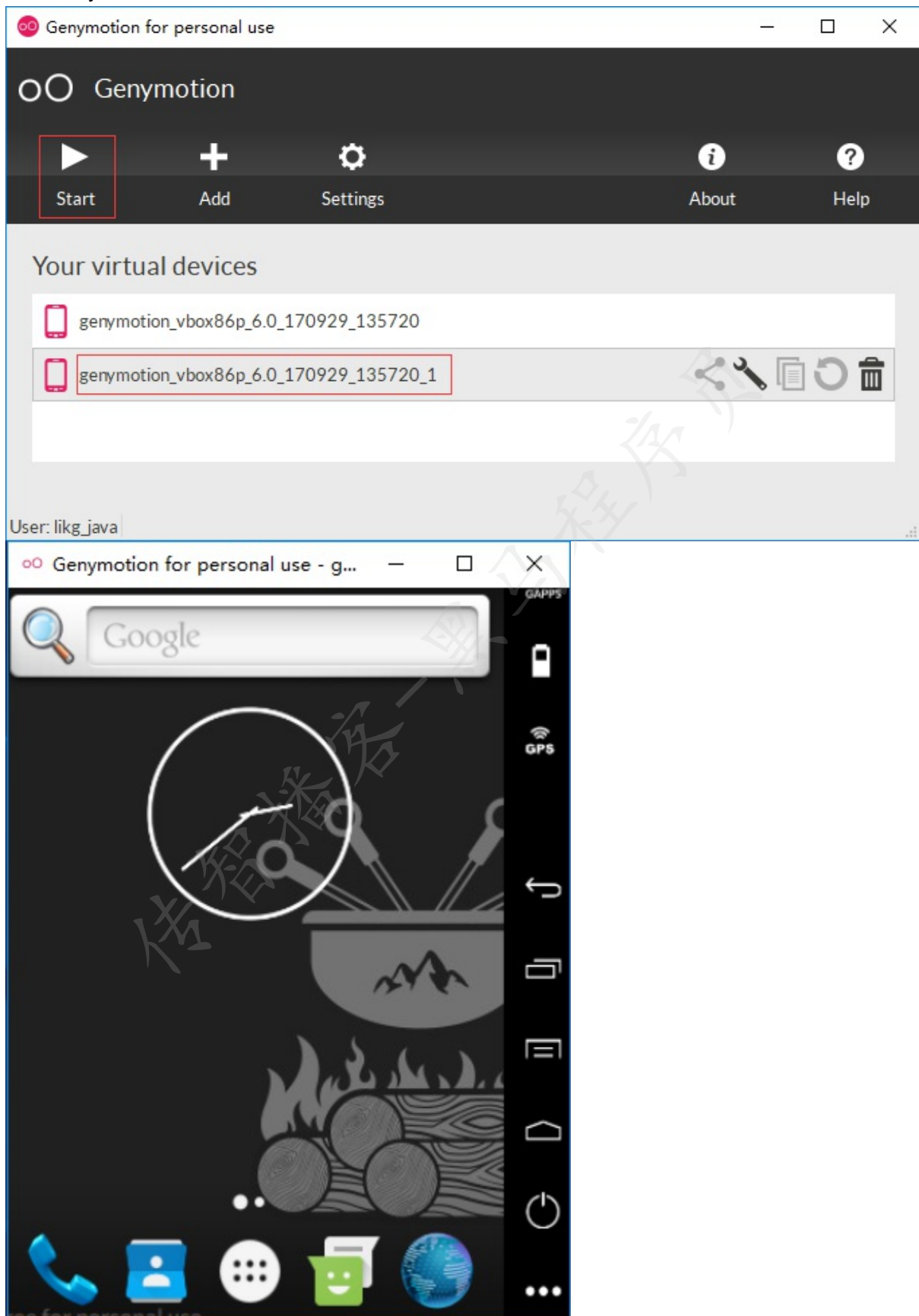


4. 点击设置按钮，可修改模拟器的参数，比如：屏幕大小、内存大小。



3.3 启动模拟器

打开Genymotion客户端，选中一款虚拟机设备，点击‘Start’按钮即可启动。



3.4 安装Genymotion ARM插件

Genymotion是基于X86的，不支持ARM架构，导致有些基于ARM架构编译的APP无法安装。

错误一：安装APP时会出现Failure [INSTALL_FAILED_NO_MATCHING_ABIS]异常

解决方案：下载Genymotion-ARM-Translation_v1.1.zip，直接拖入genymotion模拟器窗口，处理完成后点OK，再重启一下模拟器。

错误二：安装APP时会出现Genymotion unfortunately has stopped异常

解决方案：下载genymotion的arm兼容包ARM_Translation_Marshmallow.zip，直接拖入genymotion模拟器窗口，处理完成后点OK，再重启一下模拟器。

3.5 注意事项

1. 最好不要改变VirtualBox的安装路径。
2. enabled in BIOS settings，开机启动时设置virtualization technology的选项为Enabled。
3. Genymotion_Arm转换器的安装包，最好不要放在含有中文路径的目录下。

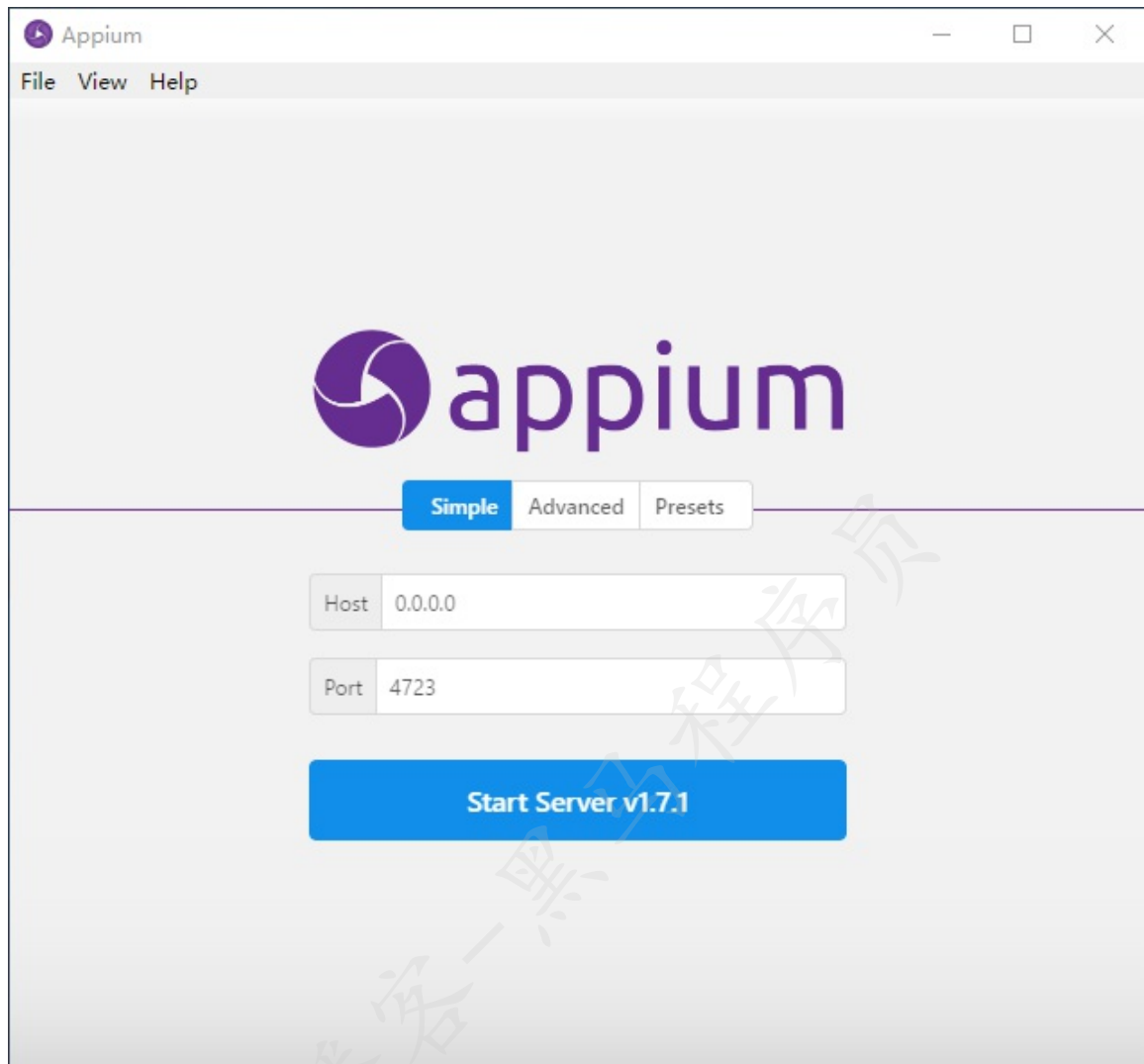
4. 安装appium

4.1 两种安装模式

- 客户端安装
- 终端安装

4.2 客户端安装

- 官网：<http://appium.io/>
- 下载地址：<https://github.com/appium/appium-desktop/releases>
- 双击启动



4.3 终端安装

1. 首先安装nodejs，下载地址：<https://nodejs.org/en/download/>
2. 三种安装方式
 - 直接在终端输入命令：`npm install -g appium` 一般不会成功，有防火墙
 - 通过国内镜像安装：`npm install -g appium -registry http://registry.cnpmjs.org`
 - 通过cnpm安装：

```
npm install -g cnpm --registry=https://registry.npm.taobao.org
cnpm install -g appium
```

3. 启动

```
appium
```

5. 安装Appium-python库

- 使用PIP工具安装

```
pip install Appium-Python-Client
```

佐智播客-黑马程序员

ADB调试工具

目标

1. 能够了解 adb 的工作原理
2. 能够应用常用的 adb 命令

1. adb 的工作原理

1.1 adb 的概念

ADB 全名 Android Debug Bridge，是一个调试工具。

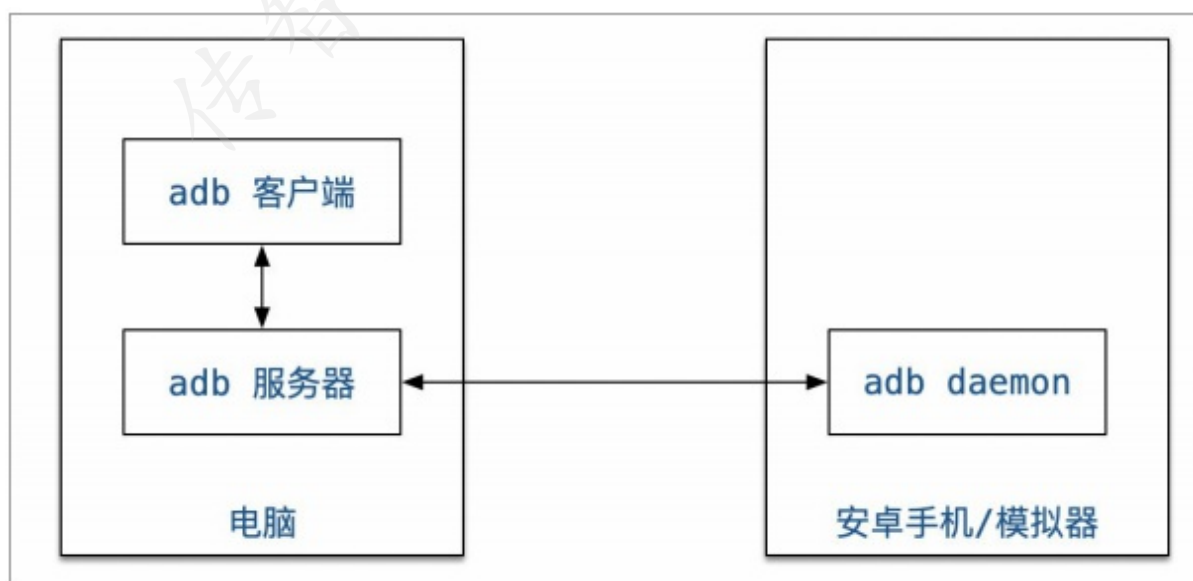
- 开发安卓应用的程序员必须要掌握
- 测试工程师在做安卓应用测试时，会使用到

1.2 adb 的构成和工作原理

提示：只需要知道组成部分和工作原理，会使用即可，面试可能会被问到

adb 包含三个部分：

- **Client**端：运行在开发机器中，即你的开发电脑，用来发送 adb 命令；
- **Daemon**守护进程：运行在调试设备中，手机或模拟器，用来接收并执行 adb 命令；
- **Server**端：同样运行在开发机器中，用来管理 Client 端和手机的 Daemon 之间的通信。



小结：adb工具可以在电脑通过终端命令操作安卓手机/模拟器。

2. adb 常用命令

2.1 获取包名和界面名【应用】

1. 包名和界面名的概念
2. 获取包名和界面名

2.1.1 包名和界面名的概念

1. 包名（**package**）：决定程序的唯一性（不是应用的名字）
2. 界面名（**activity**）：目前可以理解，一个界面名，对应着一个界面。

2.1.2 获取包名和界面名

应用场景

自动化测试需要通过代码的形式告诉手机测试哪个应用程序的哪一个界面，所以需要通过这个命令进行获取。

使用步骤

1. 打开需要测试的应用程序
2. 输入 adb 命令

命令格式

- Mac/Linux:

```
adb shell dumpsys window windows | grep mFocusedApp
```

- Windows:

```
adb shell dumpsys window windows | findstr mFocusedApp
```

示例

作用：获取设置程序的包名和界面名

1. 先在模拟器或手机中打开《设置》应用程序
2. 输入对应平台的命令

结果如下：

```
mFocusedApp=AppWindowToken{53309da token=Token{2e2fa785 ActivityRecord{2928d4fc u0 com.
android.settings/.Settings t1127}}}
```

其中：

- 包名为: `com.android.settings`
- 界面名为: `.Settings`

注意点

界面名可能会在和同事沟通交流或网站的文章中翻译为启动名

2.2 文件传输【应用】

1. 发送文件到手机
2. 从手机中拉取文件

2.2.1 发送文件到手机

应用场景

将手机需要的数据（数据库文件）在电脑上调整好，直接发送给手机

命令格式

```
adb push 电脑的文件路径 手机的文件夹路径
```

示例

作用：将桌面的 `a.txt` 发送到手机的 `sd` 卡

```
adb push C:\Users\hm\Desktop\a.txt /sdcard
```

2.2.2 从手机中拉取文件

应用场景

将手机产生的文件（数据库文件，日志文件）拉取到电脑中

命令格式

```
adb pull 手机的文件路径 电脑的文件夹路径
```

示例

作用：将手机的 `sd` 卡的 `a.txt` 拉取到桌面

```
adb pull /sdcard/a.txt C:\Users\hm\Desktop
```


2.3 获取 app 启动时间【应用】

应用场景

1. 如果企业对应用程序的启动速度有要求，则需要使用这个命令进行测试
2. 测试标准：参照同类软件，启动时间不能超出一倍即可

命令格式

```
adb shell am start -W 包名/启动名
```

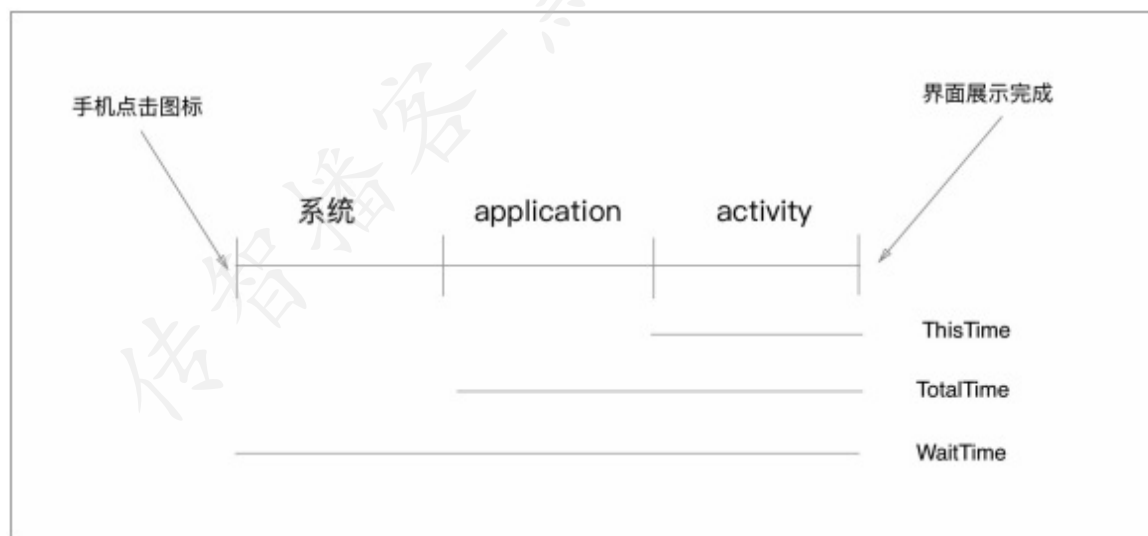
示例

作用：启动 `com.android.settings` 程序并且进入主界面 (`.Settings`)

```
adb shell am start -W com.android.settings/.Settings
```

解释

- **ThisTime**：该界面 (`activity`) 启动耗时 (毫秒)
- **TotalTime**：应用自身启动耗时 = **ThisTime** + 应用 `application` 等资源启动时间 (毫秒)
- **WaitTime**：系统启动应用耗时 = **TotalTime** + 系统资源启动时间 (毫秒)



2.4 获取手机日志【应用】

应用场景

将bug的日志信息发送给开发人员，便于开发人员定位bug

使用步骤

1. 打开需要测试的应用程序

2. 找到触发bug的位置
3. 使用查看日志命令
4. 触发bug
5. 获取日志信息

命令格式

```
adb logcat
```

示例

1. 安装bug.apk
2. 打开《有bug的程序》应用程序
3. 命令行中输入 adb logcat 命令
4. 点击登录按钮
5. 获取日志信息

结果如下：

```
E/AndroidRuntime( 6593): FATAL EXCEPTION: main
E/AndroidRuntime( 6593): Process: cn.itcast.myapplication, PID: 6593
E/AndroidRuntime( 6593): java.lang.ArrayIndexOutOfBoundsException: length=5; index=8
E/AndroidRuntime( 6593):     at cn.itcast.myapplication.LoginActivity.attemptLogin(LoginActivity.java:149)
E/AndroidRuntime( 6593):     at cn.itcast.myapplication.LoginActivity.access$000(LoginActivity.java:40)
E/AndroidRuntime( 6593):     at cn.itcast.myapplication.LoginActivity$2.onClick(LoginActivity.java:89)
E/AndroidRuntime( 6593):     at android.view.View.performClick(View.java:4780)
E/AndroidRuntime( 6593):     at android.view.View$PerformClick.run(View.java:19866)
E/AndroidRuntime( 6593):     at android.os.Handler.handleCallback(Handler.java:739)
E/AndroidRuntime( 6593):     at android.os.Handler.dispatchMessage(Handler.java:95)
E/AndroidRuntime( 6593):     at android.os.Looper.loop(Looper.java:135)
E/AndroidRuntime( 6593):     at android.app.ActivityThread.main(ActivityThread.java:5254)
E/AndroidRuntime( 6593):     at java.lang.reflect.Method.invoke(Native Method)
E/AndroidRuntime( 6593):     at java.lang.reflect.Method.invoke(Method.java:372)
E/AndroidRuntime( 6593):     at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:903)
E/AndroidRuntime( 6593):     at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:698)
```

2.5 其他命令【了解】

序		
---	--	--

号		
01	<code>adb install 路径/xx.apk</code>	安装 app 到手机
02	<code>adb uninstall 包名</code>	卸载手机上的 app，需要指定包名
03	<code>adb devices</code>	获取当前电脑已经连接设备和对应的设备号
04	<code>adb shell</code>	进入到安卓手机内部的linux系统命令行中
05	<code>adb start-server</code>	启动 adb 服务端，出 bug 时使用可以重启服务器，先关闭再启动
06	<code>adb kill-server</code>	停止 adb 服务端，出 bug 时使用可以重启服务器，先关闭再启动
07	<code>adb --help</code>	查看 adb 帮助，命令记不清楚时有用

UIAutomatorViewer工具

目标

1. 能够使用 UIAutomatorViewer 获取元素的特征信息

1. UIAutomatorViewer 的使用

应用场景

定位元素的时候必须根据元素的相关特征来进行定位，而 UIAutomatorViewer 就是用来获取元素特征的。

简介

UIAutomatorViewer 用来扫描和分析 Android 应用程序的 UI 控件的工具。

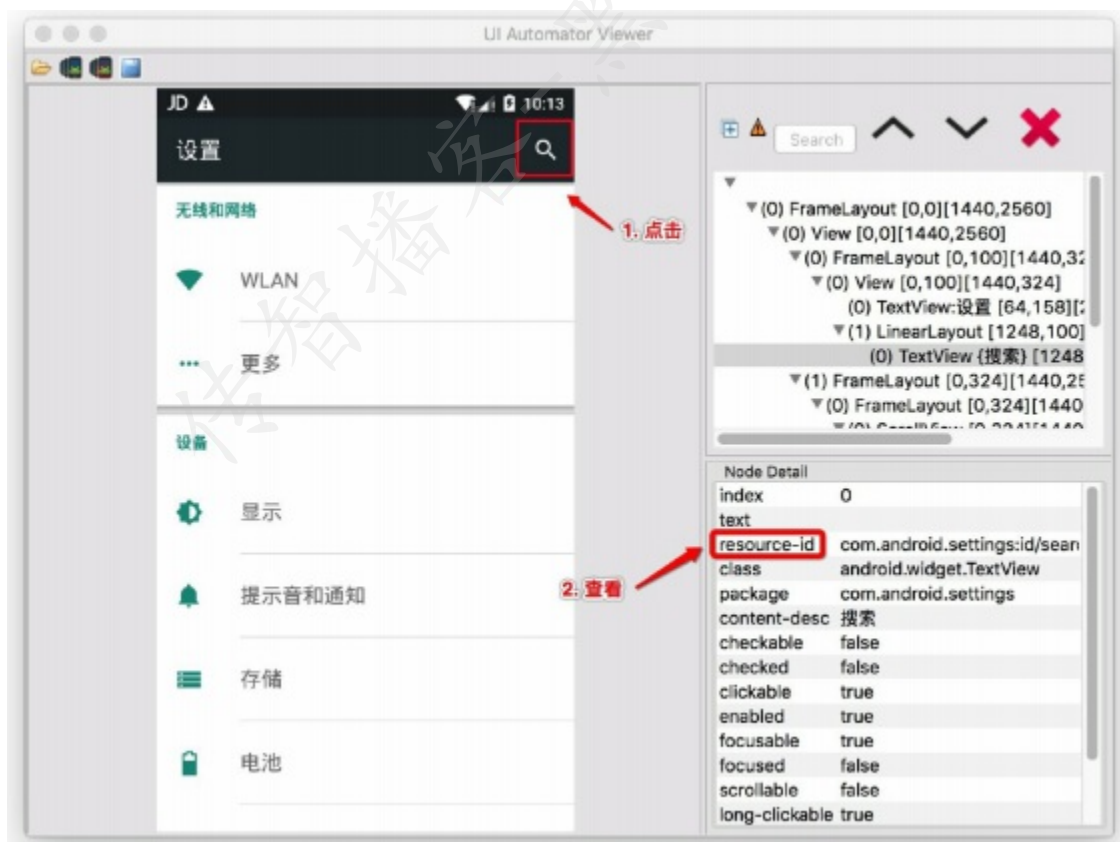
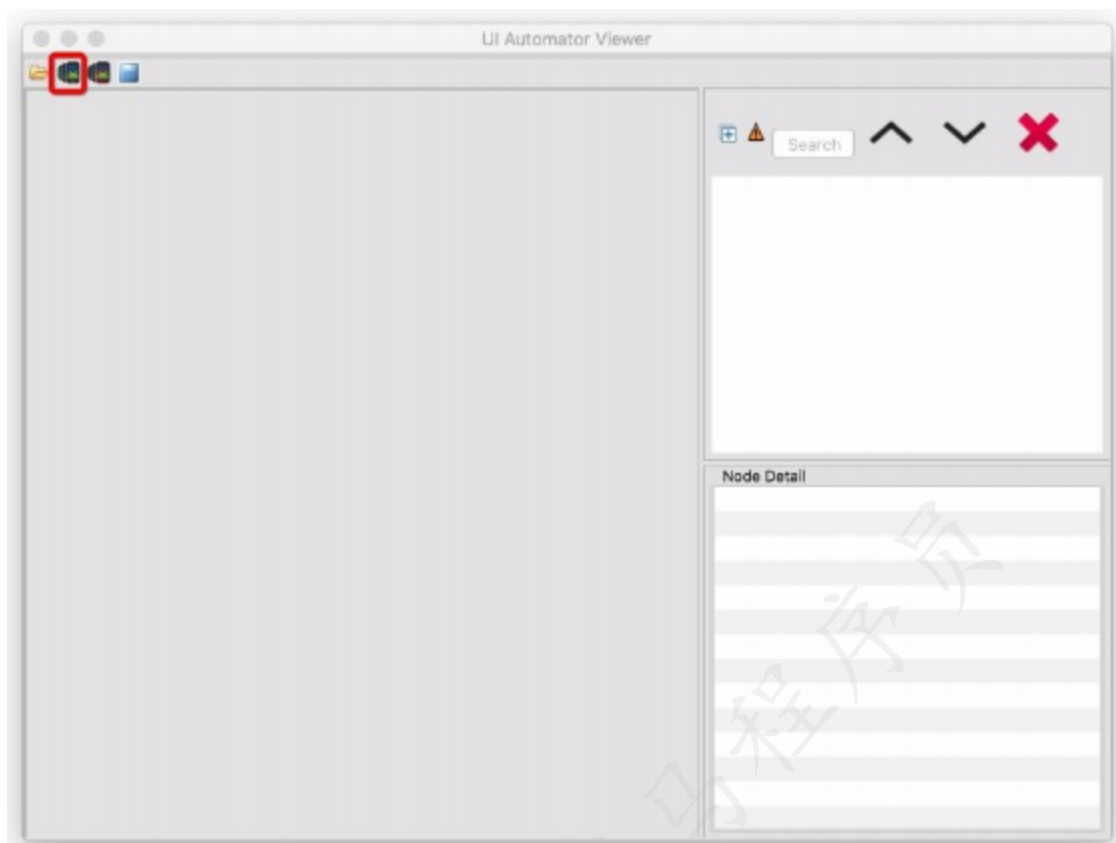
使用步骤

1. 进入SDK目录下的目录
 - mac 在 tools/bin 目录下，打开 uiautomatorviewer
 - windows 在 tools 目录下，打开 uiautomatorviewer.bat
2. 电脑连接真机或打开android模拟器
3. 启动待测试app
4. 点击 uiautomatorviewer 的左上角 Device Screenshot （从左数第二个按钮）
5. 点击希望查看的控件
6. 查看右下角 Node Detail 相关信息

示例

查看《设置》应用程序右上角“放大镜”按钮的“resource-id”

1. 打开 uiautomatorviewer
2. 打开 android 模拟器
3. 启动《设置》应用程序
4. 点击 Device Screenshot 按钮
5. 点击“放大镜”按钮
6. 查看 Node Detail 中的“resource-id”信息



注意点

1. 自动打开的命令行窗口不要关

- 如果关了，整个工具也会关闭
- 2. 打开uiautomatorviewer闪退
 - 解决方案：jdk版本问题造成的，jdk为1.9时可能会出现这个问题，请换成1.8的版本
- 3. 点击第二个按钮报错

-
- 解决方案：
 - `adb kill-server`
 - `adb start-server`

入门示例

目标

1. 能够使用 `appium` 启动任意应用程序
2. 能够了解 “前置代码” 中各项参数的含义

1. 快速体验

应用场景

在做app自动化的时候，我们肯定是针对某个产品、某个软件进行测试，那么我们一定是先让模拟器或真机帮我们打开这款软件才可以。所以接下来要学的就是如何打开某个应用程序。

需求

使用以下步骤可以打开模拟器中的 《设置》 应用程序

步骤

由于是快速体验，我们先进行复制粘贴代码看到效果后，再进行讲解

1. 打开手机模拟器
2. 打开appium工具
3. 创建一个python项目，取名为 `hello_appium`
4. 创建一个 `demo.py` 文件
5. 将下面代码直接复制，并运行即可

```
from appium import webdriver

desired_caps = dict()
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'

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

driver.quit()
```

注意点：

这段代码实际上配置了一些启动应用程序的相关参数。之后的其他项目也需要用到这个参数，可能有些参数配置的内容不同。为了方便我们后期课程会将这段代码叫做“前置代码”

2. 启动参数详解【掌握】

应用场景

如果后期项目不是测试《设置》应用程序，而是测试《短信》应用程序那么怎么打开《短信》应用程序呢？如果后期项目测试的模拟器或手机不再是5.1的版本，而是6.1的版本呢？相关配置的信息在学习之后都可以进行修改。

参数解释

```
# 导模块
from appium import webdriver

# 创建一个字典，包装相应的启动参数
desired_caps = dict()
# 需要连接的手机的平台(不限制大小写)
desired_caps['platformName'] = 'Android'
# 需要连接的手机的版本号(比如 5.2.1 的版本可以填写 5.2.1 或 5.2 或 5 ，以此类推)
desired_caps['platformVersion'] = '5.1'
# 需要连接的手机的设备号(android平台下，可以随便写，但是不能不写)
desired_caps['deviceName'] = '192.168.56.101:5555'
# 需要启动的程序的包名
desired_caps['appPackage'] = 'com.android.settings'
# 需要启动的程序的界面名
desired_caps['appActivity'] = '.Settings'

# 连接appium服务器
driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

# 退出
driver.quit()
```

启动过程【了解】

appium的启动实际上是在本机使用了4723端口开启了一个服务

1. 我们写的 python 代码会访问本机的 appium 服务器，并获取 driver 对象
2. appium 会将我们的 driver 对象调用的方法转化成 post 请求，提交给appium服务器
3. appium 通过接收到的 post 请求发送给手机，再由手机进行执行

Appium基础操作

目标

1. 能够使用 appium 在脚本内启动其他 app
2. 能够使用 appium 获取包名和界面名
3. 能够使用 appium 关闭 app 和 驱动对象
4. 能够使用 appium 安装和卸载 app
5. 能够使用 appium 将应用置于后台

1. Appium 基础操作 API

前置代码

```
from appium import webdriver

desired_caps = dict()
# 手机参数
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '5.1'
desired_caps['deviceName'] = '192.168.56.101:5555'
# 应用参数
desired_caps['appPackage'] = 'com.android.settings'
desired_caps['appActivity'] = '.Settings'

# 获取driver
driver = webdriver.Remote('http://localhost:4723/wd/hub',desired_caps)

# 退出driver
driver.quit()
```

1.1 在脚本内启动其他 app

应用场景

如果一个应用需要跳转到另外一个应用，就可以使用这个 api 进行应用的跳转，就像我们通过外卖应用下订单之后会跳转到支付应用一样。

方法名和参数

```
# 脚本内启动其他app
# 参数:
#     appPackage: 要打开的程序的包名
```

```
# appActivity: 要打开的程序的界面名
driver.start_activity(appPackage, appActivity)
```

示例

打开《设置》应用程序，等待三秒后跳转到《短信》应用程序

```
from appium import webdriver

desired_caps = dict()
# 手机参数
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '5.1'
desired_caps['deviceName'] = '192.168.56.101:5555'
# 应用参数
desired_caps['appPackage'] = 'com.android.settings'
desired_caps['appActivity'] = '.Settings'

# 获取driver
driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

# 跳转到短信
driver.start_activity('com.android.mms', '.ui.ConversationList')

time.sleep(5)

# 退出driver
driver.quit()
```

1.2 获取 app 的包名和界面名

应用场景

当我们从一个应用跳转到另外一个应用的时候，想输出其包名、界面名或者想在报告中展现对应信息，我们就可以调用这个属性来进行获取

属性名

```
# 获取包名
driver.current_package
# 获取界面名
driver.current_activity
```

示例

打开《设置》应用程序后输出当前的包名和界面名

```

from appium import webdriver

desired_caps = dict()
# 手机参数
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '5.1'
desired_caps['deviceName'] = '192.168.56.101:5555'
# 应用参数
desired_caps['appPackage'] = 'com.android.settings'
desired_caps['appActivity'] = '.Settings'

# 获取driver
driver = webdriver.Remote('http://localhost:4723/wd/hub',desired_caps)

# 打印当前包名
print(driver.current_package)
# 打印当前界面名
print(driver.current_activity)

# 退出driver
driver.quit()

```

结果

```

com.android.settings
.Settings

```

1.3 关闭 app 和 驱动对象

应用场景

有的时候我们需要关闭某个应用程序后，再打开新的应用。那么如何关闭应用程序呢？

方法名

```

# 关闭当前操作的app，不会关闭驱动对象
driver.close_app()
# 关闭驱动对象，同时关闭所有关联的app
driver.quit()

```

示例

打开《设置》，使用 close_app() 方法关闭，再尝试使用 quit() 方法，最后打印当前程序的包名，观察区别

```

from appium import webdriver

```

```

desired_caps = dict()
# 手机参数
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '5.1'
desired_caps['deviceName'] = '192.168.56.101:5555'
# 应用参数
desired_caps['appPackage'] = 'com.android.settings'
desired_caps['appActivity'] = '.Settings'

# 获取driver
driver = webdriver.Remote('http://localhost:4723/wd/hub',desired_caps)

# 关闭应用
driver.close_app()

# 退出driver
driver.quit()

```

结果

使用 quit() 后会报错，使用close_app() 不会报错

小结

close_app() 不会关闭驱动对象，只会关闭应用

quit() 会关闭驱动对象

1.4 安装和卸载以及是否安装 app

应用场景

一些应用市场的软件可能会有一个按钮，如果某一个程序已经安装则卸载，如果没有安装则安装

方法名

```

# 安装app
# 参数:
#     app_path: apk路径
driver.install_app(app_path)

```

```

# 卸载app
# 参数:
#     app_id: 应用程序包名
driver.remove_app(app_id)

```

```
# 判断app是否已经安装
# 参数:
#     app_id: 应用程序包名
# 返回值:
#     布尔类型, True为安装, False为没有安装
driver.is_app_installed(app_id)
```

示例

如果《安智市场》已经安装, 则卸载《安智市场》, 如果没有则安装

```
from appium import webdriver

desired_caps = dict()
# 手机参数
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '5.1'
desired_caps['deviceName'] = '192.168.56.101:5555'
# 应用参数
desired_caps['appPackage'] = 'com.android.settings'
desired_caps['appActivity'] = '.Settings'

# 获取driver
driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

if driver.is_app_installed("cn.goapk.market"):
    driver.remove_app("cn.goapk.market")
else:
    driver.install_app("/Users/Yoson/Desktop/anzhishichang.apk")

# 退出driver
driver.quit()
```

1.5 将应用置于后台

应用场景

银行类 app 会在进入后台一定时间后, 如果再回到前台也页面会重新输入密码, 如果需要自动化测试这种功能, 可以使用这个 api 进行测试

方法

```
# app放置到后台一定时间后再回到前台, 模拟热启动
# 参数:
#     seconds: 后台停留多少秒
driver.background_app(seconds)
```

示例

打开《设置》应用，进入后台 5 秒，再回到前台

```
from appium import webdriver

desired_caps = dict()
# 手机参数
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '5.1'
desired_caps['deviceName'] = '192.168.56.101:5555'
# 应用参数
desired_caps['appPackage'] = 'com.android.settings'
desired_caps['appActivity'] = '.Settings'

# 获取driver
driver = webdriver.Remote('http://localhost:4723/wd/hub',desired_caps)

time.sleep(3)
driver.background_app(5)
time.sleep(3)

# 退出driver
driver.quit()
```

注意点

热启动：表示进入后台回到前台。关机再开这种切断电源的行为可以叫做“冷启动”

元素定位

目标

1. 能够分别使用 `id`、`class`、`xpath` 定位某一个元素
2. 能够分别使用 `id`、`class`、`xpath` 定位某一组元素

1. 元素定位操作 API

应用场景

计算机不像人一样“聪明”，我们需要通过元素定位来获取元素，才能让计算机帮我们“操作”这个元素。

步骤

1. 打开 `uiautomatorviewer` 工具
2. 打开模拟器或真机
3. 通过 `uiautomatorviewer` 工具获取想要进行操作的元素的 `Node Detail` 信息
4. 通过元素定位 API 进行定位
5. 对元素进行相关操作

注意点

元素的定位基于当前屏幕范围内展示的可见元素。

1.1 定位一个元素【掌握】

应用场景

想要对按钮进行点击，想要对输入框进行输入，想要获取文本框的内容，定位元素是自动化操作必须要使用的方法。只有获取元素之后，才能对这个元素进行操作。

方法名

```
# 通过id定位一个元素
# 参数：
#     id_value: 元素的resource-id属性值
# 返回值：
#     定位到的单个元素
driver.find_element_by_id(id_value)
```

```
# 通过class_name定位一个元素
```

```
# 参数:
#     class_value: 元素的class属性值
# 返回值:
#     定位到的单个元素
driver.find_element_by_class_name(class_value)
```

```
# 通过xpath定位一个元素
# 参数:
#     xpath_value: 定位元素的xpath表达式
# 返回值:
#     定位到的单个元素
driver.find_element_by_xpath(xpath_value)
```

示例

通过 id 的形式，定位 “放大镜” 按钮，并点击

通过 class 的形式，定位 “输入框”，输入 “hello”

通过 xpath 的形式，定位 “返回” 按钮，并点击

关键代码

```
driver.find_element_by_id("com.android.settings:id/search").click()
driver.find_element_by_class_name("android.widget.EditText").send_keys("hello")
driver.find_element_by_xpath("//*[@content-desc='收起']").click()
```

小结

1. find_element_by_id 方法中传入的是 Node Detail 信息中的 resource-id
2. find_element_by_class_name 方法中传入的是 Node Detail 信息中的 class
3. find_element_by_xpath 方法中传入的是 Node Detail 信息中的 xpath表达式

注意点

如果很多元素的 “特征” 相同，使用 find_element_by_xxx 的方法会找到第一个

1.2 定位一组元素【掌握】

应用场景

和定位一个元素相同，但如果想要批量的获取某个相同特征的元素，使用定位一组元素的方式更加方便。

方法名

```
# 通过id定位一组元素
```



```
# 参数:
#     id_value: 元素的resource-id属性值
# 返回值:
#     列表, 定位到的所有符合调价你的元素
driver.find_elements_by_id(id_value)
```

```
# 通过class_name定位一组元素
# 参数:
#     class_value: 元素的class属性值
# 返回值:
#     列表, 定位到的所有符合调价你的元素
driver.find_elements_by_class_name(class_value)
```

```
# 通过xpath定位一组元素
# 参数:
#     xpath_value: 定位元素的xpath表达式
# 返回值:
#     列表, 定位到的所有符合调价你的元素
driver.find_elements_by_xpath(xpath_value)
```

示例

通过 id 的形式, 获取所有 resource-id 为 "com.android.settings:id/title" 的元素, 并打印其文字内容

通过 class_name 的形式, 获取所有class 为 "android.widget.TextView" 的元素, 并打印其文字内容

通过 xpath 的形式, 获取所有包含 "设" 的元素, 并打印其文字内容

关键代码

```
titles = driver.find_elements_by_id("com.android.settings:id/title")
for title in titles:
    print(title.text)

text_views = driver.find_element_by_class_name("android.widget.TextView")
for text_view in text_views:
    print(text_view.text)

elements = driver.find_element_by_xpath("//*[contains(@text,'设')]")
for element in elements:
    print(element.text)
```

1.3 定位元素的注意点【了解】

应用场景

了解这些注意点可以以后在出错误的时候，更快速的定位问题原因。

示例

使用 `find_element_by_xx` 或 `find_elements_by_xx` 的方法，分别传入一个没有的 "特征" 会是什么结果呢？

核心代码

```
driver.find_element_by_id("xxx")
```

```
driver.find_elements_by_id("xxx")
```

小结

1. 如果使用 `find_element_by_xx` 方法，如果传入一个没有的特征，会报 `NoSuchElementException` 的错误。
2. 如果使用 `find_elements_by_xx` 方法，如果传入一个没有的特征，不会报错，会返回一个空列表

元素操作

目标

1. 能够使用代码点击按钮
2. 能够使用代码对输入框输入文字
3. 能够使用代码对输入框清空文字
4. 能够使用代码获取元素的文本内容
5. 能够使用代码获取元素的位置和大小
6. 能够使用代码根据属性名获取元素的属性值

1. 元素操作API

1.1 点击元素

应用场景

需要点击某个按钮的时候使用

方法名

```
# 对element按钮进行点击操作  
element.click()
```

示例

1. 打开《设置》
2. 点击放大镜按钮

核心代码

```
driver.find_element_by_id("com.android.settings:id/search").click()
```

1.2 输入和清空输入框内容

应用场景

需要对输入框进行输入或清空的时候使用

方法名

```
# 对element输入框进行输入操作  
# 参数:
```

```
# value: 输入的内容
element.send_keys(value)
# 对element输入框进行输入操作
element.clear()
```

示例

1. 打开《设置》
2. 点击 ”放大镜“
3. 输入 ”hello“
4. 暂停 2 秒
5. 清空所有文本内容
6. 暂停 5 秒
7. 输入 ”你好“

核心代码

```
driver.find_element_by_id("com.android.settings:id/search").click()
edit_text = driver.find_element_by_class_name("android.widget.EditText")
edit_text.send_keys("hello")
time.sleep(2)
edit_text.clear()
time.sleep(5)
edit_text.send_keys("你好")
```

注意点

默认输入中文无效，但不会报错，需要在 ”前置代码“ 中增加两个参数

```
desired_caps['unicodeKeyboard'] = True
desired_caps['resetKeyboard'] = True
```

1.3 获取元素的文本内容

应用场景

需要获取按钮、文本框、输入框等控件的文本内容时使用

属性名

```
# 获取element控件的文本内容
# 返回值:
# 控件的文本内容
element.text
```

示例

1. 打开《设置》
2. 获取所有 resource-id 为 "com.android.settings:id/title" 的元素，并打印其文字内容

核心代码

```
titles = driver.find_elements_by_id("com.android.settings:id/title")
for title in titles:
    print(title.text)
```

1.4 获取元素的位置和大小

应用场景

需要获取元素的位置和大小的时候使用

属性名

```
# 获取element的位置
# 返回值:
#     字典, x为元素的x坐标, y为元素的y坐标
element.location
# 获取element的大小
# 返回值:
#     字典, width为宽度, height为告诉
element.size
```

示例

1. 打开《设置》
2. 获取 "放大镜" 的位置和大小

核心代码

```
search_button = driver.find_element_by_id("com.android.settings:id/search")
print(search_button.location)
print(search_button.size)
```

1.5 获取元素的属性值

应用场景

根据特征定位到元素后，使元素的属性名获取对应的属性值

方法名

```
# 对element进行点击操作
# 参数:
#     value: 要获取的属性名
# 返回值:
#     根据属性名得到的属性值
element.get_attribute(value) # value:元素的属性
```

示例

1. 打开《设置》
2. 获取所有 resource-id 为 "com.android.settings:id/title" 的元素
3. 使用 get_attribute 获取这些元素的 enabled、text、content-desc、resource-id、class 的属性值

核心代码

```
titles = driver.find_elements_by_id("com.android.settings:id/title")
for title in titles:
    print(title.get_attribute("enabled"))
    print(title.get_attribute("text"))
    print(title.get_attribute("name"))
    print(title.get_attribute("resourceId"))
    print(title.get_attribute("ClassName"))
```

注意点

value='text' 返回text的属性值

value='name' 返回content-desc / text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

滑动和拖拽事件

目标

1. 能够使用 `swipe` 滑动屏幕
2. 能够使用 `scroll` 滑动屏幕
3. 能够使用 `drag_and_drop` 滑动屏幕

1. 滑动和拖拽事件

应用场景

我们在做自动化测试的时候，有些按钮是需要滑动几次屏幕后才会出现，此时，我们需要使用代码来模拟手指的滑动，也就是我们将来学习的滑动和拖拽事件

1.1 `swipe` 滑动事件

概念

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。

方法名

```
# 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
# 参数:
#     start_x: 起点X轴坐标
#     start_y: 起点Y轴坐标
#     end_x: 终点X轴坐标
#     end_y: 终点Y轴坐标
#     duration: 滑动这个操作一共持续的时间长度，单位: ms
driver.swipe(start_x, start_y, end_x, end_y, duration=None)
```

示例1

模拟手指从（100, 2000），滑动到（100, 1000）的位置

核心代码

```
driver.swipe(100, 2000, 100, 1000)
```

示例2

模拟手指从（100, 2000），滑动到（100, 100）的位置

核心代码

```
driver.swipe(100, 2000, 100, 100)
```

示例3

模拟手指从（100, 2000），滑动到（100, 100）的位置，持续5秒

核心代码

```
driver.swipe(100, 2000, 100, 100, 5000)
```

小结

距离相同时，持续时间越长，惯性越小

持续时间相同时，手指滑动的距离越大，实际滑动的距离也就越大

1.2 scroll 滑动事件

概念

从一个元素滑动到另一个元素，直到页面自动停止。

方法名

```
# 从一个元素滑动到另一个元素，直到页面自动停止
# 参数:
#   origin_el:      滑动开始的元素
#   destination_el: 滑动结束的元素
driver.scroll(origin_el, destination_el)
```

示例

从“存储”滑动到“更多”

核心代码

```
save_button = driver.find_element_by_xpath("//*[@text='存储']")
more_button = driver.find_element_by_xpath("//*[@text='更多']")
driver.scroll(save_button, more_button)
```

小结

不能设置持续时间，惯性很大

1.3 drag_and_drop 拖拽事件

概念

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。

方法名

```
# 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
# 参数:
#     origin_el:      滑动开始的元素
#     destination_el: 滑动结束的元素
driver.drag_and_drop(origin_el, destination_el)
```

示例

将“存储”拖拽到“更多”

核心代码

```
save_button = driver.find_element_by_xpath("//*[@text='存储']")
more_button = driver.find_element_by_xpath("//*[@text='更多']")
driver.drag_and_drop(save_button, more_button)
```

小结

不能设置持续时间，没有惯性

2. 滑动和拖拽事件的选择

滑动和拖拽无非就是考虑是否有“惯性”，以及传递的参数是“元素”还是“坐标”。

可以分成以下四种情况

- 有“惯性”，传入“元素”
 - `scroll`
- 无“惯性”，传入“元素”
 - `drag_and_drop`
- 有“惯性”，传入“坐标”
 - `swipe`，并且设置较短的 `duration` 时间
- 无“惯性”，传入“坐标”
 - `swipe`，并且设置较长的 `duration` 时间

高级手势TouchAction

目标

1. 能够使用代码完成轻敲手势
2. 能够使用代码完成按下手势
3. 能够使用代码完成抬起手势
4. 能够使用代码完成等待操作
5. 能够使用代码完成长按手势
6. 能够使用代码完成手指移动操作

1. 高级手势TouchAction

应用场景

TouchAction 可以实现一些针对手势的操作，比如滑动、长按、拖动等。我们可以将这些基本手势组合成一个相对复杂的手势。比如，我们解锁手机或者一些应用软件都有手势解锁的这种方式。

使用步骤

1. 创建 **TouchAction** 对象
2. 通过对象调用想执行的手势
3. 通过 **perform()** 执行动作

注意点

所有手势都要通过执行**perform()**函数才会运行。

1.1 轻敲操作【掌握】

应用场景

模拟手指对某个元素或坐标按下并快速抬起。比如，固定点击（100, 100）的位置。

方法名

```
# 模拟手指对元素或坐标的轻敲操作
# 参数:
#     element: 元素
#     x: x坐标
#     y: y坐标
TouchAction(driver).tap(element=None, x=None, y=None).perform()
```

示例

1. 打开《设置》
2. 轻敲“WLAN”

核心代码

```
el = driver.find_element_by_xpath("//*[@contains(@text,'WLAN')]")
TouchAction(driver).tap(el).perform()
```

1.2 按下和抬起操作【掌握】

应用场景

模拟手指一直按下，模拟手指抬起。可以用来组合成轻敲或长按的操作

方法名

```
# 模拟手指对元素或坐标的按下操作
# 参数:
#     el: 元素
#     x: x坐标
#     y: y坐标
TouchAction(driver).press(el=None, x=None, y=None).perform()
```

```
# 模拟手指对元素或坐标的抬起操作
TouchAction(driver).release().perform()
```

示例1

使用坐标的形式按下 WLAN（650, 650），2 秒后，按下（650, 650）的位置

核心代码

```
TouchAction(driver).press(x=650, y=650).perform()
time.sleep(2)
TouchAction(driver).press(x=650, y=650).perform()
```

示例2

使用坐标的形式按下 WLAN（650, 650），2 秒后，按下（650, 650）的位置，并抬起

核心代码

```
TouchAction(driver).press(x=650, y=650).perform()
time.sleep(2)
TouchAction(driver).press(x=650, y=650).release().perform()
```

1.3 等待操作【掌握】

应用场景

模拟手指等待，比如按下后等待 5 秒之后再抬起。

方法名

```
# 模拟手指暂定操作
# 参数:
#     ms: 暂停的毫秒数
TouchAction(driver).wait(ms=0).perform()
```

示例

使用坐标的形式点击 WLAN（650, 650），2 秒后，按下（650, 650）的位置，暂停 2 秒，并抬起

核心代码

```
TouchAction(driver).tap(x=650, y=650).perform()
time.sleep(2)
TouchAction(driver).press(x=650, y=650).wait(2000).release().perform()
```

1.4 长按操作【掌握】

应用场景

模拟手指对元素或坐标的长按操作。比如，长按某个按钮弹出菜单。

方法名

```
# 模拟手指对元素或坐标的长按操作
# 参数:
#     el: 元素
#     x: x坐标
#     y: y坐标
#     duration: 长按时间，毫秒
TouchAction(driver).long_press(el=None, x=None, y=None, duration=1000).perform()
```

示例

使用坐标的形式点击 WLAN（650, 650），2 秒后，长按（650, 650）的位置持续 2 秒

核心代码

```
TouchAction(driver).tap(x=400, y=400).perform()
```

```
time.sleep(2)
TouchAction(driver).long_press(x=400, y=400, duration=2000).release().perform()
```

1.5 移动操作【掌握】

应用场景

模拟手指移动操作，比如，手势解锁需要先按下，再移动。

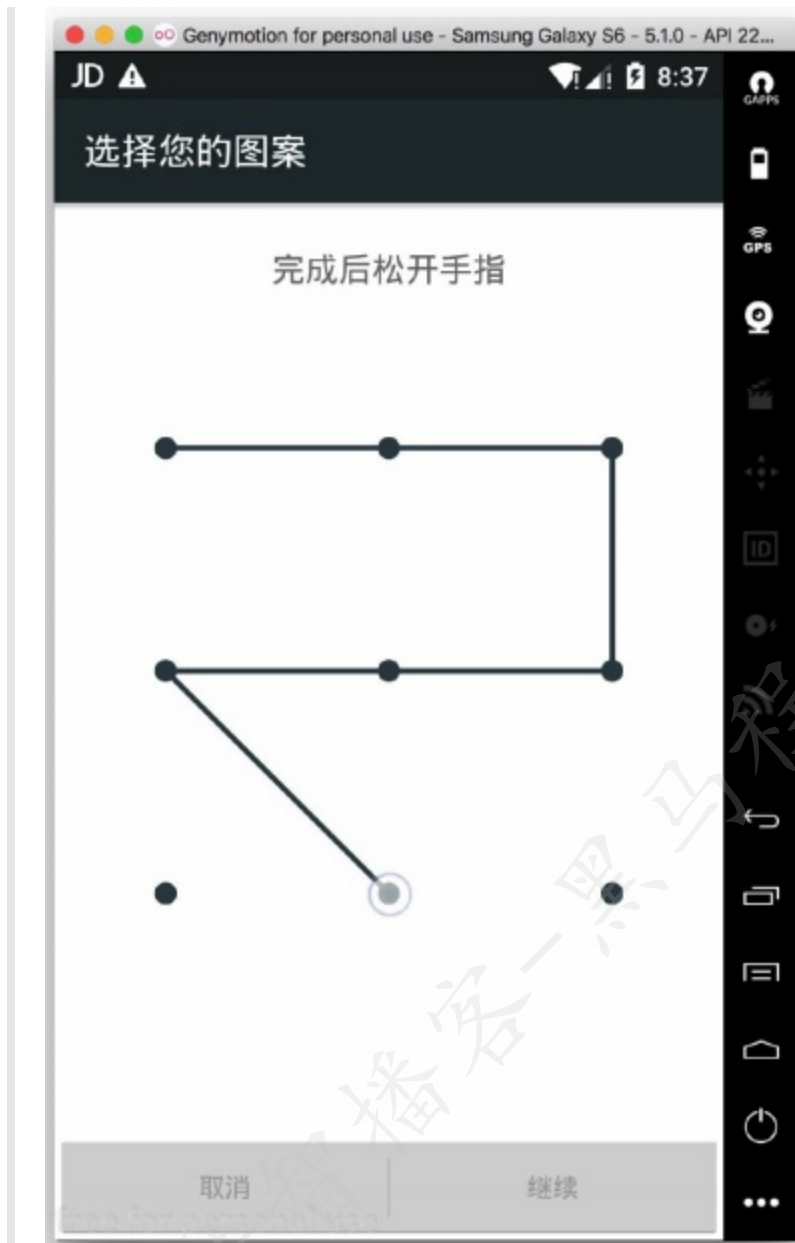
方法名

```
# 模拟手指对元素或坐标的移动操作
# 参数:
#     el: 元素
#     x: x坐标
#     y: y坐标
TouchAction(driver).move_to(el=None, x=None, y=None).perform()
```

示例

在手势解锁中，画一个如下图的案例

包名界面名为 **com.android.settings/.ChooseLockPattern**



核心代码

```
TouchAction(driver).press(x=246, y=857).move_to(x=721, y=867).move_to(x=1200, y=851).move_to(x=1200, y=1329).move_to(x=724, y=1329).move_to(x=246, y=1329).move_to(x=718, y=1815).release().perform()
```

手机操作

目标

1. 能够获取手机分辨率
2. 能够获取手机截图
3. 能够获取和设置网络状态
4. 能够发送键到设备
5. 能够打开和关闭手机通知栏

1. 手机操作API

1.1 获取手机分辨率【掌握】

应用场景

自动化测试可能会需要根据当前设备的屏幕分辨率来计算一些点击或者滑动的坐标

方法名

```
# 获取手机分辨率  
driver.get_window_size()
```

示例

输出当前设备的屏幕分辨率

核心代码

```
print(driver.get_window_size())
```

执行结果

```
{'height': 800, 'width': 480}
```

1.2 手机截图【掌握】

应用场景

有些自动化的操作可能没有反应，但并不报错。此时我们就可以将操作过后的关键情况，截图留存。后期也可以根据图片发现问题。

方法名

```
# 获取手机分辨率
# 参数:
#     filename: 指定路径下, 指定格式的图片
get_screenshot_as_file(filename)
```

示例

1. 打开设置页面
2. 截图当前页面保存到当前目录, 命名为screen.png

核心代码

```
driver.get_screenshot_as_file(os.getcwd() + os.sep + './screen.png')
```

执行效果

项目目录下会将设置页面保存成 screen.png

1.3 获取和设置手机网络【掌握】

应用场景

视频应用在使用流量看视频的时候, 大部分都会提示用户正在是否继续播放。作为测试人员, 我们可能需要用自动化的形式来判断是否有对应的提示。即, 用流量的时候应该有提示, 不用流量的时候应该没有提示。

1.3.1 获取手机网络

属性名

```
# 获取手机网络
driver.network_connection
```

示例

获取当前网络类型, 并打印

核心代码

```
print(driver.network_connection)
```

执行结果

6

结果对照

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

1.3.2 设置手机网络

方法名

```
# 设置手机网络
# 参数:
#     connectionType: 网络类型
driver.set_network_connection(connectionType)
```

示例

设置当前设备为飞行模式

核心代码

```
driver.set_network_connection(1)
```

执行效果

设备变为飞行模式

1.4 发送键到设备【掌握】

应用场景

模拟按“返回键”“home键”等等操作，比如，很多应用有按两次返回键退出应用的功能，如果这个功能需要我们做自动化，那么一定会用到这个方法

方法名

```
# 发送键到设备
# 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
driver.press_keycode(keycode, metastate=None)
```

注意点

按键对应的编码，可以在百度搜索关键字“android keycode”

例如: <https://blog.csdn.net/feizhixuan46789/article/details/16801429>

示例

点击三次音量加, 再点击返回, 再点击两次音量减。

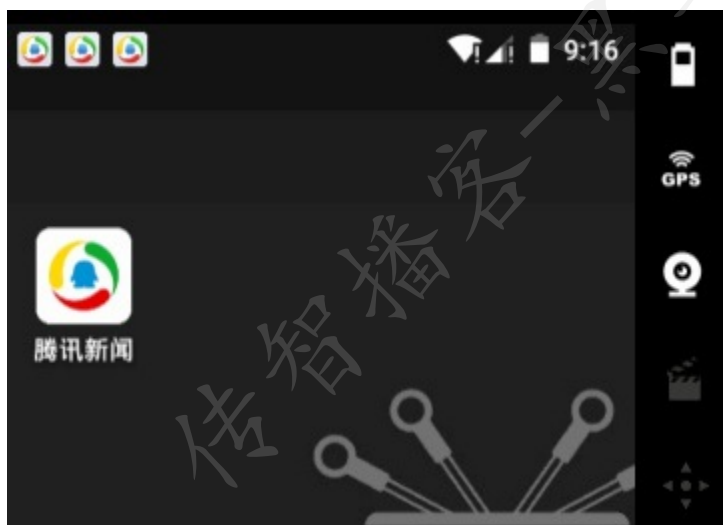
核心代码

```
driver.press_keycode(24)
driver.press_keycode(24)
driver.press_keycode(24)
driver.press_keycode(4)
driver.press_keycode(25)
driver.press_keycode(25)
```

1.5 操作手机通知栏【掌握】

应用场景

测试即时通信类软件的时候, 如果 A 给 B 发送一条消息, B 的通知栏肯定会显示对应的消息。我们想通过通知栏来判断 B 是否收到消息, 一定要先操作手机的通知栏



方法名

```
# 打开手机通知栏
driver.open_notifications()
```

注意点

appium官方并没有为我们提供关闭通知的api, 那么现实生活中怎么关闭, 就怎样操作就行, 比如, 手指从下往上滑动, 或者, 按返回键

示例

打开通知栏，两秒后，关闭通知栏

核心代码

```
driver.open_notifications()  
time.sleep(2)  
driver.press_keycode(4)
```

扩展

目标

1. 能够获取 toast 内容
2. 能够通过 chrome 查看 WebView 中的元素
3. 能够使用代码获取 WebView 中的元素
4. 能够解决没有 chromedriver 的问题
5. 能够使用 Monkey 对 app 进行压力测试
6. 能够理解 Monkey 各项参数的含义
7. 能够理解异常情况时对应的效果
8. 能够将脚本运行在真机上

1. Toast

1.1 安装环境

安装注意事项:

1. 安装之前最好先运行一下代码，如果可以直接使用，那么就不必再进行安装。
2. 安装过程可能需要对C盘进行读写，请确保使用的是管理员权限。最好使用 power shell。
3. 安装过程中可能会有各种问题，尽量使用手机网络进行安装。

1. 安装node.js （使用 npm 或 node 验证）

```
node-v8.11.3-x64.msi(windows) 或 node-v8.10.0.pkg(mac) 进行安装
```

2. 安装cnpm （使用cnpm验证）

```
npm install -g cnpm --registry=https://registry.npm.taobao.org
```

3. 下载 appium-uiautomator2-driver

```
cnpm install appium-uiautomator2-driver
```

```
Yoson -- bash -- 80x23
itheima-2:~ itheima$ cnpm install appium-uiautomator2-driver
✓ Installed 1 packages
✓ Linked 0 latest versions
✓ Run 0 scripts
✓ All packages installed (used 203ms(network 201ms), speed 50.52kB/s, json 1(10.15kB), tarball 0B)
itheima-2:~ itheima$
```

使用 `npm install` 或者 `cnpm install` 安装完成后，都会提示 `Installed xx packages` 或者 `All packages installed` 只要看到这种，就说名成功了。如果不成功则请确保按照注意点做后，再次使用相同的命令重试。或尝试清除npm或者cnpm的缓存，命令如下：

npm清缓存使用：

```
npm cache clean --force
npm cache verify
npm config set strict-ssl false
```

cnpm清缓存使用：

```
cnpm cache clean --force
cnpm cache verify
cnpm config set strict-ssl false
```

1.2 获取Toast内容

应用场景

举个例子，输入用户名和密码然后点登录之后，会弹出一个 `toast` 的弹框。我们可以如果学会查找 `toast`，就可以使用这个登录的 `toast` 来进行断言的判断操作。

步骤

1. 前置代码添加

```
desired_caps['automationName'] = 'UiAutomator2'
```

2. 使用xpath找text即可

```
def find_toast(driver, message, timeout=3):  
    """  
    # message: 预期要获取的toast的部分消息  
    """  
    message = "//*[contains(@text,'" + message + "')] " # 使用包含的方式定位  
  
    element = WebDriverWait(driver, timeout, 0.1).until(lambda x: x.find_element(By.XPATH, message))  
    return element.text
```

2. WebView

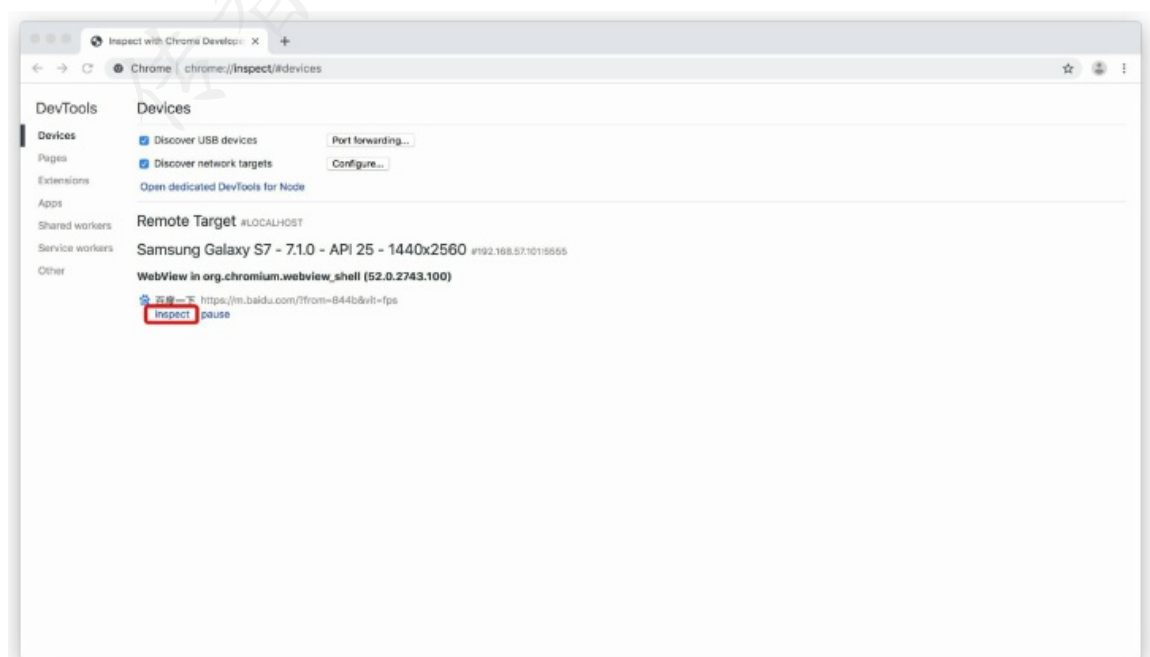
2.1 查看webview元素的方式

通过 **chrome** 直接连接手机查看

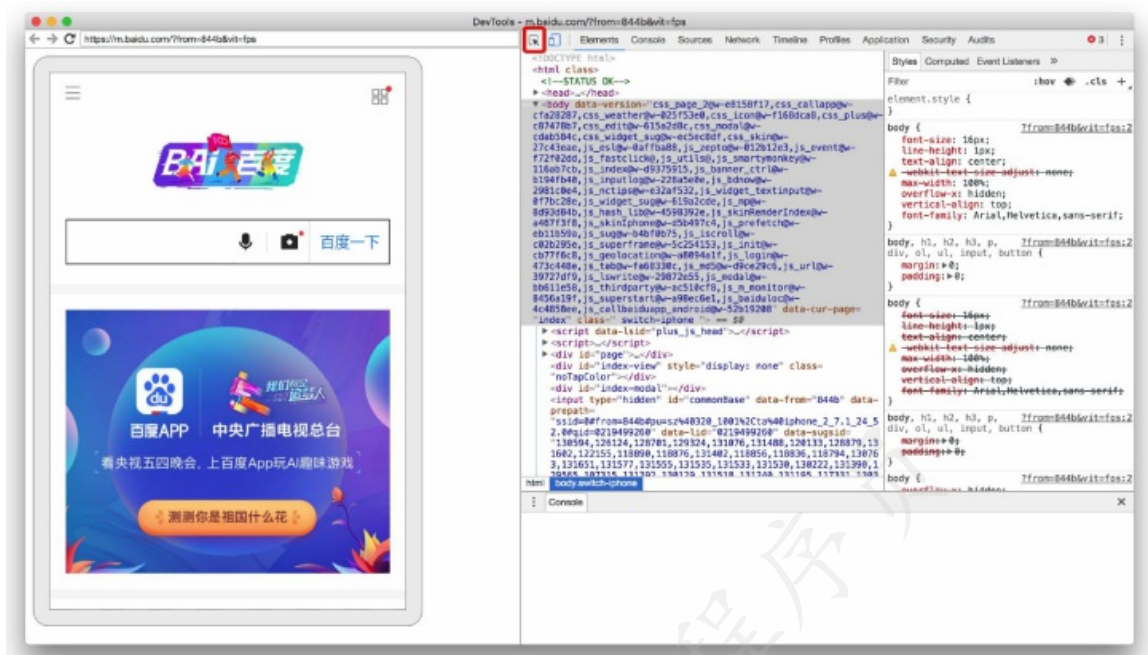
1. 使用 genymotion 打开需要查看的 webview 界面



2. 在 chrome 中输入 chrome://inspect 地址，并点击 inspect



3. 选中 "select an element..." 选项



4. 选中要查看的元素，即可自动跳转到对应的代码

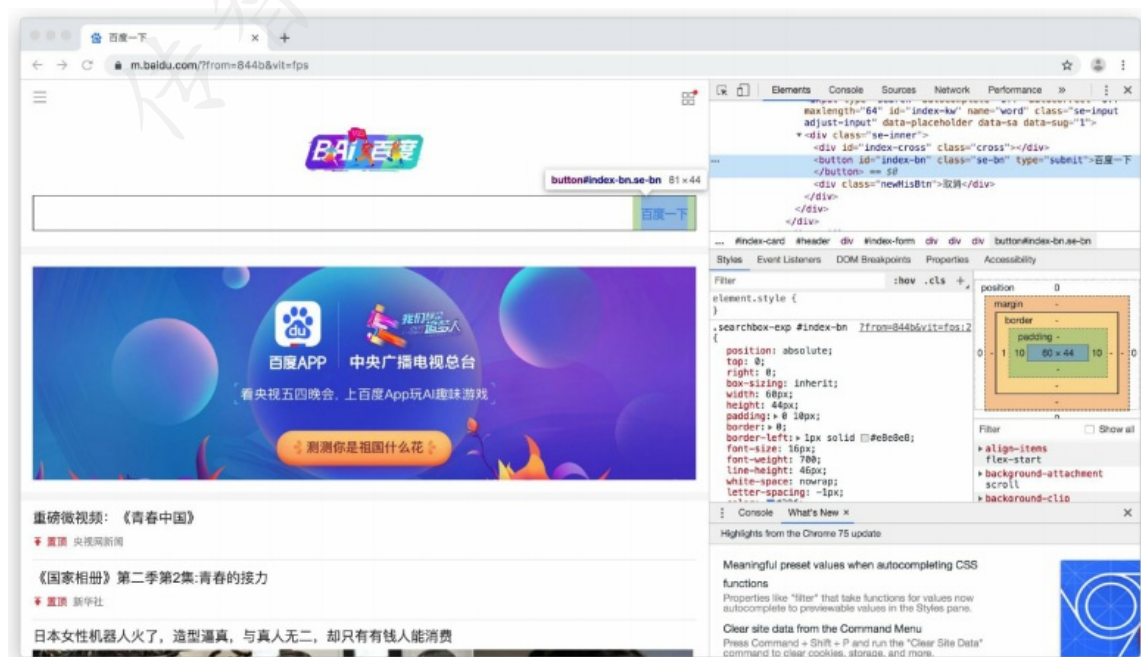


通过 **chrome** 浏览器查看手机的网页地址

1. 使用 genymotion 打开需要查看的 webview 界面，并全选地址

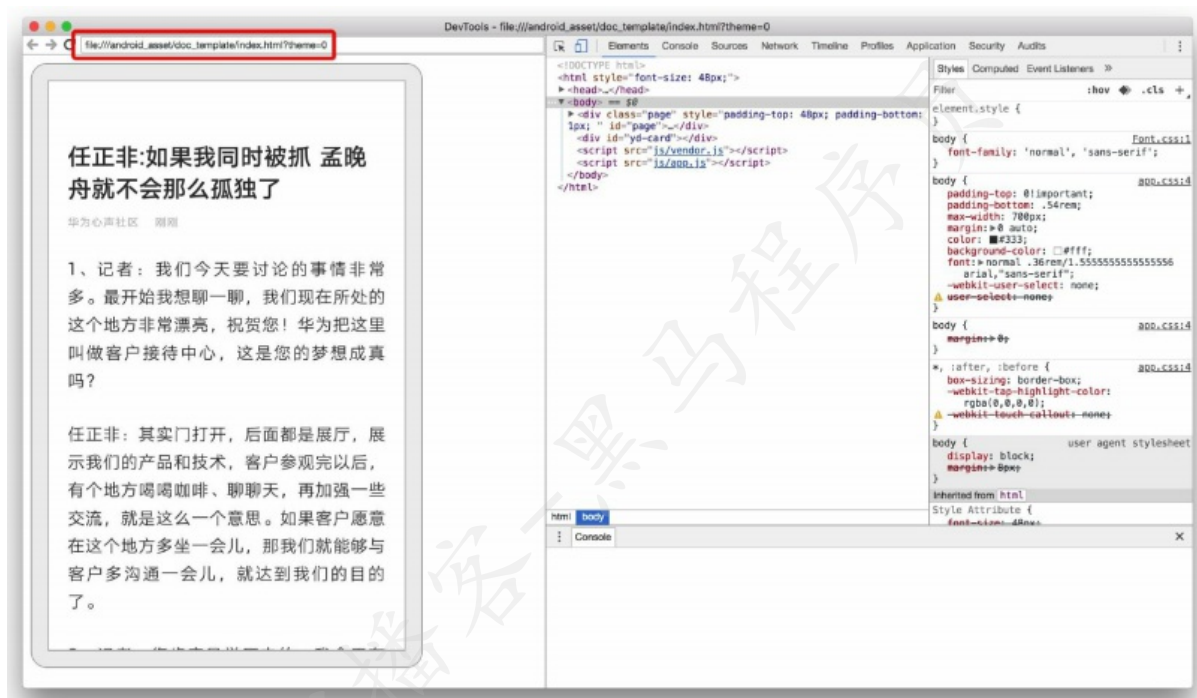


2. 将地址粘贴到 chrome 中进行打开，并使用 右键元素-检查 的形式进行查看



关于查看元素的注意点

- inspect 工具 和 android 版本有关
 - 工具是否有 "select an element..." 按钮取决于 android 版本，有些版本可能并没有 "select an element..." 的按钮，比如 android 5.1，如果真的需要对 android 5.1 需要查看，只能从代码中一个一个找。
- 能否使用chrome 直接查看手机地址与网页地址有关
 - 比如，网易新闻的新闻页面 是使用的 WebView，但开发人员是将整个网页下载到手机后，再通过下载在手机的地址进行加载。而复制的地址是手机的绝对路径，在电脑上是无法打开的。



2.2 实现webview自动化

- 前置代码，和之前相同（打开的包名和启动名是浏览器软件）
- 获取，driver的所有的上下文。
 - 得到，一个原生的app的字符串，还有各种webview的字符串。

```
contexts = driver.contexts
for i in contexts:
    print(i)
```

- 如:
- NATIVE_APP WEBVIEW_cn.goapk.market WEBVIEW_com.android.browser
- 通过，driver的switchto来切换上下文

```
# 告诉appium需要查找的是 com.android.browser程序的webview的内容
driver.switch_to.context("WEBVIEW_com.android.browser")
```

- 切换后可以使用selenium的方法进行元素定位
- 包括，点击已经输入文字等api都是相同的。

案例：

在浏览器应用中打开百度首页，并在搜索框中输入10086，再点击搜索。然后打开知乎首页。

核心代码

```
from appium import webdriver

desired_caps = dict()
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '5.1'
desired_caps['deviceName'] = '192.168.56.101:5555'
desired_caps['appPackage'] = 'com.android.browser'
desired_caps['appActivity'] = '.BrowserActivity'

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

# 打开百度
driver.find_element_by_id("com.android.browser:id/url").send_keys("www.baidu.com")
driver.press_keycode(66)

print(driver.contexts)

# 切换到网页环境
driver.switch_to.context("WEBVIEW_com.android.browser")

# 定位百度输入框并输入10086
driver.find_element_by_id("index-kw").send_keys("10086")
# 定位百度一下按钮并点击
driver.find_element_by_id("index-bn").click()

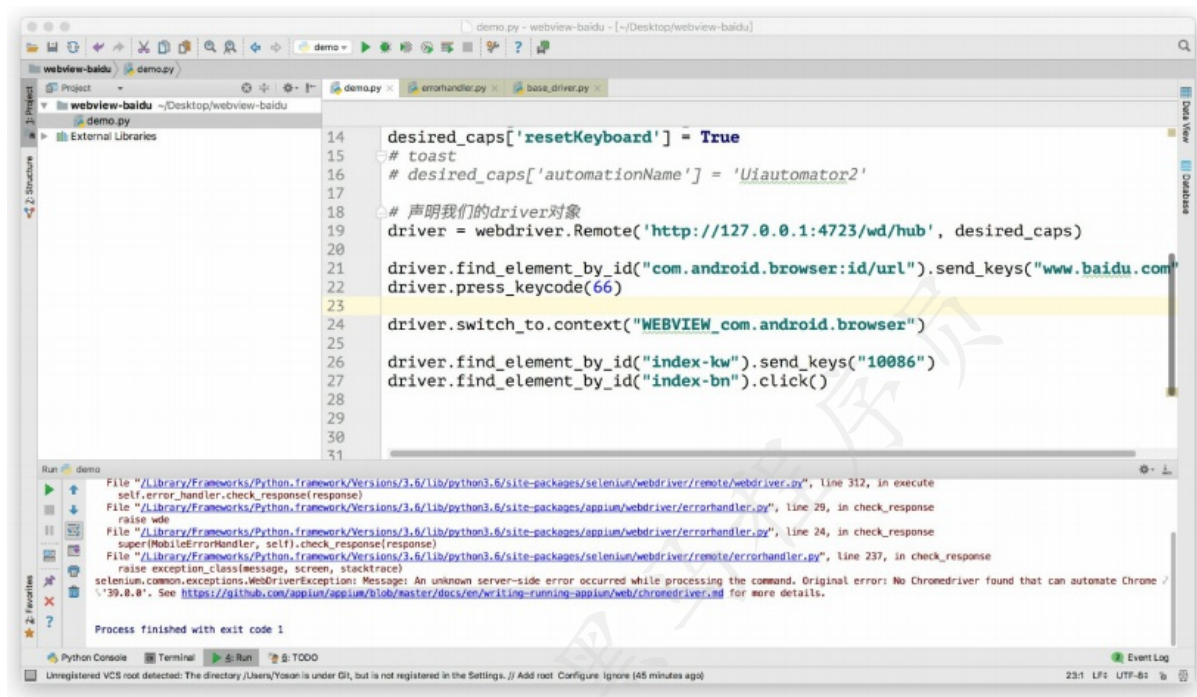
# 切换到原生环境
driver.switch_to.context("NATIVE_APP")

# 打开百度
driver.find_element_by_id("com.android.browser:id/url").send_keys("www.zhihu.com")
driver.press_keycode(66)
```

2.3 关于没有对应 chromedriver 的问题

根据错误信息下载对应的 **chromedriver**

错误为:



打开提示的网址:

<https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/web/chromedriver.md>

此处有chrome版本对应的chromedriver版本

需要下载chromedriver。下载哪个版本?

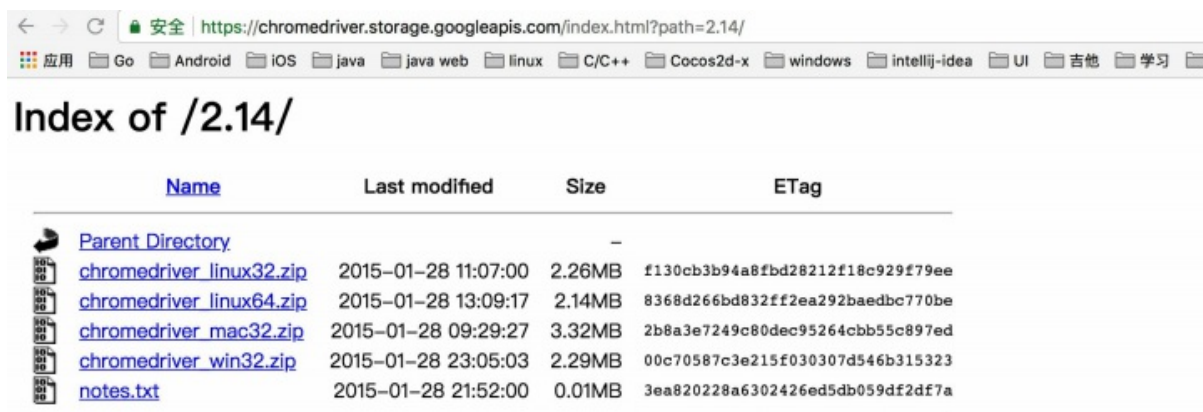
看自己手机的浏览器的版本, 在设置-应用-全部-android system webview



找到对应的下载地址：

2.22	49.0.2623.0	v2.22 (link)
2.21	46.0.2490.0	v2.21 (link)
2.20	43.0.2357.0	v2.20 (link)
2.19	43.0.2357.0	v2.19 (link)
2.18	43.0.2357.0	v2.18 (link)
2.17	42.0.2311.0	v2.17 (link)
2.16	42.0.2311.0	v2.16 (link)
2.15	40.0.2214.0	v2.15 (link)
2.14	39.0.2171.0	v2.14 (link)
2.13	38.0.2125.0	v2.13 (link)
2.12	36.0.1985.0	v2.12 (link)
2.11	36.0.1985.0	v2.11 (link)

通过link下载。

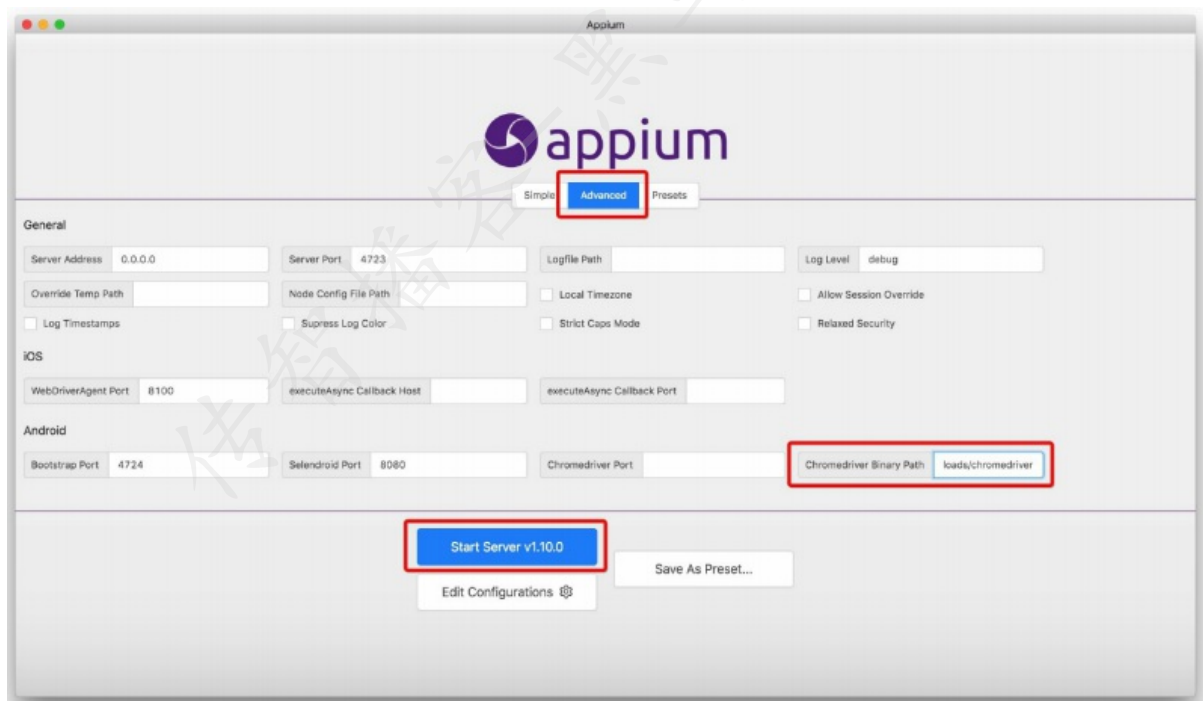


Name	Last modified	Size	ETag
Parent Directory		-	
chromedriver_linux32.zip	2015-01-28 11:07:00	2.26MB	f130cb3b94a8fbd28212f18c929f79ee
chromedriver_linux64.zip	2015-01-28 13:09:17	2.14MB	8368d266bd832ff2ea292baedbc770be
chromedriver_mac32.zip	2015-01-28 09:29:27	3.32MB	2b8a3e7249c80dec95264cbb55c897ed
chromedriver_win32.zip	2015-01-28 23:05:03	2.29MB	00c70587c3e215f030307d546b315323
notes.txt	2015-01-28 21:52:00	0.01MB	3ea820228a6302426ed5db059df2df7a

解压zip会得到一个chromedriver文件。

启动 appium 时加载 chromedriver

1. 点击 Advanced
2. 将 chromedriver 的路径输入到 chromedriver binary path 中
3. 点击 start server



3. Monkey

3.1 Monkey 简介和基本使用

环境

同 Android 环境

什么是 Monkey

顾名思义，Monkey就是猴子，Monkey测试，就像一只猴子，在电脑面前，乱敲键盘在测试。猴子什么都不懂，只知道乱敲

通过Monkey程序模拟用户触摸屏幕、滑动Trackball、按键等操作来对设备上的程序进行压力测试，检测程序多久的时间会发生异常

Monkey 用来做什么

Monkey 主要用于Android 的压力测试 自动的一个压力测试小工具，主要目的就是为了测试app 是否会Crash.

Monkey 程序介绍

- Monkey程序由Android系统自带，使用Java语言写成，在Android文件系统中的存放路径是：
/system/framework/monkey.jar;
- Monkey.jar程序是由一个名为“monkey”的Shell脚本来启动执行，shell脚本在Android文件系统中的存放路径是：/system/bin/monkey;
- Monkey 命令启动方式：
 - 可以通过PC机CMD窗口中执行: adb shell monkey {+命令参数} 来进行Monkey测试
 - 在PC上adb shell 进入Android系统，通过执行 monkey {+命令参数} 来进行Monkey 测试
 - 在Android机或者模拟器上直接执行monkey 命令，可以在Android机上安装Android终端模拟器（Terminal Emulator for Android）

Monkey 输出日志

```
adb shell monkey -p cn.goapk.market 100 > 路径/log.txt
```

3.2 Monkey 的参数

启动指定 app

- -p <允许的包名列表>

用此参数指定一个或多个包。指定包之后，monkey将只允许系统启动指定的app。如果指定包列表，monkey将允许系统启动设备中的所有app。

指定一个包：adb shell monkey -p cn.goapk.market 100

指定多个包：adb shell monkey -p fishjoy.control.menu -p cn.goapk.market 100

日志级别

- -v

用于指定反馈信息级别（信息级别就是日志的详细程度），总共分3个级别，分别对应的参数如下表所示：

Level 0 : `adb shell monkey -p cn.goapk.market -v 100` // 缺省值，仅提供启动提示、测试完成和最终结果等少量信息

Level 1 : `adb shell monkey -p cn.goapk.market -v -v 100` // 提供较为详细的日志，包括每个发送到Activity的事件信息

Level 2 : `adb shell monkey -p cn.goapk.market -v -v -v 100` // 最详细的日志，包括了测试中选中/未选中的Activity信息

随机数种子

- -s（随机数种子）

用于指定伪随机数生成器的seed值，如果seed相同，则两次Monkey测试所产生的事件序列也相同的。示例：

monkey测试1: `adb shell monkey -p cn.goapk.market -s 10 100`

monkey测试2: `adb shell monkey -p cn.goapk.market -s 10 100`

事件间隔时间

- --throttle <毫秒>

`cn.goapk.market --throttle 3000 100`

随机事件出现的百分比

`--pct-touch <percent>`

调整触摸事件的百分比(触摸事件是一个down-up事件，它发生在屏幕上的某单一位置)。

`--pct-motion <percent>`

调整动作事件的百分比(动作事件由屏幕上某处的一个down事件、一系列的伪随机事件和一个up事件组成)。

`--pct-pinchzoom <percent>`

缩放事件百分比

`--pct-trackball <percent>`

调整轨迹事件的百分比(轨迹事件由一个或几个随机的移动组成，有时还伴随有点击)。

`--pct-rotation <percent>`

屏幕旋转事件百分比

`--pct-nav <percent>`

调整“基本”导航事件的百分比(导航事件由来自方向输入 设备的up/down/left/right组成)。

`--pct-majornav <percent>`

调整“主要”导航事件的百分比(这些导航事件通常引发图形界面中的动作，如：回退按键、菜单按键)

`--pct-syskeys <percent>`

调整“系统”按键事件的百分比(这些按键通常被保留，由系统使用，如Home、Back、Start Call、End Call及音量控制键)。

`--pct-appswitch <percent>`

调整启动Activity的百分比。在随机间隔里，Monkey将执行一个startActivity()调用，作为最大程度覆盖包中全部Activity的一种方法。

`--pct-flip <percent>`

键盘翻转事件百分比

`--pct-anyevent <percent>`

调整其它类型事件的百分比。它包罗了所有其它类型的事件，如：按键、其它不常用的设备按钮、等等。

3.3 Monkey 日志分析

- 正常情况

如果Monkey测试顺利执行完成，在log的最后，会打印出当前执行事件的次数和所花费的时间：

// Monkey finished 代表执行完成

- 异常情况

Monkey 测试出现错误后，一般的分析步骤

1. 程序无响应的问题: 在日志中搜索“ANR”(可能仅仅是因为卡)
2. 崩溃问题: 在日志中搜索“Exception”(如果出现空指针，NullPointerException)肯定是有bug
Monkey 执行中断，在log最后也能看到当前执行次数

4. Android 真机调试

应用场景

在企业工作中，我们的自动化脚本更多的是使用真机进行测试。那么如何将脚本运行到真机中呢？

4.1 准备工作

1. 真机一台
2. 数据线一根
3. 打开USB调试的开关（进入开发者模式）

- i. 开发者模式如何进入，不同厂商是不一样的。
 - ii. 可能需要百度。
4. 在电脑上安装对应的驱动
 - i. 需要从厂商的官网下载，或者找一个类似360手机助手这种软件，自动下载。
5. 使用 `adb devices` 检查真机是否已经正常的连接

4.2 操作步骤

1. 新建一个 `python` 项目，将一下代码复制到项目的 `py` 文件中。

```
from appium import webdriver

import time

desired_caps = dict()
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'

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

time.sleep(2)

driver.quit()
```

2. 修改 `android` 系统版本 以及 待启动的 `app` 等参数。
3. 使用定位元素的形式对某个按钮进行点击或其他操作。(和之前的步骤都是一样的啦！)