

安卓自动化测试Java版

Author @

AnthonyGao

核心框架

- appium <http://appium.io/>
- testng <https://testng.org/doc/documentation-main.html>
- java-client
- app framework

Android 要素

- adb
- uiautomatorviewer

环境前置条件

- JDK
- JDK
- JDK
- 如果这个都不会安装配置，那就放弃吧～

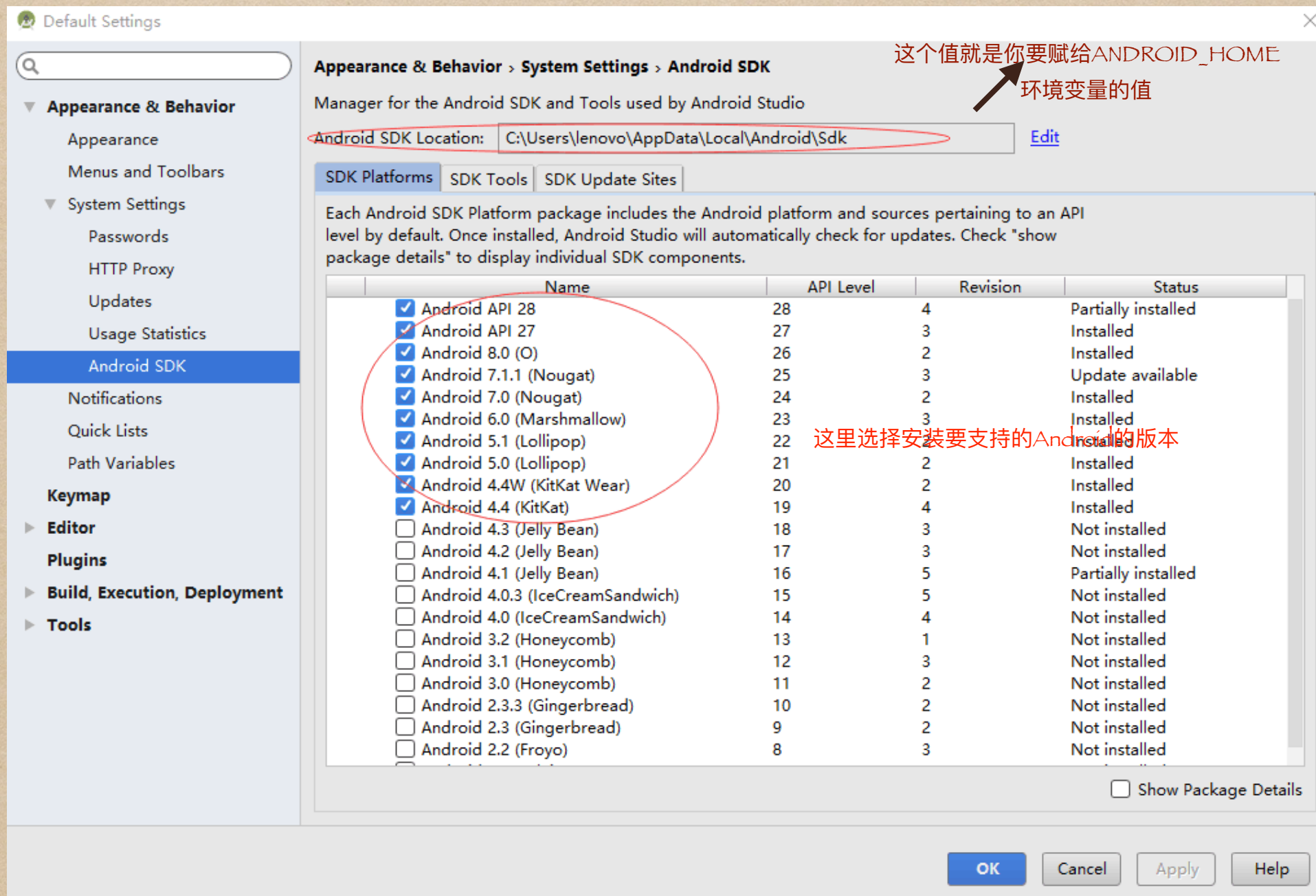
windows

android sdk快速配置

这里有个小技巧，可以直接选择安装android studio，然后按图一步步完成sdk环境配置，studio安装完成后打开点击下图标红位置

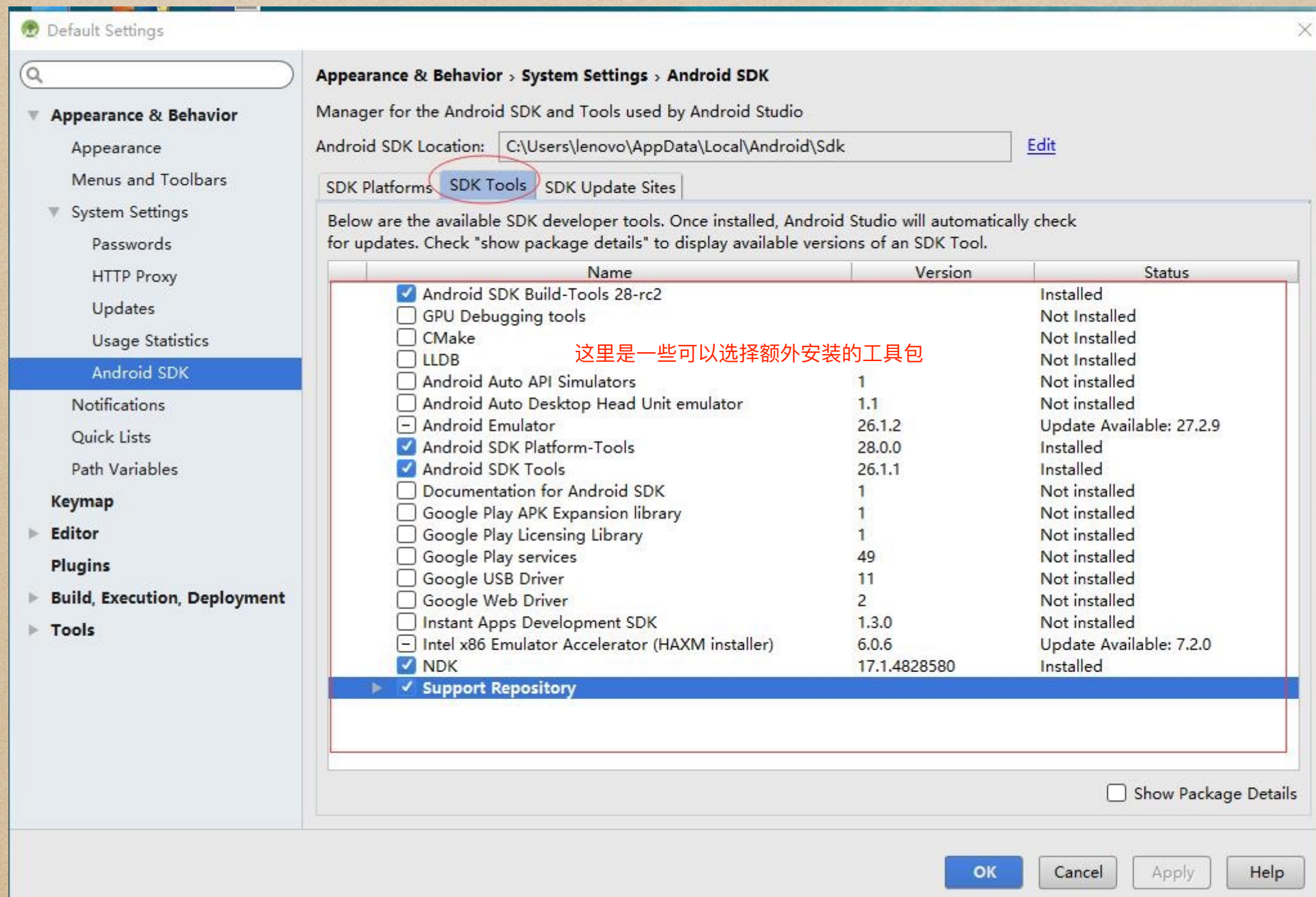


windows android sdk快速配置



windows

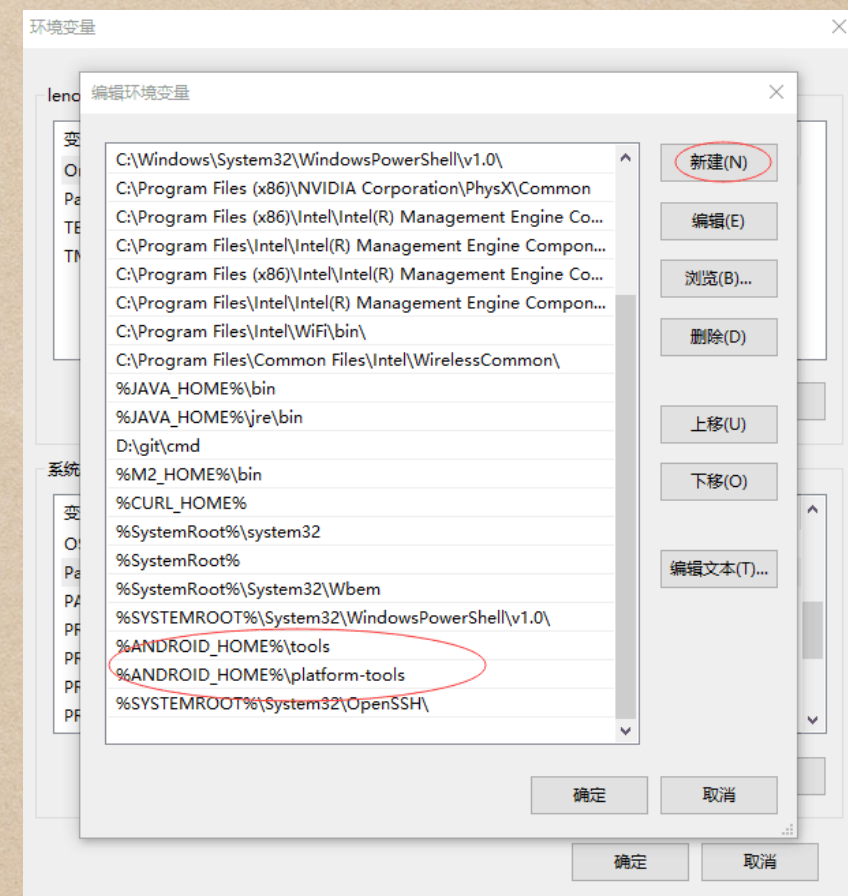
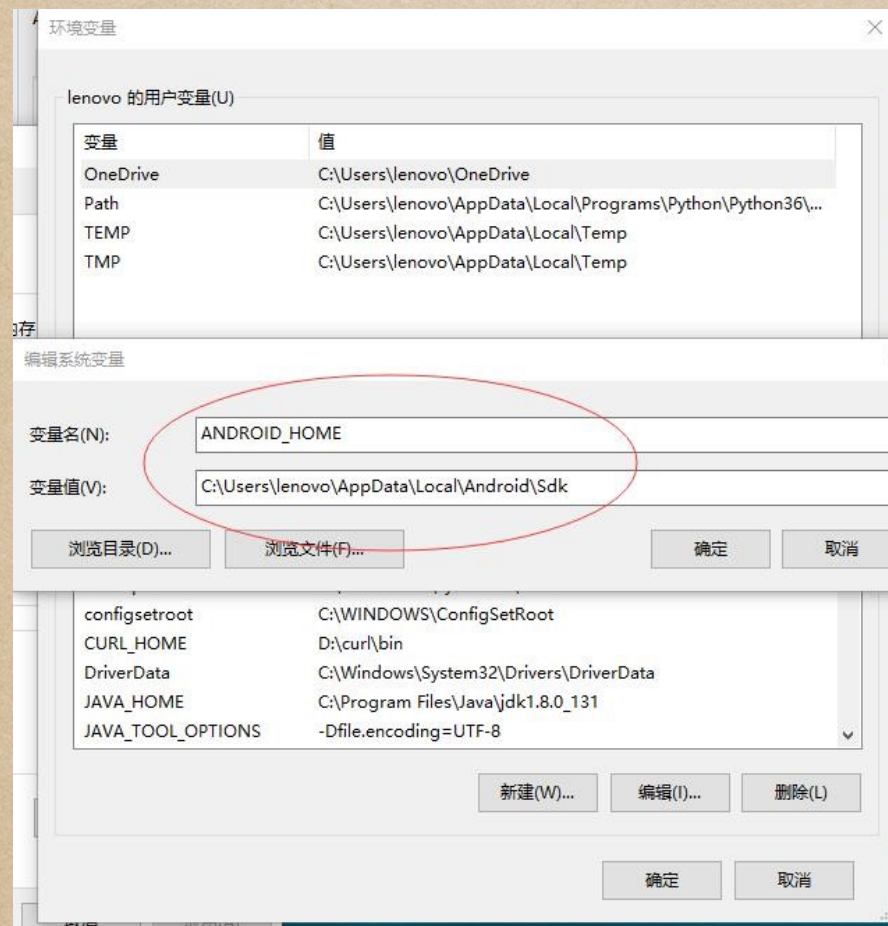
android sdk快速配置



windows android sdk配置

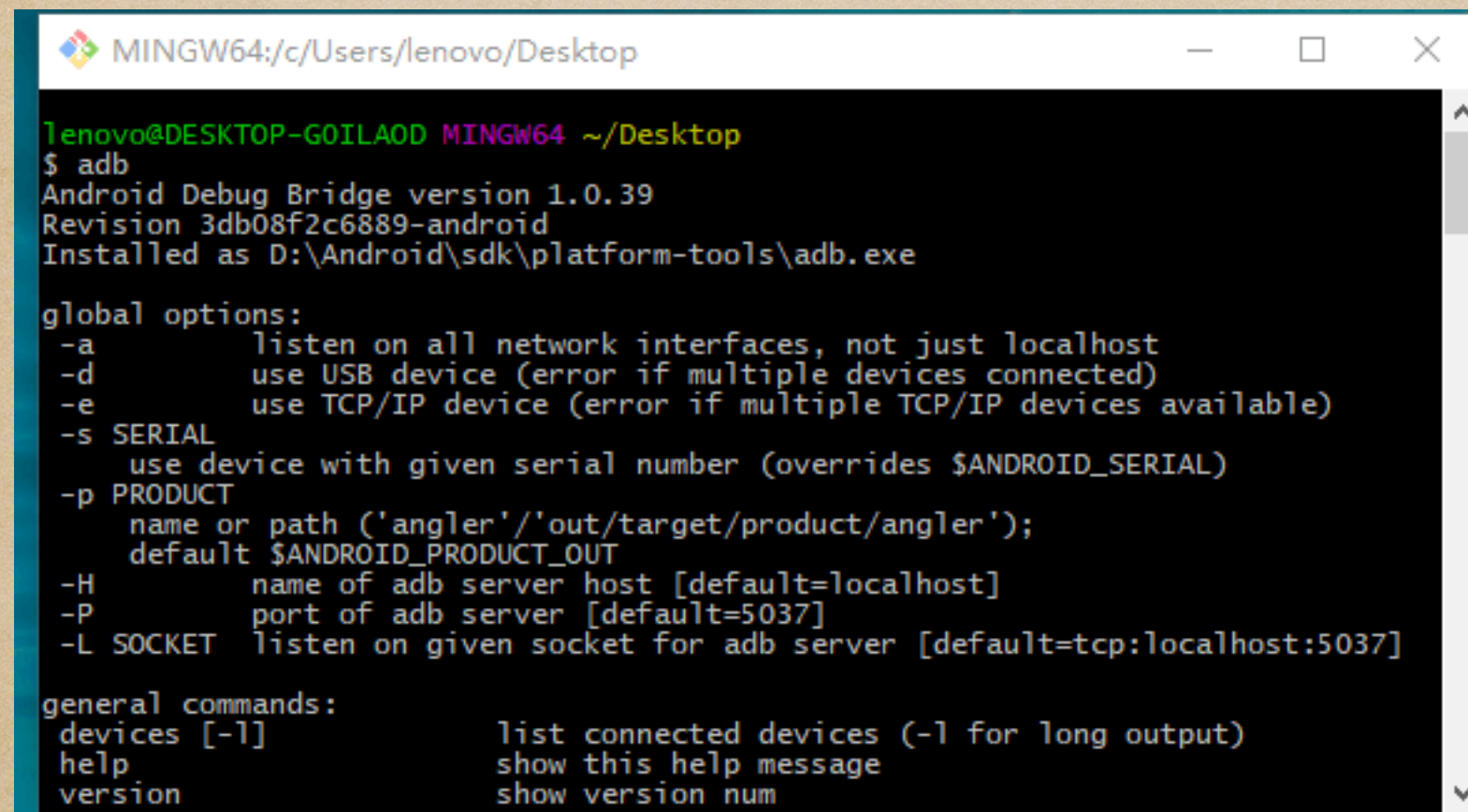
开始配置环境变量

打开计算机属性——高级系统设置——环境变量



windows android sdk配置

在命令行输入adb出现如下图所示就说明安装配置完成了

A screenshot of a Windows command prompt window titled "MINGW64:/c/Users/lenovo/Desktop". The prompt shows the user "lenovo@DESKTOP-G0ILAOD" in a "MINGW64" environment at the path "~/Desktop". The user has entered the command "\$ adb", and the output displays the Android Debug Bridge (adb) version information and a list of global options and general commands. The global options include flags for listening on all network interfaces, using USB or TCP/IP devices, specifying a serial number, product, server host, port, and socket. The general commands include listing connected devices, showing help, and showing the version number.

```
lenovo@DESKTOP-G0ILAOD MINGW64 ~/Desktop
$ adb
Android Debug Bridge version 1.0.39
Revision 3db08f2c6889-android
Installed as D:\Android\sdk\platform-tools\adb.exe

global options:
-a          listen on all network interfaces, not just localhost
-d          use USB device (error if multiple devices connected)
-e          use TCP/IP device (error if multiple TCP/IP devices available)
-s SERIAL   use device with given serial number (overrides $ANDROID_SERIAL)
-p PRODUCT  name or path ('angler'/'out/target/product/angler');
            default $ANDROID_PRODUCT_OUT
-H          name of adb server host [default=localhost]
-P          port of adb server [default=5037]
-L SOCKET   listen on given socket for adb server [default=tcp:localhost:5037]

general commands:
devices [-l]    list connected devices (-l for long output)
help           show this help message
version        show version num
```


mac android sdk快速配置

1. 下载Android studio, 所有操作和window安装相似, 环境变量配置参照如下形式:
2. 在terminal 中输入 touch .bash_profile
3. 输入 open -e .bash_profile
4. 在打开的文本中输入

```
export ANDROID_HOME=/Users/nidemingzi/Library/Android/sdk  
export PATH=$PATH:/Users/nidemingzi/Library/Android/sdk/tools:/Users/nidemingzi/Library/Android/sdk/tools/bin:/Users/nidemingzi/Library/Android/sdk/platform-tools
```

保存退出文本编辑 并且关闭窗口

重新打开一个terminal, 输入adb version,得到如下信息说明Android环境已经搭建好了

```
$ANDROID_HOME/tools/bin/adb version  
[anthony:~ anthony$ adb version  
Android Debug Bridge version 1.0.39  
Revision 3db08f2c6889-android  
Installed as /Users/anthony/Library/Android/sdk/platform-tools/adb  
anthony:~ anthony$ ]
```


windows 安装appium

- download (<https://github.com/appium/appium-desktop/releases>) select appiumForWindows.zip
- install appium.exe

mac os 安装appium

- download (<https://github.com/appium/appium-desktop/releases>) select appium.dmg
- install appium.dmg with default settings

里程碑

到此，安卓自动化测试所需的环境就都已经搭建完毕了，接下来我们将开启有码之门。

回顾下环境搭建的关键点：

JDK

Android SDK

Appium client

码前干货积累

在开启有码之门之前，我们有必要了解些adb 命令，标红一定要熟练运用

① adb devices ——— 设备，获取设备列表及设备状态

② adb logcat ——— 打印Android的系统日志

③ adb install ——— 安装apk

④ adb install -r ——— 覆盖安装

⑤ adb push ——— 推送本地文件至Android设备

码前干货积累

用usb线连接你的安卓手机到你的电脑，注意打开Android手机的开发者模式，不知道怎么打开的请自行百度谷歌

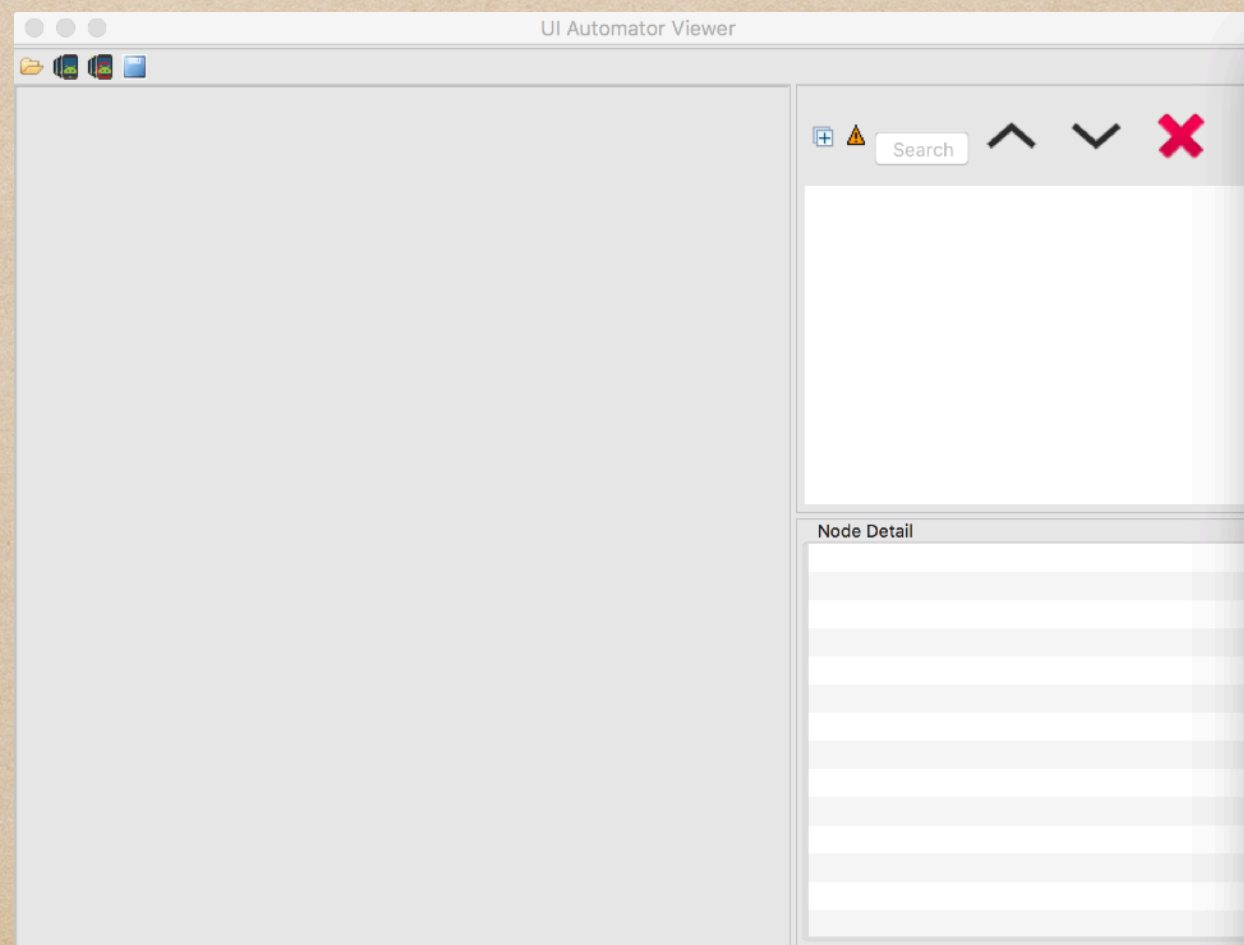
在你的命令行敲入adb devices，标红的就是连接到你电脑的安卓手机的设备识别码

```
anthony:~ anthony$ adb devices
List of devices attached
Z5HUS4GYU4WS65EU    device
anthony:~ anthony$
```


码前干货积累

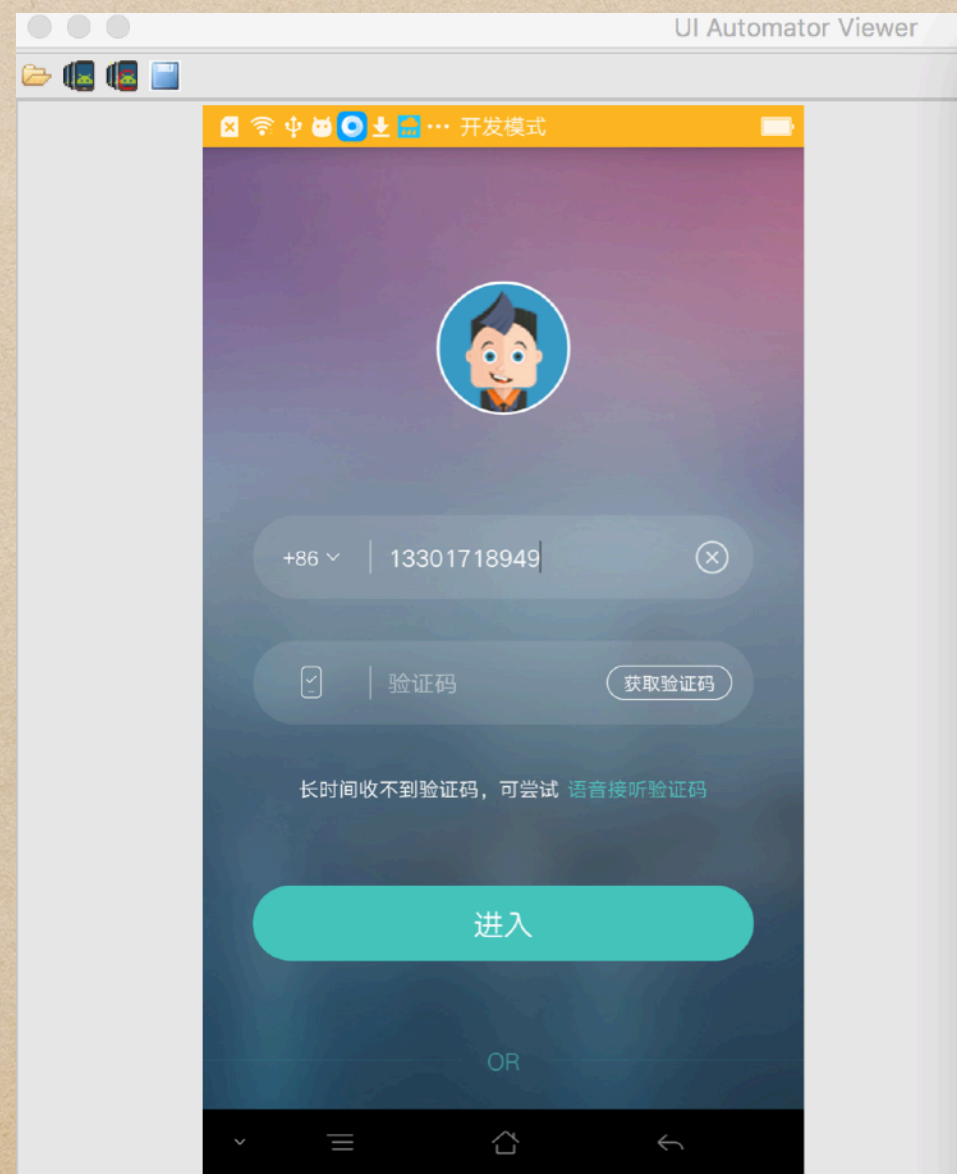
熟悉你的智能识别器：uiautomatorviewer

在命令行敲入：uiautomatorviewer，然后你将看到以下一幕：



码前干货积累

点击上图的两个小安卓图标的任意一个，你将会看到你的手机屏幕被获取了



码前干货积累

最后我们再来熟悉下appium的一些基本语法，这里不一一介绍了，贴出官方文档链接，自己看吧

<http://appium.io/docs/en/writing-running-appium/caps/>

接下来我们真的要开始打码了~

Android自动化测试初码

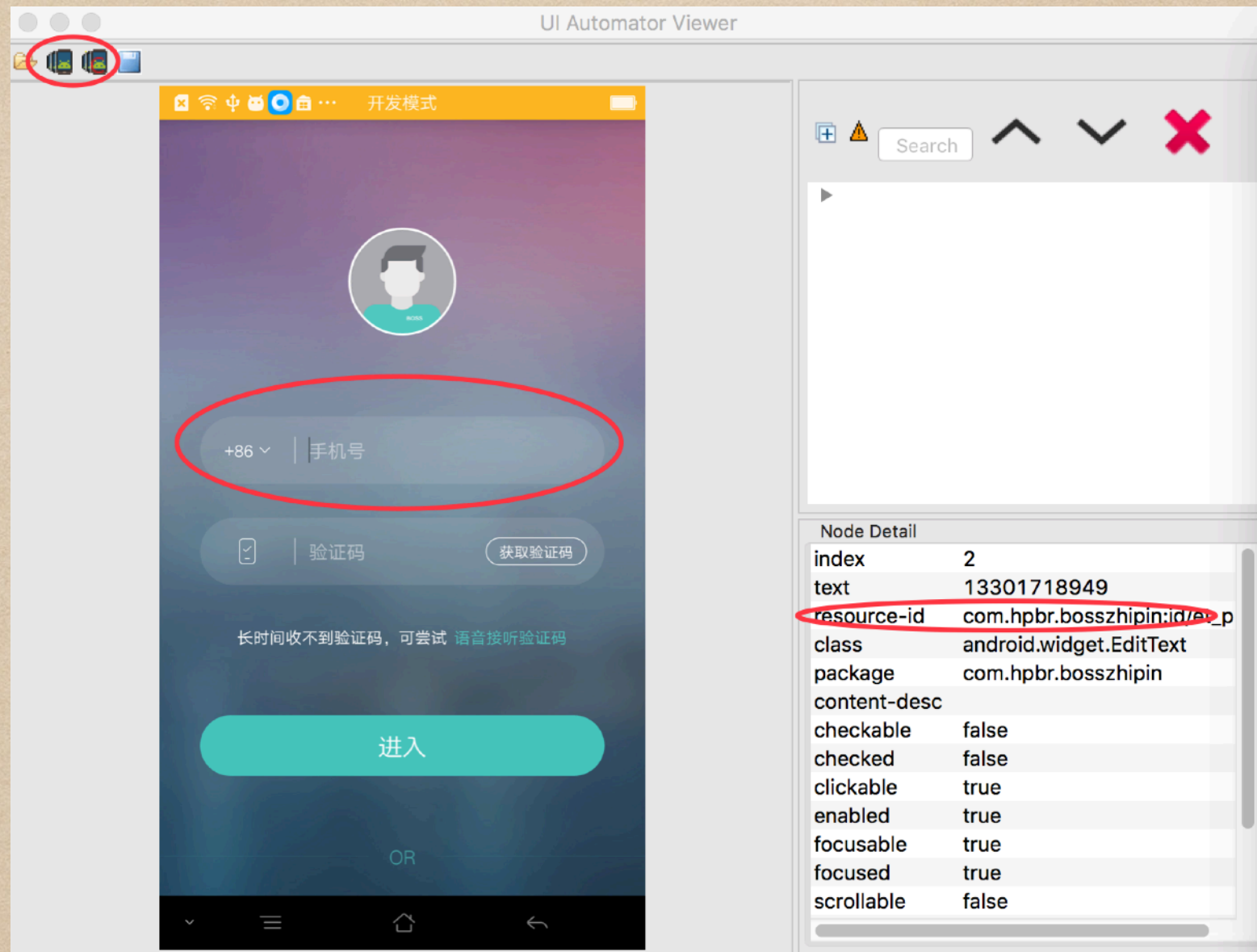
- 首先我们来想一下用例
- 打开boss直聘app
- 在用户名输入框中输入手机号

这里首先要明确一个思想，不接受反驳：所有UI的自动化测试都是以界面元素识别作为驱动的

我们想一下这个场景，其实这个过程中就涉及到一个元素：boss直聘登录输入框，

那么我们打开智能识别器uiautomatorviewer，android 机器打开boss直聘app，然后我们就可以看到如下图所示信息

UI automator viewer



Android自动化测试初码

分析完了，接着就要撸码了，想象还真是有点激动，但是我们还是要先稳住，接下来请看如何一步步创建自动化测试项目

首先创建一个maven项目

New Maven project
Specify Archetype parameters

Group Id:

Artifact Id:

Version:

Package:

Properties available from archetype:

Name	Value

Advanced

< Back Next > Cancel Finish

然后修改pom.xml如下

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5
6   <groupId>com.anthony.test</groupId>
7   <artifactId>training</artifactId>
8   <version>1.0.0-SNAPSHOT</version>
9   <packaging>jar</packaging>
10
11   <name>training</name>
12   <url>http://maven.apache.org</url>
13
14   <properties>
15     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
16   </properties>
17
18   <dependencies>
19     <dependency>
20       <groupId>io.appium</groupId>
21       <artifactId>java-client</artifactId>
22       <version>6.0.0</version>
23     </dependency>
24
25     <dependency>
26       <groupId>org.testng</groupId>
27       <artifactId>testng</artifactId>
28       <version>6.9.10</version>
29     </dependency>
30
31   </dependencies>
32 </project>
33
```


终于开启Boss有码

项目基本配置完成了，我们接下来在test.training包中新建一个BossTest测试类，并完成编码

```
/**
 * @author anthony
 * @date 2018年6月5日
 * @updateTime 上午11:10:26
 */
public class BossTest {

    public static AppiumDriver<WebElement> idriver;

    @BeforeTest

    public void setPreCondition() throws MalformedURLException {
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability("platformName", "Android");
        capabilities.setCapability("deviceName", "OPPO R7s");
        capabilities.setCapability("platformVersion", "4.4.4");
        capabilities.setCapability("appPackage", "com.hpbr.bosszhipin");
        capabilities.setCapability("appActivity",
            "com.hpbr.bosszhipin.module.launcher.WelcomeActivity");
        capabilities.setCapability("unicodeKeyboard", "True");
        capabilities.setCapability("resetKeyboard", "True");
        capabilities.setCapability("noReset", true);
        idriver = new AndroidDriver<WebElement>(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);
    }

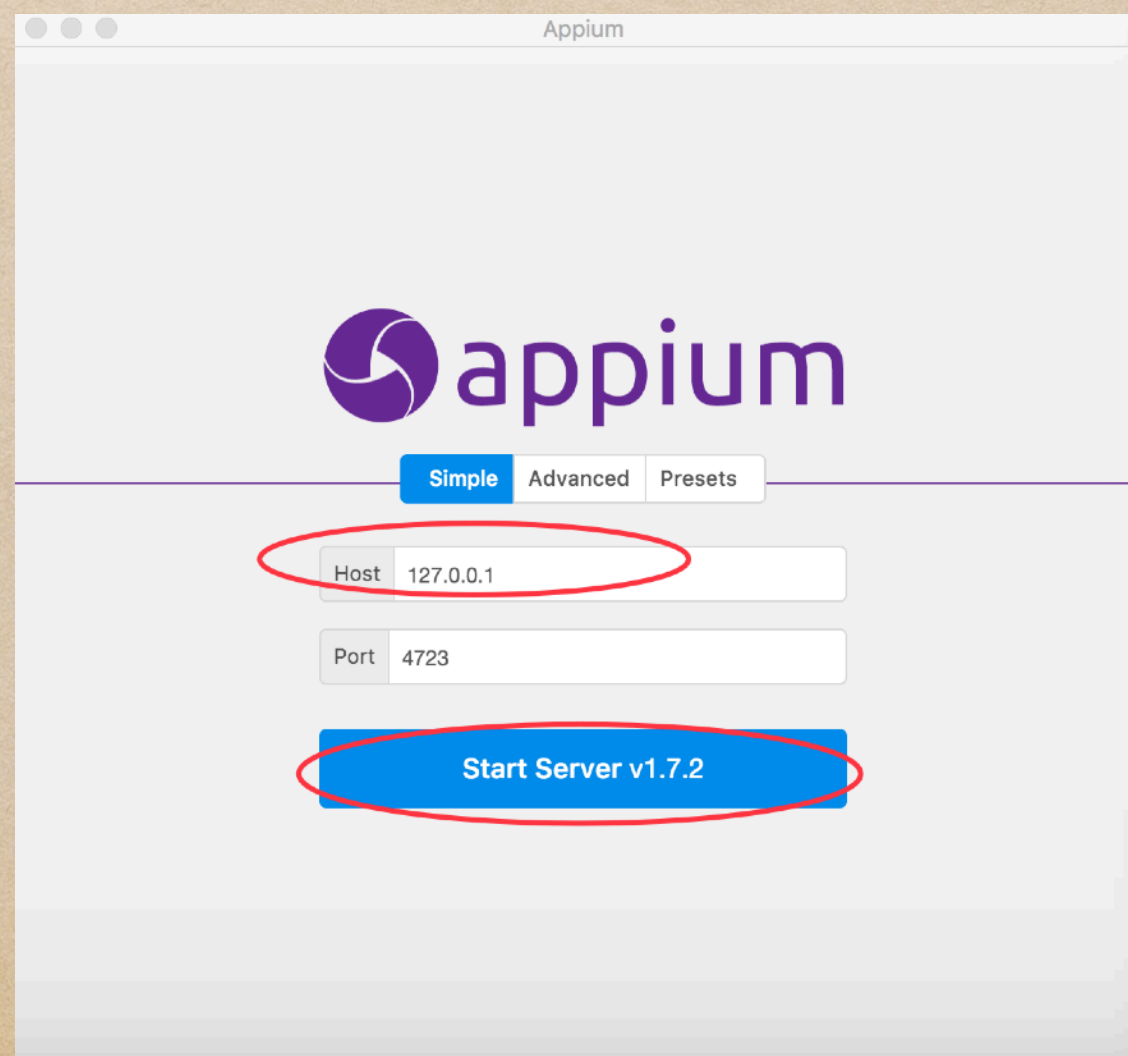
    @Test

    public void loginTest() throws InterruptedException {
        Thread.sleep(3000);
        idriver.findElement(By.id("com.hpbr.bosszhipin:id/et_phone")).sendKeys("18888888888");
    }
}
```


Android自动化测试初次运行

编码已经完成了，那我们要怎么运行刚刚完成的用例呢？接下来就是见证奇迹的时刻，但是在奇迹发生之前我们还需要再做一些准备：

还记得之前我们安装的appium吧，启动它，并且完成配置：



Android自动化测试初次运行

appium server启动好后，回到项目工程，run as testNG，然后你就可以观看回放结果了，你将会看到boss直聘app被打开，然后电话框内被输入了18888888888

手机浏览器也可以有码

看完了app有码，很好奇移动端的wap是不是也可以有码呢？答案是：肯定的，我们以android下的谷歌浏览器为例访问haosou

```
/**
 * @author anthony
 * @date 2018年6月21日
 * @updateTime 下午1:39:33
 */
public class H5Test {

    AppiumDriver<WebElement> driver;
    @BeforeTest
    public void startTest() throws MalformedURLException{
        DesiredCapabilities desiredCapabilities = new DesiredCapabilities();
        desiredCapabilities.setCapability(MobileCapabilityType.BROWSER_NAME, "Chrome");
        desiredCapabilities.setCapability(MobileCapabilityType.PLATFORM_NAME, "Android");
        desiredCapabilities.setCapability(MobileCapabilityType.PLATFORM_VERSION, "4.4.4");
        desiredCapabilities.setCapability(MobileCapabilityType.DEVICE_NAME, "OPPO R7s");
        desiredCapabilities.setCapability("appPackage", "com.android.chrome");
        desiredCapabilities.setCapability("appActivity", "com.android.chromecom.google.apps.chrome.Main");
        driver = new AndroidDriver<WebElement>(new URL("http://127.0.0.1:4723/wd/hub"),desiredCapabilities);
    }

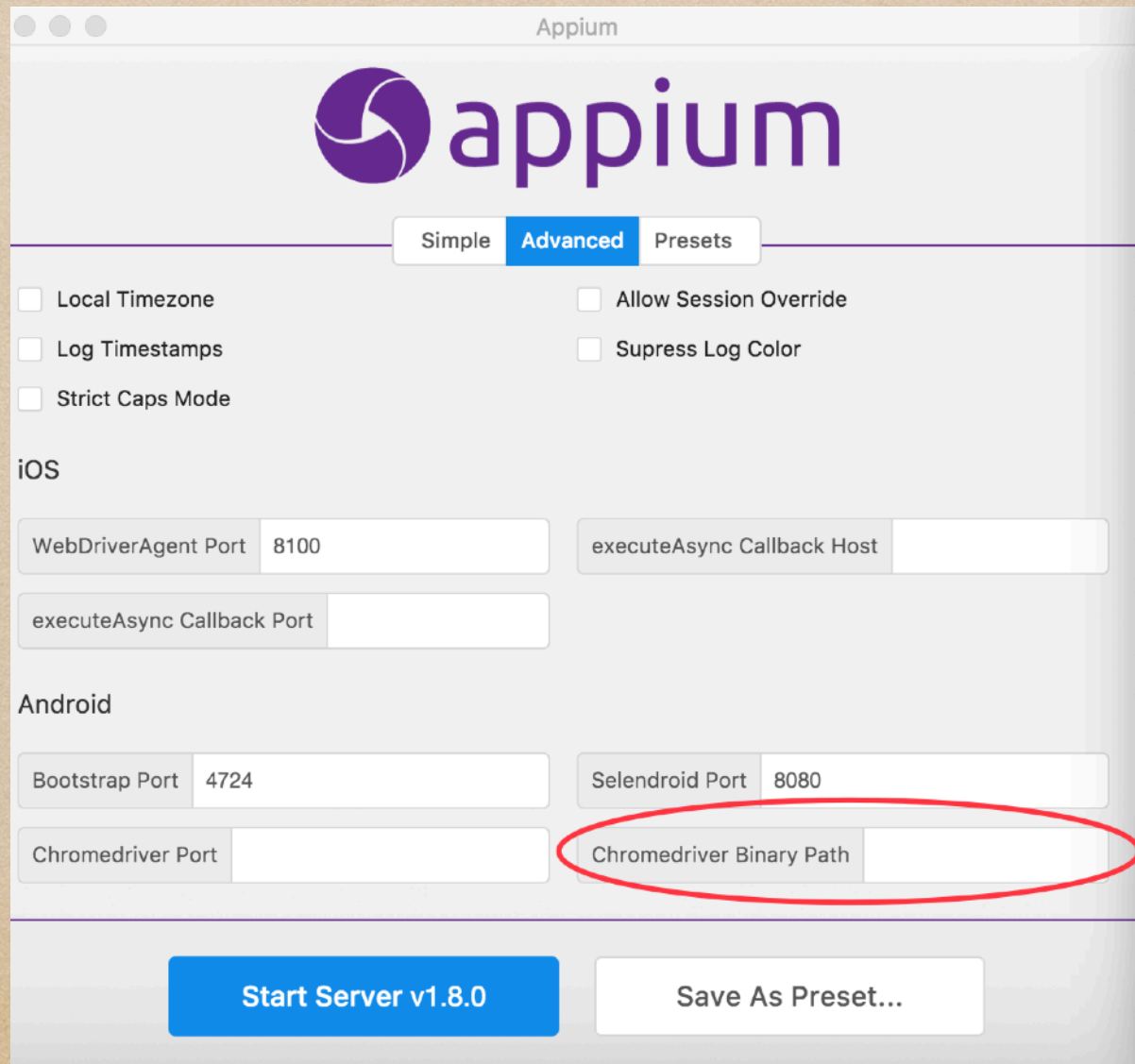
    @Test
    public void doTest() throws InterruptedException{
        driver.get("http://www.haosou.com/");
        driver.findElement(By.id("q")).sendKeys("Java");
        driver.findElement(By.className("search-btn")).click();
    }

    @AfterTest
    public void endTest(){
        driver.quit();
    }
}
```


Android浏览器运行

这里需要额外的一个driver: chromedriver, 访问这个链接吧下载适合你的那一款, 然后在appium启动时选择高级:

<http://appium.io/docs/en/writing-running-appium/web/chromedriver/>

The image shows the Appium web interface in 'Advanced' mode. It features a header with the Appium logo and three tabs: 'Simple', 'Advanced' (which is selected), and 'Presets'. Below the tabs, there are several configuration sections. The 'iOS' section includes checkboxes for 'Local Timezone', 'Log Timestamps', 'Strict Caps Mode', 'Allow Session Override', and 'Supress Log Color'. The 'Android' section contains input fields for 'WebDriverAgent Port' (8100), 'executeAsync Callback Host', 'executeAsync Callback Port', 'Bootstrap Port' (4724), 'Selendroid Port' (8080), 'Chromedriver Port', and 'Chromedriver Binary Path'. The 'Chromedriver Binary Path' field is circled in red. At the bottom, there are two buttons: 'Start Server v1.8.0' and 'Save As Preset...'.

Appium

Simple Advanced Presets

☐ Local Timezone ☐ Allow Session Override

☐ Log Timestamps ☐ Supress Log Color

☐ Strict Caps Mode

iOS

WebDriverAgent Port 8100 executeAsync Callback Host

executeAsync Callback Port

Android

Bootstrap Port 4724 Selendroid Port 8080

Chromedriver Port Chromedriver Binary Path

Start Server v1.8.0 Save As Preset...

填入你的chromedriver的真实路径

appium server启动好后，回到项目工程，run as testNG，然后你就可以观看回放结果了，你可以看到你手机中的谷歌浏览器被打开，然后自动访问了haosou网站，然后在输入框中输入了java并完成了搜索

Android自动化测试进阶

简易封装

之前有记录了关于怎样搭建安卓自动化的测试环境，也记录了一些小技巧，也给出实例怎样真正开始动手写一个自动化测试代码，但是真正的自动化测试项目中，只会那些还是远远不够的，细想一下，如果每一个步骤要有 `driver.findElement` 只论可读性的话就该够你喝一壶的了，更别谈维护了。这时脑子上灯突然闪了下，貌似你好像有了些灵感，那些重复的语句，可以封装吗？那么我们就先从 `findElement` 的方法开始我们的框架设计之旅。

我们首先来看看原生的appium找元素的方法,以id为例：`driver.findElement(By.id(" "))`,暂时抛开 `driver`，`findElement` 方法中包含了2个部分 `By` 和 `id`（元素的属性）`By` 是原生的关键字，这里我们也先不动它，我们先从 `id` 入手，其实不光 `id`，有很多通过元素属性找元素的方法，比如：`id`,`name`,`cssSelector`,`xpath`,`linkText`,`tagName` 等等等。。这里我们就可以做文章了，比如我们实例化一个 `By locator` 的方法，把有可能用到的元素属性都列在里面，根据用户提供的元素属性的不同调用不同的方法，元素属性包含2个部分：属性名称，属性 `value` 中间可用 `'':` 隔开。

1. 如果用户的输入没有 `''`，那么默认调用 `By.id()` 方法
2. 如果用户的输入包含 `''`，那么根据实际情况来判断，通过 `''` 把元素用户输入分开，通过 `''` 左边分离出来的字符串来判断具体的调用方法。通过 `''` 右边分离出来的字符串来传入实际的属性 `value`

具体实现如下：

Android自动化测试进阶

简易封装

之前有记录了关于怎样搭建安卓自动化的测试环境，也记录了一些小技巧，也给出实例怎样真正开始动手写一个自动化测试代码，但是真正的自动化测试项目中，只会那些还是远远不够的，细想一下，如果每一个步骤要有 `driver.findElement` 只论可读性的话就该够你喝一壶的了，更别谈维护了。这时脑子上灯突然闪了下，貌似你好像有了些灵感，那些重复的语句，可以封装吗？那么我们就先从 `findElement` 的方法开始我们的框架设计之旅。

我们首先来看看原生的appium找元素的方法,以id为例：`driver.findElement(By.id(" "))`,暂时抛开 `driver`，`findElement` 方法中包含了2个部分 `By` 和 `id`（元素的属性）`By` 是原生的关键字，这里我们也先不动它，我们先从 `id` 入手，其实不光 `id`，有很多通过元素属性找元素的方法，这里我们只介绍最容易用的两种方法 `ByID`，和 `ByXpath`，比如我们实例化一个 `By locator` 的方法，把有可能用到的元素属性都列在里面，根据用户提供的元素属性的不同调用不同的方法，元素属性包含2个部分：属性名称，属性value中间可用 `'` `:` `'` 隔开。

1. 如果用户的输入没有 `'` `:` `'`，那么默认调用 `By.id()` 方法
2. 如果用户的输入包含 `'` `:` `'`，那么根据实际情况来判断，通过 `'` `:` `'` 把元素用户输入分开，通过 `'` `:` `'` 左边分离出来的字符串来判断具体的调用方法。通过 `'` `:` `'` 右边分离出来的字符串来传入实际的属性value

具体实现如下：

Android自动化测试进阶

简易封装MyBy

```
/**
 * @author anthony
 *
 */
public class MyBy extends By {
    @Override
    public List<WebElement> findElements(SearchContext context) {
        // TODO Auto-generated method stub
        return null;
    }

    public static By locator(String locator) {
        if (!locator.contains("xpath")) {
            return By.id(locator);
        } else if (locator.contains("xpath")) {
            String[] lArr = locator.split(":");
            String by = lArr[0];
            String using = locator.substring(by.length() + 1);
            return By.xpath(using);
        } else {
            print(" Element " + locator + "cannot be found.. ");
            throw new IllegalArgumentException("Cannot find elements when name text is null.");
        }
    }
}
```


Android自动化测试进阶

简易封装MyWebElement

定义完我们自己的MyBy后，那我们怎么用呢？如何才能把找元素的方法封装完毕呢？接下来我们需要定义自己的MyWebElement并且实现原生WebElement,添加属于自己的getElement方法来调用MyBy的找元素方法

```
public WebElement getWebElement() {  
    driver.manage().timeouts().implicitlyWait(IMPLICITLY_WAIT_TIME, TimeUnit.SECONDS);  
    return driver.findElement(MyBy.locator(locator));  
}  
  
public MyWebElement(String locator) {  
    driver.manage().timeouts().implicitlyWait(IMPLICITLY_WAIT_TIME, TimeUnit.SECONDS);  
    this.locator = locator;  
}
```

同时重新实现WebElement自身的必要函数，比如click, submit.....等，详情见下文：

Android自动化测试进阶

简易封装BasePage

定义好了我们自己的MyWebElement后，我们是不是已经大体猜到了此次框架设计的小心思，是不是自动化测试中一个经常用的词要脱口而出了？对，就是 对象库设计模式 定义自己的BasePage，封装一些最常用的公用方法如，打开app，关闭app等等同时以对象库的方式来存储页面元素从而实现页面的元素的高效管理

Android自动化测试进阶

简易封装BasePage

```
public class BasePage {  
    public static AppiumDriver<WebElement> idriver;  
  
    public static WebDriver driver;  
    public static File app;  
  
    /**  
     * Following functions are used to hide driver get MyWebElement  
     */  
    public static MyWebElement myWebelement(WebElement webElement) {  
        return new MyWebElement(webElement);  
    }  
  
    public static MyWebElement myWebelement(String locator) {  
        return new MyWebElement(locator);  
    }  
}
```


Android自动化测试进阶

简易封装BasePage

```
/**
 * 开启app
 *
 * @param browser
 * @throws MalformedURLException
 */
public static void openAndroidApp(String deviceName, String osVersion, String appPackage, String appActivity)
    throws MalformedURLException {
    DesiredCapabilities capabilities = new DesiredCapabilities();
    capabilities.setCapability("platformName", "Android");
    capabilities.setCapability("deviceName", deviceName);
    capabilities.setCapability("platformVersion", osVersion);
    capabilities.setCapability("appPackage", appPackage);
    capabilities.setCapability("appActivity", appActivity);
    capabilities.setCapability("unicodeKeyboard", "True");
    capabilities.setCapability("resetKeyboard", "True");
    capabilities.setCapability("noReset", true);
    idriver = new AndroidDriver<WebElement>(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);
    try {
        Thread.sleep(3000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```


Android自动化测试进阶

简易封装BasePage

```
/**
 * install app
 */
public static void installAndroidApp(String appPath) {
    idriver.installApp(appPath);
}

/**
 * remove app
 */
public static void removeApp(String bundleId) {
    idriver.removeApp(bundleId);
}

/**
 * 退出app
 */
public static void exitApp() {
    idriver.quit();
}
```


Android自动化测试进阶

简易封装里程碑

至此，简易测试框架就基本搭建完成了，核心三要素是什么？

MyBy

MyWebElement

BasePage

简单的测试框架已经完成，那如何才能使用起来呢？写用例的方式也还和以前一样么？

亮点又在哪里呢？接下来我们进入自动化测试进阶章节.....

Android自动化测试进阶 之描述性编程

简易的框架已经搭建完成，这里就可以公布接下来我们将采取何种方式进行自动化测试开发了

对象库+描述性编程模式

1. 元素管理使用对象库模式
2. 测试用例编写使用描述性编程

安卓自动化测试开发之 对象库+描述性编程

把app的每一个页面当作一个对象库，页面上的所有元素都定义在该页面中，实现对象库管理

```
package com.anthony.test.training.page;
```

```
import com.appium.baseapp.BasePage;
```

```
import com.appium.baselibs.MyWebElement;
```

```
public class BossLoginPage extends BasePage{
```

```
    //在BossLoginPage对象库中定义对象手机输入框MobileInput
```

```
    public static MyWebElement mobileInput = myWebelement("com.hpbr.bosszhipin:id/et_phone");
```

```
}
```


Android自动化测试 之描述性编程

测试用例使用描述性编程，所有的测试步骤都是：

对象库+对象+action的模式，浅显易懂

```
/**
 * 登录boss直聘
 * @throws InterruptedException
 */
@Test
public void loginTest() throws InterruptedException{
    BossLoginPage.mobileInput.sendKeys("18888888888");
}
```


Android自动化测试 之testng控制执行

花了很长的篇幅讲解了自动化测试的初打码，测试简易框架开发，基于框架的测试用例开发，接下来我们再来聊聊测试执行，本人推荐使用TestNG作为测试用例执行的控制器，运行也特别简单，直接选中该文件选择run as testng就行

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Training">
  <test name="Training" preserve_order="true">
    <classes>
      <class name="com.anthony.test.training.BossLoginTest" />
    </classes>
  </test>
</suite>
```


以上就是安卓自动化测试的简易总结，最后寄语：
多动手，别做思想的巨人，行动的矮者
自动化测试不是万能的，适可而止



thanks