第7章 App测试

章节概述/ Summary

移动设备因其具有智能性、互动性等特点被广泛应用于人们的日常生活。随着移动设备的普及，越来越多的App做了适配，例如QQ、微信、淘宝等，这些App只需要安装在移动设备上就能随时随地使用。由于移动设备的使用环境比较复杂，所以App在正式上线前都需要进行测试，如果没有经过测试就直接上线，可能会出现一系列问题，例如用户信息的泄露、App的崩溃、App在使用过程中经常卡顿等。这些问题不但会增加App的维护成本，而且会影响用户的使用体验。由此可见，App的质量保证离不开测试。本章将对App测试的相关知识进行讲解。

7.1 App测试概述

7.1 App测试概述

- 了解App测试概述，能够描述App测试与PC端软件测试的区别

App（Application，应用程序）是指为实现特定功能或满足特定需求而设计的、可在各类电子终端（包括电脑、手机、平板电脑、智能设备等）上运行的计算机软件。随着智能手机、平板电脑的普及，“App” 的日常语境逐渐偏向 “移动应用” （即移动设备上的软件），但这并不意味着它的定义只能局限于移动设备。

App的特性

设备多样性

传统的PC端软件都是安装在计算机中的， 而可以安装 App 的设备比较多，例如手机、平板电脑、智能手表等，这些设备轻巧便携，满足了用户对移动生活、工作的强烈需求。

网络多样性

传统的 PC 端软件一般都是通过计算机连接有线网络进行使用的，虽然现代的计算机也可以连接无线网络，但是这些网络都是比较稳定。App 通过移动设备连接移动通信网络或无线网络进行使用，例如 3G、4G、5G、Wi-Fi。

平台多样性

传统的 PC 端软件所依赖的平台主要有 Windows、macOS、Linux 等，种类相对较少，而 App 所依赖的平台则有很多种，例如 iOS、Android、鸿蒙、Windows Phone、BlackBerry 等，其中使用较多的平台是 iOS 和Android。

无论是App测试还是PC端软件测试，都离不开测试的基础知识，它们测试的流程和方法基本相同。例如，App测试和PC端软件测试都需要检查界面的布局，它们可以使用同样的测试方法设计测试用例，例如边界值分析法、等价类划分法等。

App与PC端软件在测试方面的区别

页面布局不同

对于PC端软件，计算机设备屏幕比较大，可以同时显示很多信息，用户可以快速看到屏幕上显示的所有信息，页面布局比较灵活。但是对于 App，移动设备屏幕小，显示的信息有限，在测试时需要考虑布局是否合理。

使用场合不同

PC端软件的使用地点比较固定，网络信号相对也比较稳定；而App的使用地点不固定，网络信号相对也不稳定，测试时需要考虑网络信号较差的情况下App的使用情况。此外，还要考虑在移动设备电量不足的情况下，App是否能正常使用。

输入方法不同

PC端软件大多使用键盘和鼠标进行输入，App在移动设备上使用时，输入方法比较多，例如触屏、电容笔、语音等。App测试时需要测试多种输入方法是否都能正常使用。

操作方式不同

PC端软件使用鼠标可以精确操作，而App在移动设备上使用时大多是触屏操作，点击时误差较大，且不支持鼠标指针悬停事件。

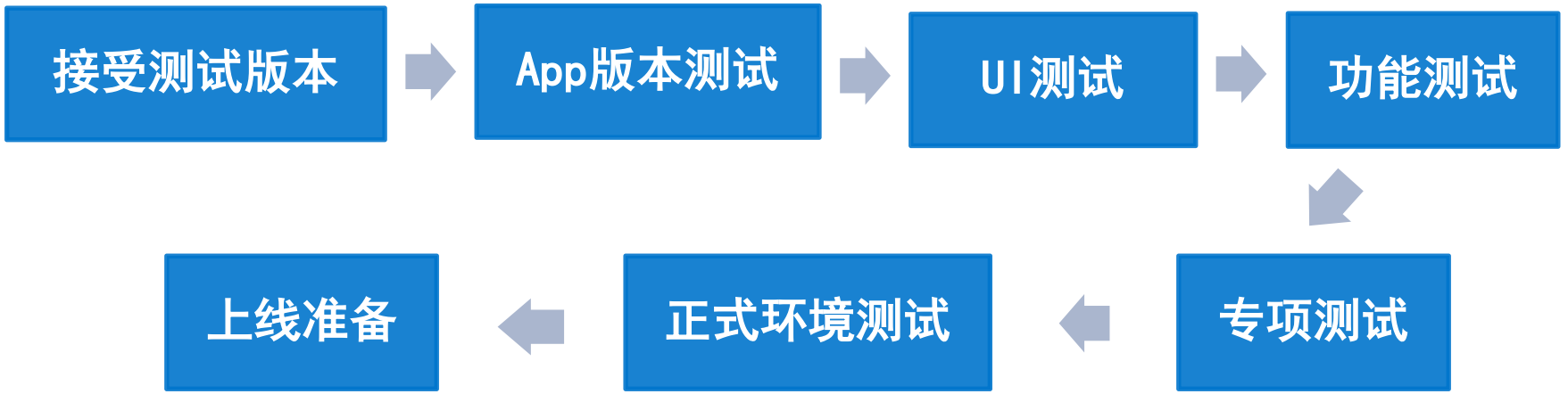
多学一招

App测试的流程

App测试的流程与PC端软件测试的流程大体相同，在测试之前都需要分析需求，然后制定测试计划、编写测试用例等。相对于PC端软件测试，在App测试的过程中，测试人员除了要考虑基本的功能测试、性能测试外，还要考虑App本身固有的属性特征以展开专项测试，因此在测试具体实施细节上也与 PC 端软件测试并不相同。

多学一招

App测试的流程通常包括7个环节，具体如下。



7.2 App测试要点

7.2.1 UI测试

- 了解App的UI测试，能够描述UI测试的3个要点

7.2.1 UI测试

App的UI测试主要是测试App的用户界面（如窗口、菜单、对话框等）的布局风格是否满足用户要求、文字表达是否简洁准确、界面是否美观、操作是否简便等。UI测试的目的是为用户提供相应的访问或浏览功能，确保用户界面符合公司或行业的标准，保证用户界面的友好性、易操作性等。

App的UI测试的3个要点

导航测试

由于移动设备屏幕窄小，显示信息有限，所以App界面的导航尤其重要。

图形测试

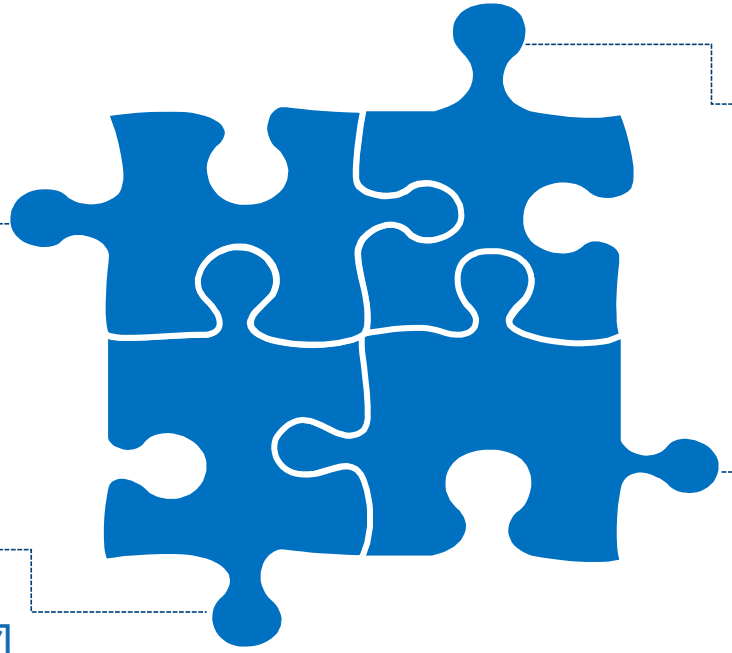
图形测试包括图片、边框、颜色、按钮等，要确保每一个图形都有明确用途。

内容测试

内容测试主要是测试文字的使用情况。

7.2.1 UI测试

在进行导航测试时，通常需要考虑以下4点。



在App功能界面之间是否有按钮和窗口导航。

导航布局合理且直观，符合用户习惯。

导航与App界面结构、菜单、风格是否一致。

导航帮助是否准确，是否需要搜索引擎。

01 使用自适应界面设计，界面展示的内容根据窗口大小自适应。

02 图片质量高，尺寸符合设计要求，能够显示清晰。

03 界面字体与标签风格一致。

04 背景、字体、图片颜色搭配得当，整体使用颜色不宜过多，让用户视觉体验良好。

05 界面中横向与竖向的控件操作方式统一。

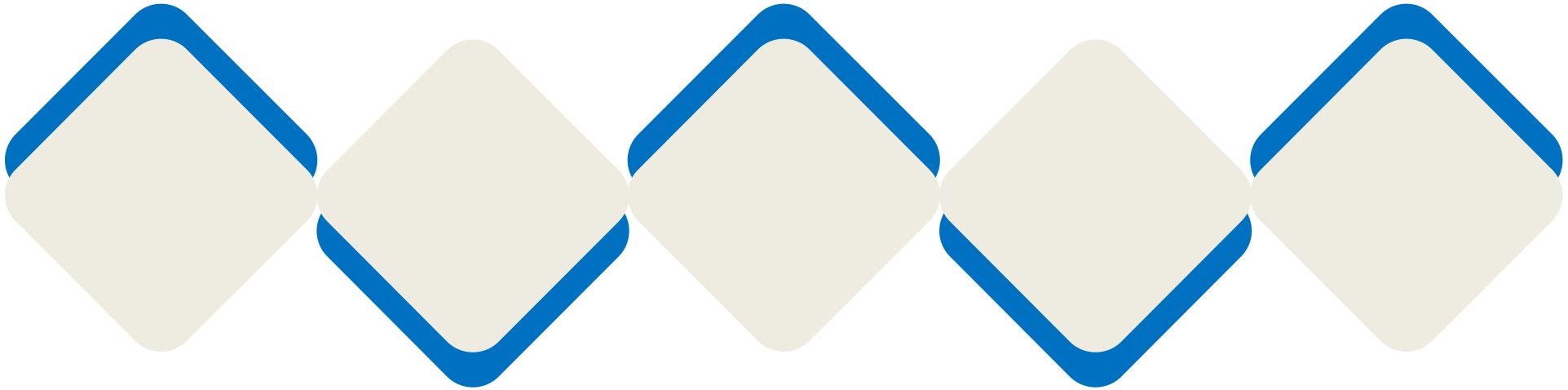
7.2.1 UI测试

内容测试通常需要考虑以下5点。

文字是否表达准确，例如，输入框或提示框中的说明文字是否对应当前的功能。

文字用语是否简洁、友好，是否存在表意不明。

文字长度是否有限制。



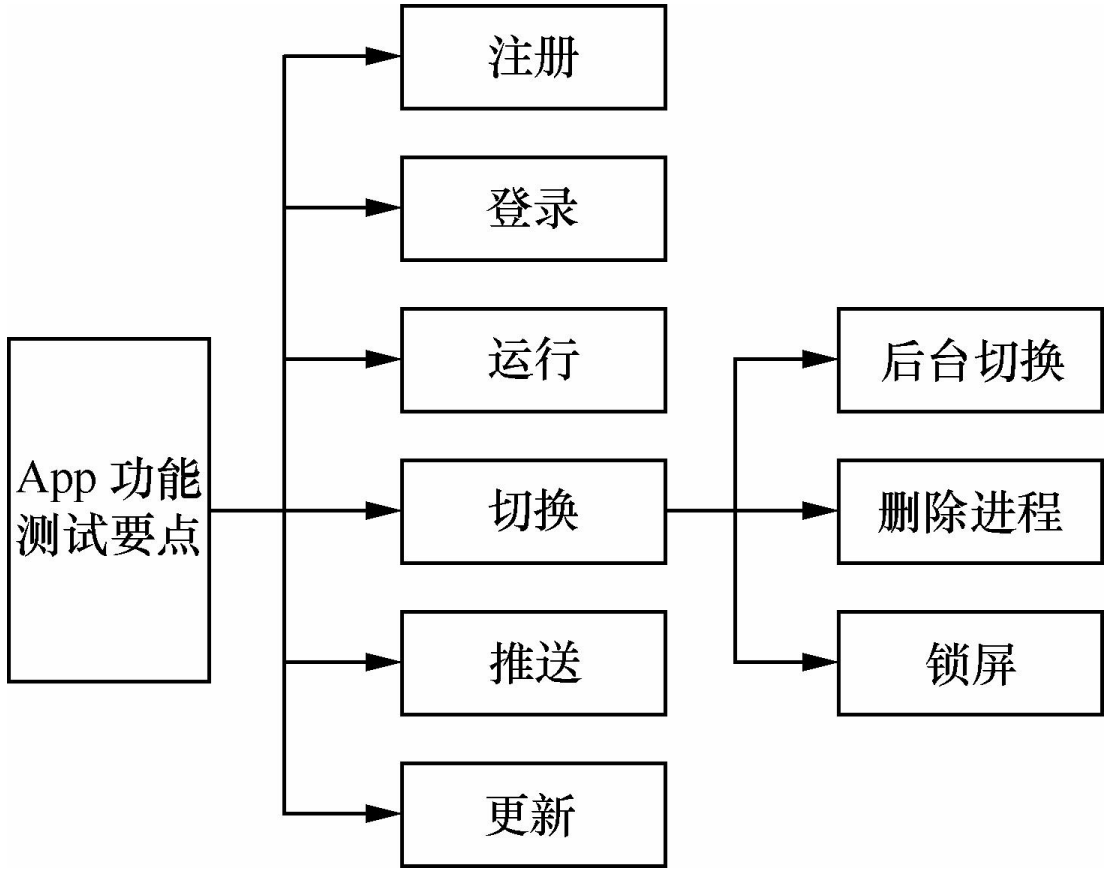
文字是否有错别字。

文字是否有敏感性词汇。

7.2.2 功能测试

- 了解App功能测试，能够描述App功能测试的6个要点

App功能测试主要根据软件需求说明验证App的功能是否得到了正确的实现。App功能测试要点如下图所示。



1.注册

在测试App的注册功能时，测试人员需要根据软件需求说明，测试用户的注册信息是否符合规范，例如用户名、密码、手机号等是否符合规范。此外，还需要测试用户注册成功或失败时，App是否给出相关提示信息。

2.登录

在测试App的登录功能时，通常测试以下4点。

登录方式

登录方式有很多种，例如用户名与密码登录、短信验证登录、手势登录、人脸识别登录、指纹登录、第三方登录（如QQ、微信）等，具体的测试实施过程，需要测试人员根据软件需求说明设计测试用例后再展开。

切换账号登录

当切换账号登录时，测试登录的信息是否及时更新。

多平台登录

有一些App可以同时在移动端和PC端登录，在测试多平台同时登录时，需要关注App是否允许登录、是否给出提示信息、是否及时看到数据的更新等。

超时登录

当用户登录的持续时间太长时，账号信息会在一定时间内过期。在超时登录的情况下，测试App是否给出提示信息。

3.运行

App运行测试包括测试在不同网络环境下，运行App是否正常；在不同系统环境下，运行App是否正常；强行关闭App后，再次运行App是否正常；在运行过程中，如果有来电、短信等通信中断，App是否能暂停运行，优先处理通信，并在处理完后正常恢复运行。

4.切换

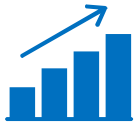
App的切换测试

后台切换

当移动设备同时运行多个App时，在多个App之间进行切换，要确保再次切换回来时App还保持在原来的界面上。

删除进程

测试从后台直接删除进程后，当再次打开App时是否符合概要设计描述，同时测试删除进程时是否将App建立的会话一起删除。



锁屏

锁屏包括手动锁屏和自动锁屏。测试锁屏之后App响应是否符合概要设计描述，例如再次打开App时，它还保持显示锁屏前的界面，并且可以继续使用。

5.推送

使用计算机时，经常会收到推送信息，这些推送信息通常是由系统或软件推送的。在移动设备中，很多App也会发送推送消息，例如支付宝推送一个红包、今日头条推送实时热点新闻等。在对App进行测试时也需要测试推送功能，确保用户可以及时收到推送消息。

6.更新

通常App的更新测试主要从旧功能和新需求这两个方面展开，即确保旧功能可以正常使用的同时，还需要实现新需求。当App有新版本时，测试App是否有更新提示，如果进行更新操作，则需要对App更新后的功能展开测试，确保App更新后的功能可以正常使用。如果取消更新，则需要确保旧版本的App也能正常使用，并且在下次运行App时，仍然出现更新提示。

7.2.3 专项测试

- 了解App专项测试，能够描述App专项测试的6个要点

7.2.3 专项测试

App专项测试包括安装测试、卸载测试、升级测试、交互性测试、弱网测试、耗电量测试等。

1.安装测试

App的安装方式与PC端软件的安装方式稍有不同，App安装测试要考虑App来源、对移动设备的兼容性等。

（1）App的安装渠道比较多，例如谷歌应用商店（Google Play）、应用宝、APP Store等，甚至可以通过扫码安装。对于多渠道的安装方式，在测试时应对每个渠道都进行测试，以确保通过每个渠道都能正常安装App。对于已经安装的App，如果再次安装，需要弹出已安装或更新的提示，而不是产生冲突。

（2）移动设备的种类比较多，例如一个品牌的手机会有不同的系列，每个系列也会有多个型号。此外，App所依赖的平台也比较多，在测试时要考虑App对不同手机、不同平台的兼容性。

（3）App在安装过程中是否可以取消安装，如果可以取消安装，应确保取消安装的处理要与App概要设计描述一致。 例如，如果App概要设计描述取消安装的处理过程为“取消安装并进行回滚处理，将已经安装的文件全部删除“，那么在实际取消安装时也必须如此处理。

（4）如果安装过程中出现意外情况，例如死机、重启、电量耗尽关机等，App安装的处理是否与App概要设计描述一致。例如，如果App概要设计描述安装过程出现意外情况的处理过程为“安装过程中电量耗尽关机，安装中断，当再次开机时继续安装”，那么在实际安装过程中也必须如此处理。

（5）如果移动设备空间不足，要确保有相应提示。例如，当剩下100MB空间时，要安装一个200MB的App，有的App直接提示空间不足，无法安装；有的App会先安装，待空间用尽时再提示。

（6）App安装过程中要进行UI测试，例如在安装过程中给用户提供进度条提示。

（7）App安装完成之后，测试其是否能正常运行，检查安装后的文件夹及文件是否写入指定目录。

2.卸载测试

App卸载测试主要有以下5点。

1. 卸载时，有卸载提示信息。
2. App在卸载过程中是否支持取消卸载，如果支持取消卸载，要确保取消卸载的处理与App概要设计描述一致。
3. App卸载的过程中如果出现意外情况，例如死机、重启、电量耗尽关机等，要有相应的处理措施。如进行回滚，当再次开机时需要重新卸载；中断卸载，当再次开机时继续卸载；启动后台进程守护卸载，当再次开机时提示卸载完成。
4. App卸载过程要进行UI测试，例如在卸载过程中，给用户提供进度条提示。
5. 卸载完成后，关于App相应的安装文件是否要全部删除，应当给用户提供提示信息，提示相应文件全部删除或者让用户自己选择是否删除。

3.升级测试

升级测试是在已安装App的基础上进行的，测试要点有以下4点。

如果有新版本升级，打开App时要有相应提示。

升级包下载中断时要有相应处理措施，支持继续下载或者重新下载。

App安装渠道有多种，相应的升级渠道也有多种，要对多渠道升级进行测试，确保每个渠道的升级都能顺利完成。

测试不同操作系统版本中的App升级是否都能通过。

4.交互性测试

移动设备通常都有电话、短信、蓝牙等软件，App在使用时难免会受到干扰。例如，在使用 App 的过程中，如果需要拨打电话，接听电话，启动蓝牙、相机、手电筒等，App 要做好相应的处理措施，确保自身不会产生功能性错误。

5.弱网测试

App在移动设备上使用时，通常需要连接移动网络，由于移动网络的情况复杂多变，网络信号会受到环境的影响，所以容易出现网络不稳定的情况。然而很多App的一些隐藏问题只有在复杂的网络环境下才会显现出来。例如，正在使用的App遇到网络信号切换或变弱时，不能响应或产生功能性错误，在测试时要特别对App进行弱网测试，尽早发现问题。

6.耗电量测试

移动设备的电量有限一直是困扰用户的一个问题，同时也是移动设备发展的一个瓶颈。如果App架构设计不好，或者代码有缺陷，就可能导致电量消耗比较大，因此App耗电量测试也很重要。如果App耗电量较大，应改进App，使其在电量不足的情况下释放掉一部分性能以节省电量。

7.2.4 性能测试

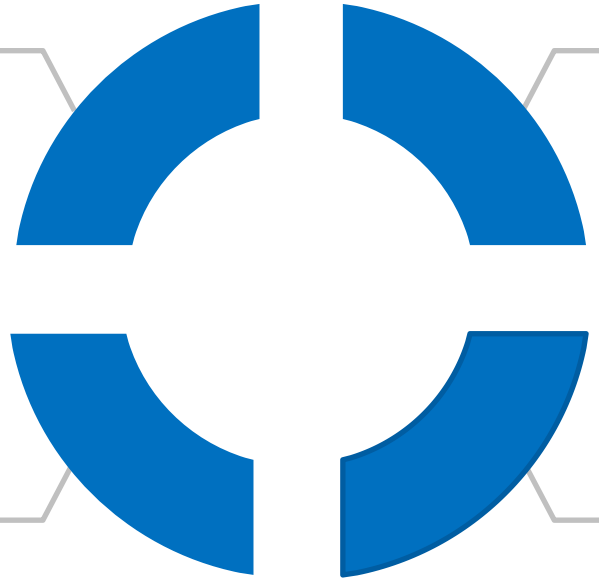
- 了解App性能测试，能够描述App的性能测试的4个要点

7.2.4 性能测试

App性能测试主要测试App在边界、压力等极端条件下运行是否能满足用户需求，例如，在电量不足、访问量增大等情况下App运行是否正常。

7.2.4 性能测试

App性能测试的要点如下。



边界测试

在各种边界压力下，例如电量不足、存储空间不足、网络不稳定时，测试App是否能正确响应和正常运行。

压力测试

对App不断施加压力，例如不断增加负载、不断增大数据吞吐量等，以确定App的服务瓶颈，获得App能提供的最强性能，确定App性能是否满足用户需求。

耗能测试

测试App运行时对移动设备的资源占用情况，包括内存、CPU使用率，在App耗能测试时需要验证App在长期运行时的耗电量是否满足用户需求。

响应能力测试

响应能力测试实质上也是一种压力测试，即测试在一定条件下App是否可以正确响应，以及响应时间是否满足用户需求。

7.2.5 兼容性测试

- 了解App的兼容性测试，能够描述App的兼容性测试的5个要点

7.2.5 兼容性测试

兼容性是指软件之间、硬件之间或软硬件组合系统之间相互协调工作的程度。对于App的兼容性，如果某个App能够稳定地工作在若干个操作系统中，并且不会出现频繁崩溃、意外退出等问题，则说明App的兼容性比较好。

7.2.5 兼容性测试

随着App应用的范围越来越大，用户群体逐渐增多，用户使用的移动设备型号也越来越多，这使得App兼容性测试成为App质量保证必须要考虑的测试要点。兼容性测试的目的是提高App产品的质量，尽可能使App产品达到平台无关性，使App产品的市场更广阔。通常 App 兼容性测试的要点主要有系统、屏幕分辨率、屏幕尺寸、网络和品牌。

1.系统兼容测试

App系统兼容性测试主要涉及Android和iOS，其中Android系统又分为14、15等版本；iOS又分为17、18、26等版本。由于不同的系统版本有不同的特征，所以在不同的系统上使用App时都有可能产生各种各样的兼容问题，例如，某一款App在Android系统上能够正常安装和使用，而在iOS上却无法安装和使用，所以在进行App兼容性测试时需要覆盖系统兼容。

2.屏幕分辨率兼容测试

目前手机的屏幕种类很多，常见的有全面屏、刘海屏、水滴屏、折叠屏等，不同的手机屏幕的分辨率也有所不同。在不同分辨率的设备上使用App时，呈现的界面效果也会有所差异。如果没有适配不同手机的屏幕分辨率，则可能会影响用户的使用体验。例如，在分辨率为1920像素×1080像素的屏幕中显示的App界面样式清晰美观，满足用户的使用需求，而在1280像素× 720像素的分辨率的屏幕中可能出现App界面样式显示不全、图片模糊等问题，所以在进行App兼容性测试时需要在不同分辨率的设备上测试，并观察用户界面的效果。

3.屏幕尺寸兼容测试

在分辨率相同但屏幕尺寸不同的移动设备上使用App时，容易出现图片显示不完整、字体大小不一致等问题，因此需要测试屏幕大小的兼容性。

4.网络兼容测试

很多App的使用需要连接网络，在测试网络兼容性时要保证网络环境能够全部覆盖，例如Wi-Fi、2G、3G、4G、5G，同时需要考虑电信、移动、联通等运营商的网络环境。在切换网络环境时，测试App能否兼容不同的网络环境。

5.品牌兼容性测试

App在不同品牌的移动设备上使用时，也可能出现缺陷，常见的移动设备品牌有华为、小米、三星、OPPO、vivo、苹果等。由于不同品牌的移动设备在运行速度、软件兼容上有区别，所以需要测试App是否可以在不同品牌的移动设备上使用。

多学一招

第三方测试平台

App可以使用第三方平台进行测试，第三方平台（如阿里EasyTest、华为云测、贵众云测试等）提供了全面、专业的测试服务，用户可选择品牌机型、操作系统版本、性能测试、功能测试等，极大地提高了App测试效率。

7.3 搭建App测试环境

7.3.1 安装JDK与Android SDK

- 掌握JDK与Android SDK的安装，能够独立安装JDK与Android SDK

7.3.1 安装JDK与Android SDK

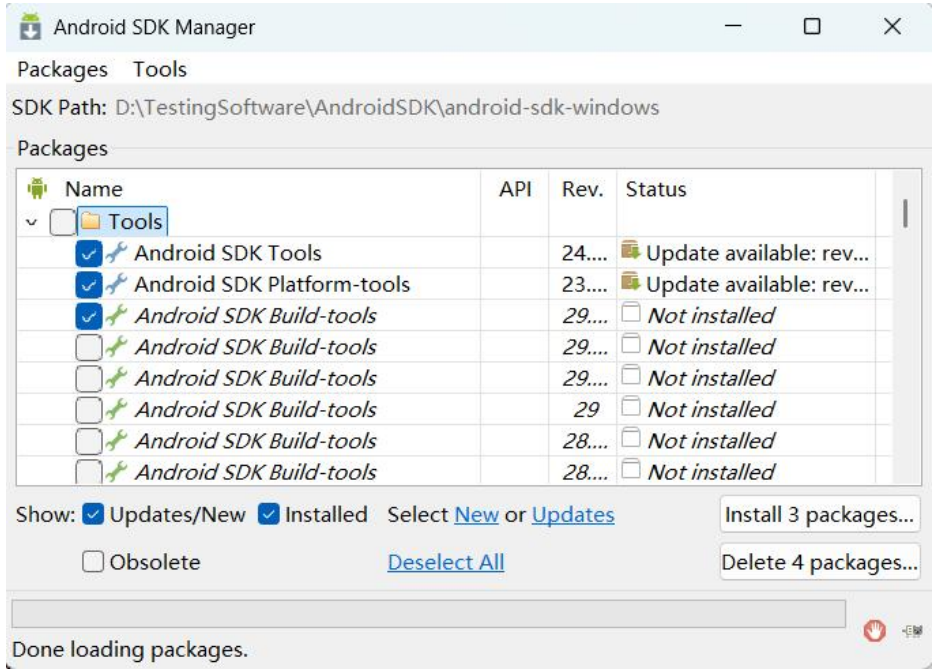
由于安卓手机的App是基于Android系统开发的，使用的编程语言是Java，所以在运行或测试App时需要使用Java环境，搭建Java环境也就是安装JDK。在测试App的过程中，还需要定位App界面元素，此时需要使用Android SDK，所以还需要安装Android SDK。

7.3.1 安装Android SDK

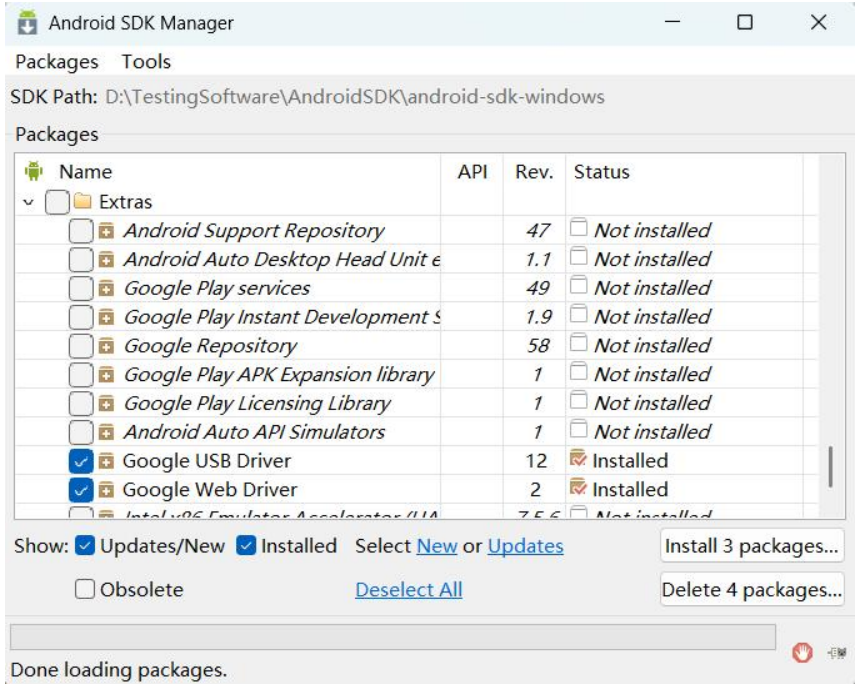
1 android-sdk_r24.1.2-windows.zip

2 SDK Manager.exe

3

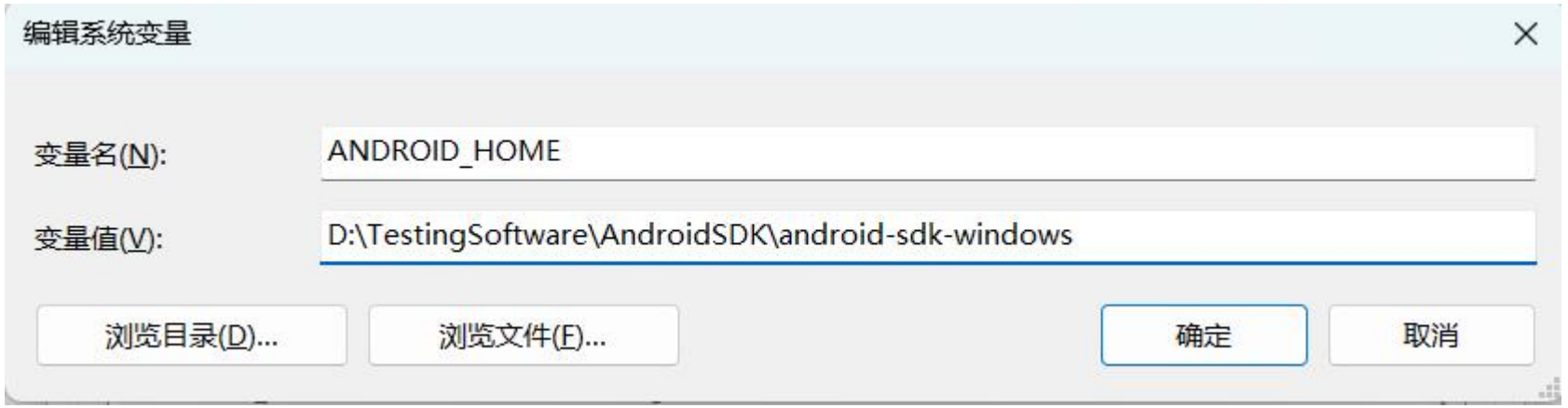


4



7.3.1 Android SDK

5



6

```
%ANDROID_HOME%\tools
%ANDROID_HOME%\platform-tools
%ANDROID_HOME%\build-tools
```

7

```
C:\Users\Administrator>adb version
Android Debug Bridge version 1.0.32
Revision eac51f2bb6a8-android
```

7.3.2 安装Android模拟器

- 掌握Android模拟器的安装，能够独立下载与安装雷电模拟器

7.3.2 安装Android模拟器

当测试Android系统的App时，需要将App运行在Android模拟器或Android系统的其他设备上，然后测试App中的各项功能是否会出现缺陷。以第三方的雷电模拟器为例，该模拟器的安装步骤比较简单，没有复杂的操作，直接单击“下一步”就可以完成安装，所以此处不再详细描述雷电模拟器的安装步骤。

多学一招

ADB调试工具

ADB（Android Debug Bridge），本质上是一个客户端-服务器端程序。其中客户端是用来操作的电脑，服务端是 Android 设备。它是一个用于管理Android设备（如模拟器、手机等）的调试工具，位于Android SDK安装目录下的platform-tools文件夹中。当配置完Android的环境变量后，可以直接在命令提示符窗口中使用adb命令对Android设备进行操作或获取设备上安装的App信息，例如，在设备上安装App、卸载App、连接某个设备、获取App的包名和界面名等信息。

多学一招

在进行App测试时经常会使用一些adb命令来启动或停止ADB服务器、获取App的日志信息、连接或断开Android设备等。常用的adb命令如下表所示。

adb命令	说明
adb start-server	启动ADB服务器
adb kill-server	停止ADB服务器
adb devices	查看设备名称
adb logcat	获取日志信息
adb connect IP地址	连接某个设备

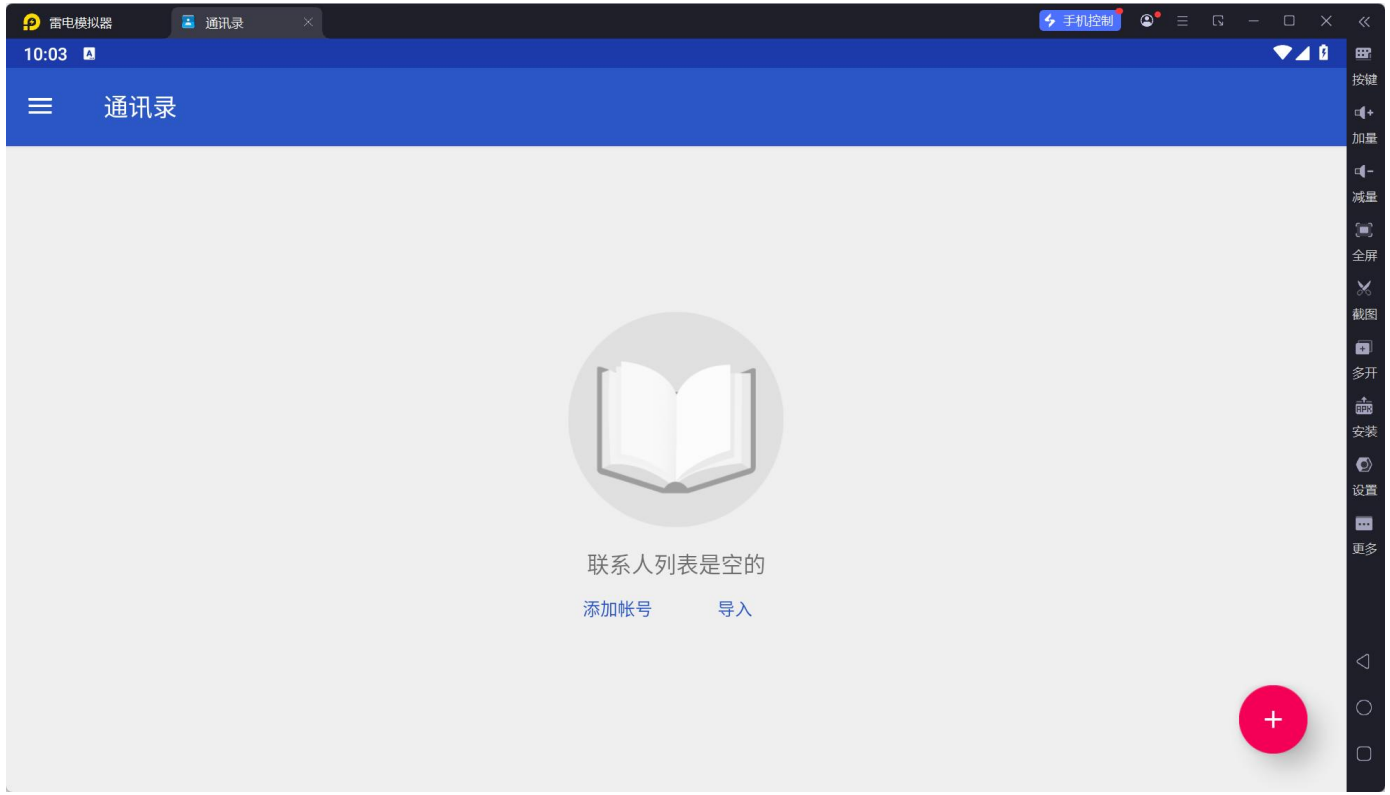
多学一招

adb命令	说明
adb install apk的文件路径	安装App
adb uninstall App的包名	卸载App
adb --help	查看adb命令的帮助
adb shell dumpsys window windows | findstr mFocusedApp	获取App的包名和界面名
adb shell dumpsys activity | find "mFocusedActivity"	获取App的包名和界面名

7.3.3 配置Android环境变量

多学一招

下面以获取雷电模拟器中“通讯录”App的包名和界面名为例，演示adb命令的使用。首先打开雷电模拟器，然后在该模拟器中找到“通讯录”App，双击该App进入通讯录界面，如右图所示。



多学一招

打开计算中的命令提示符窗口，执行如下命令。

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

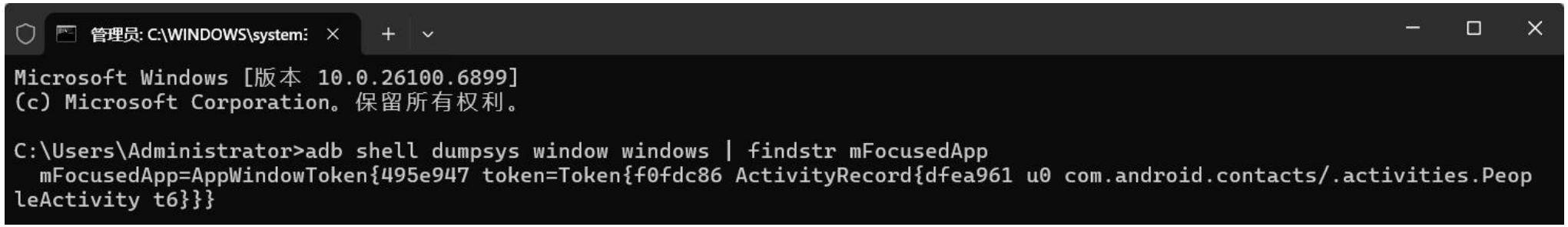
或者

```
adb shell dumpsys activity | find "mFocusedActivity"
```

7.3.3 配置Android环境变量

多学一招

adb命令成功获取了通讯录App的包名和界面名，其中包名为com.android.contacts，界面名为.activities.PeopleActivity。



7.3.4 安装Appium与Appium-Python-Client库

- 掌握Appium与Appium-Python-Client库的安装，能够完成Appium与Appium-Python-Client库的安装

7.3.4 安装Appium与Appium-Python-Client库

Appium是一个开源的、跨平台的移动端自动化测试工具，它支持使用WebDriver协议驱动Android系统、iOS和Windows系统上安装的应用程序。Appium支持多系统，（如Windows、Linux ）并支持多语言（如Python、Java、JavaScript等）。

Appium-Python-Client 是 Python 语言的 Appium 客户端库，它实现了 Appium 协议，允许开发者或测试人员使用 Python 代码与 Appium 服务器进行通信，从而编写自动化脚本控制移动应用（iOS/Android）的操作，比如模拟点击、输入文本、获取元素属性等，是 Python 开发者进行移动应用自动化测试的核心工具

7.3.4 安装Appium与Appium-Python-Client库

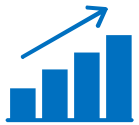
Appium的测试对象

原生应用

在iOS或Android系统中运行的应用，可直接通过应用商店下载与安装。

移动Web应用

在移动端浏览器中可以访问的Web应用，Appium支持iOS系统中安装的Safari和Chrome浏览器，以及Android系统中的内置浏览器。



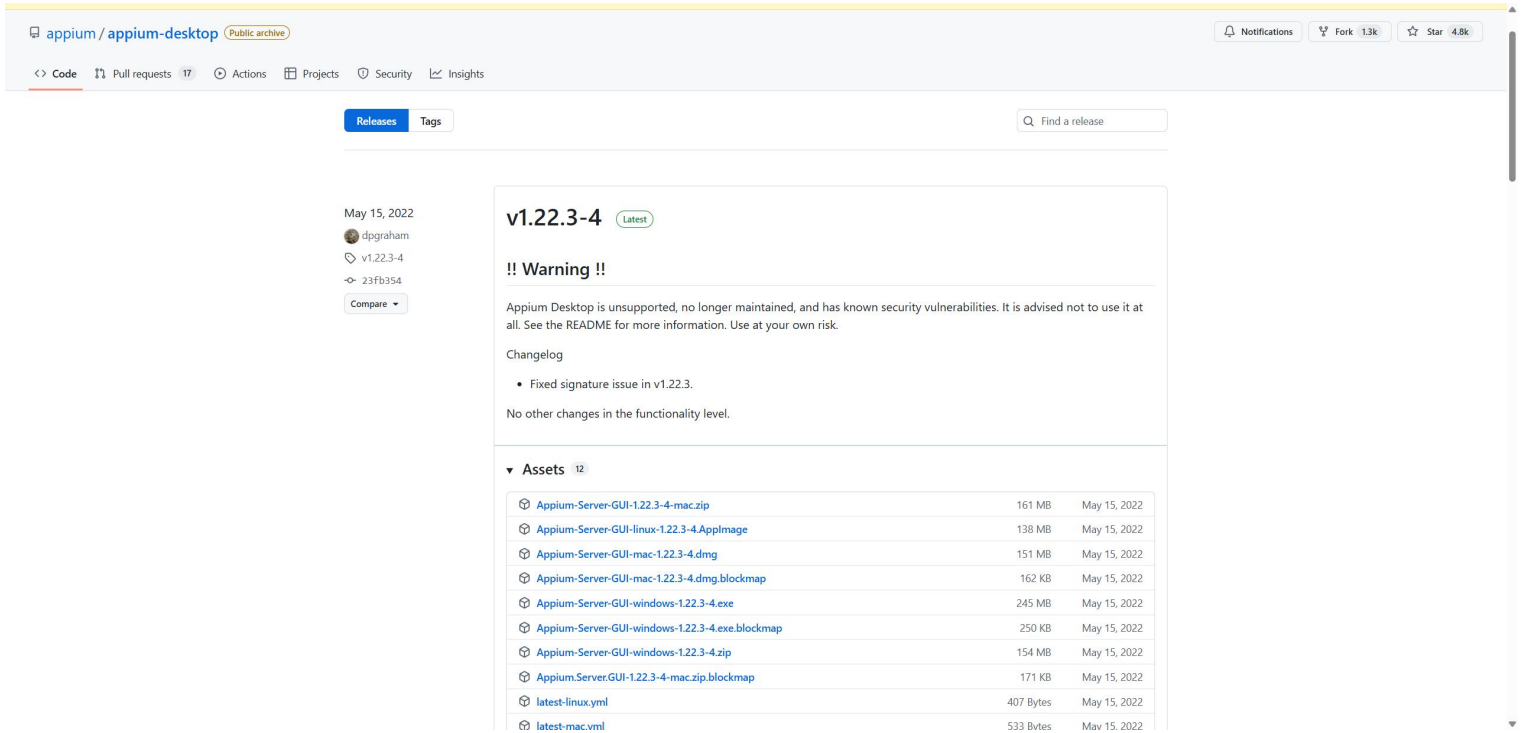
混合应用

用原生应用封装网页视图的应用程序。比如微信的聊天和朋友圈为原生代码，公众号、部分小程序等则通过网页技术（HTML5）实现

7.3.4 安装Appium与Appium-Python-Client库

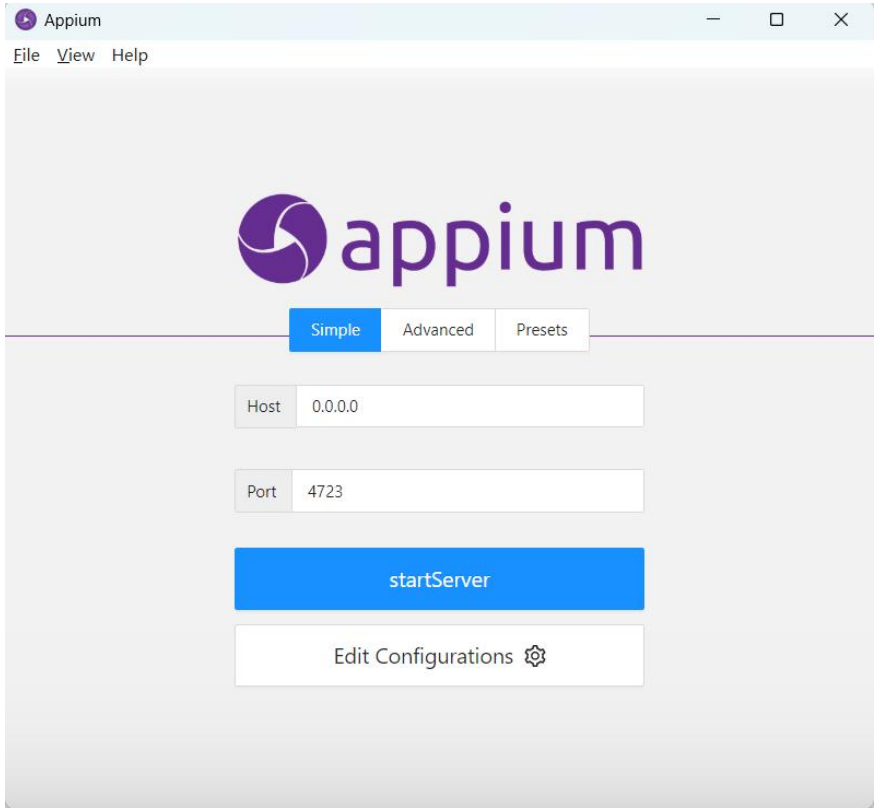
1.安装Appium

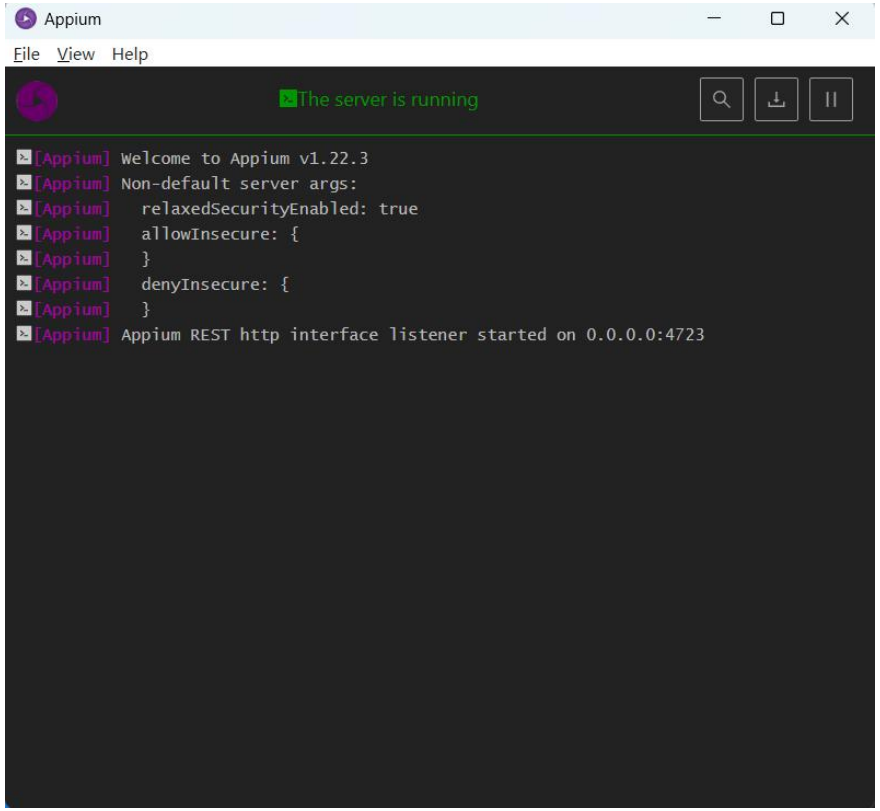
访问Appium官方网站（https://github.com/appium/appium-desktop/releases），在该页面单击“Appium-Server-GUI-windows-1.22.3-4.exe”即可下载Appium的安装文件，如下图所示。



7.3.4 安装Appium与Appium-Python-Client库

Appium安装完成后，单击Appium启动页面的“startServer”按钮进入服务器运行页面，如果输出“Welcome to Appium v1.22.3”等启动信息，则说明Appium启动成功。





1.安装Appium-Python-Client库

在计算机中打开命令提示符窗口，通过pip命令安装Appium-Python-Client库，具体安装命令如下。

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

安装完Appium-Python-Client库后，可以通过“pip list”命令验证Appium-Python-Client库是否安装成功。

```
pip list
```

7.3.5 Appium Inspector工具的简单使用

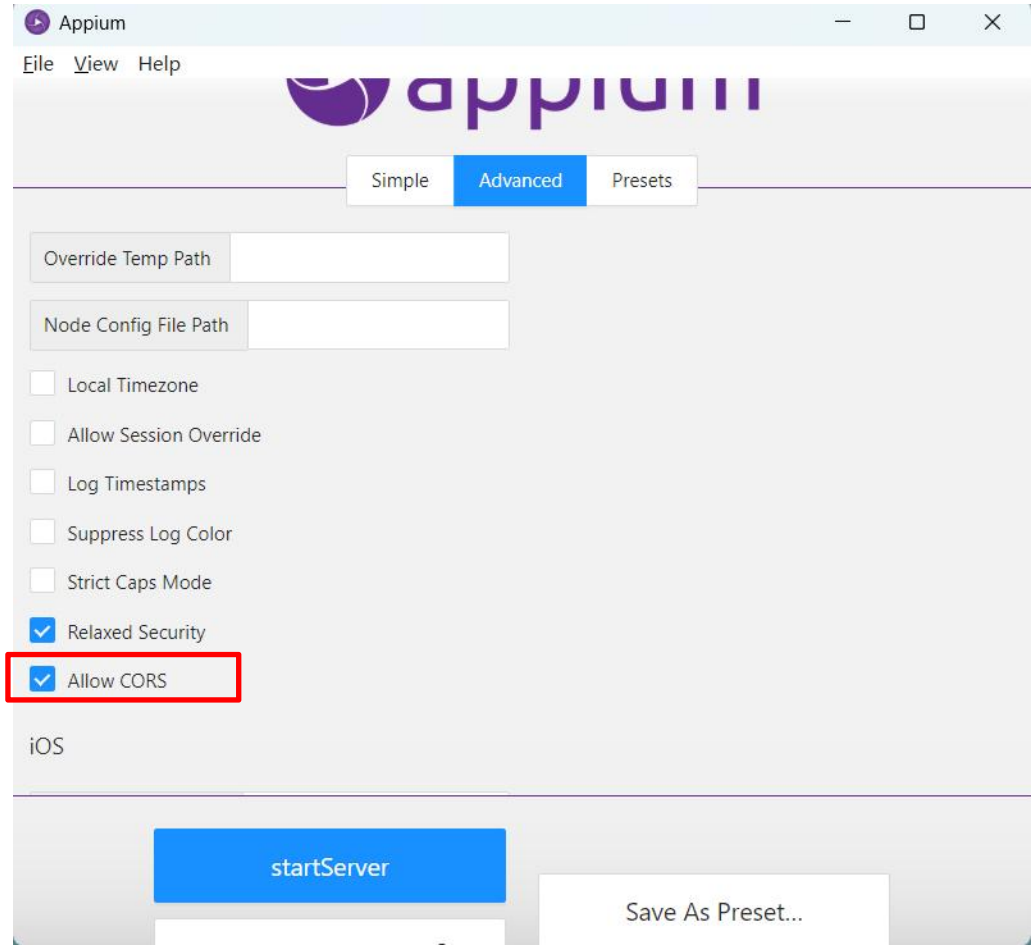
- 掌握Appium Inspector工具的使用，能够灵活应用Appium Inspector工具定位App界面的元素

7.3.5 Appium Inspector工具的简单使用

Appium Inspector 是 Appium 框架配套的可视化调试工具，能直观展示移动应用的 UI 结构，支持实时识别按钮、文本框等界面元素，便于测试人员定位元素、调试自动化脚本，还可跨 iOS 和 Android 平台使用，部分版本具备操作录制生成脚本框架的功能。

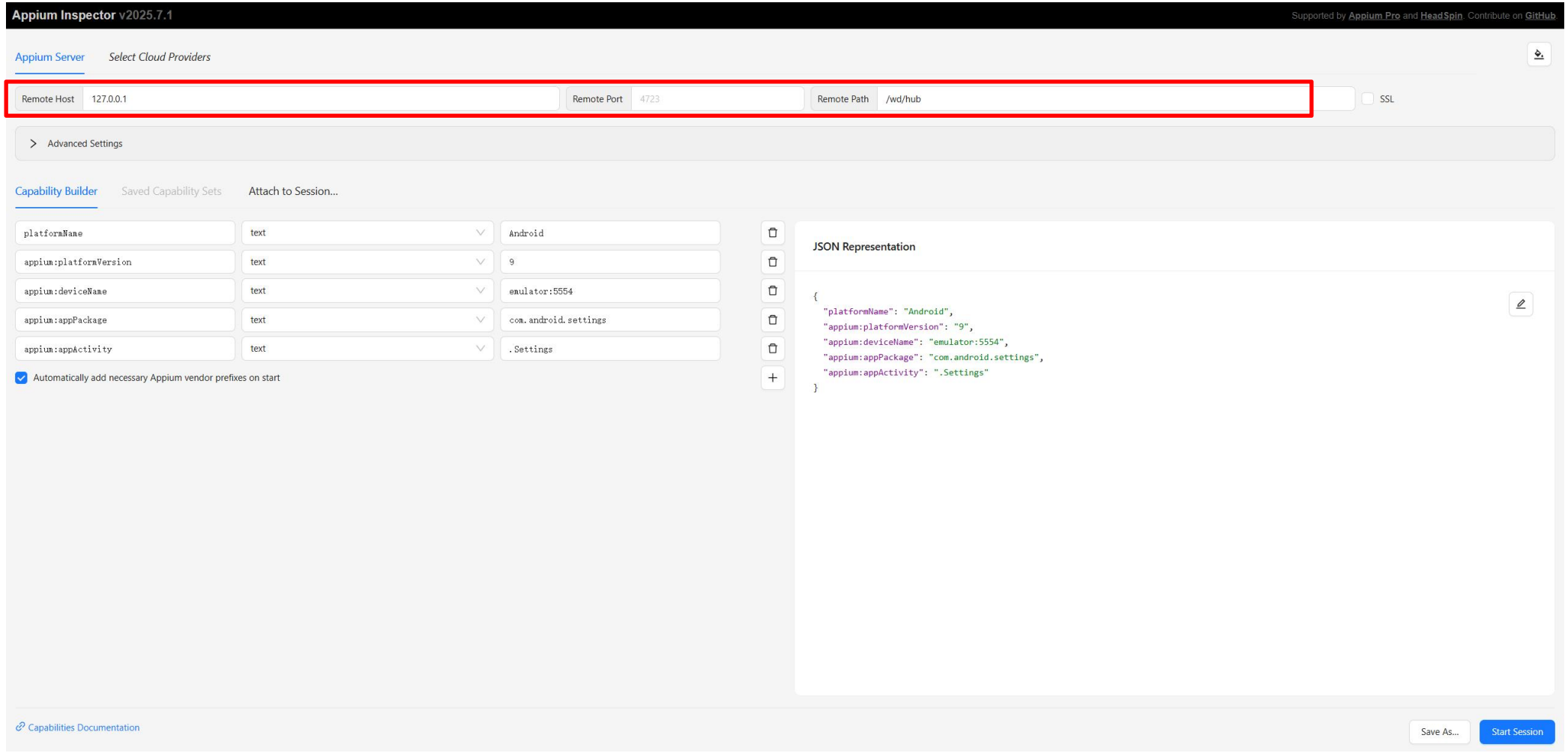
7.3.5 Appium Inspector工具的简单使用

在使用Appium Inspector工具定位App界面元素时，首先打开Appium软件，启动Appium Server（勾选Allow CORS选项，让网页版Inspector 跨域连接到本地服务器）



7.3.5 Appium Inspector工具的简单使用

然后进入网址https://inspector.appiumpro.com/， 输入配置信息连接Appium Server，

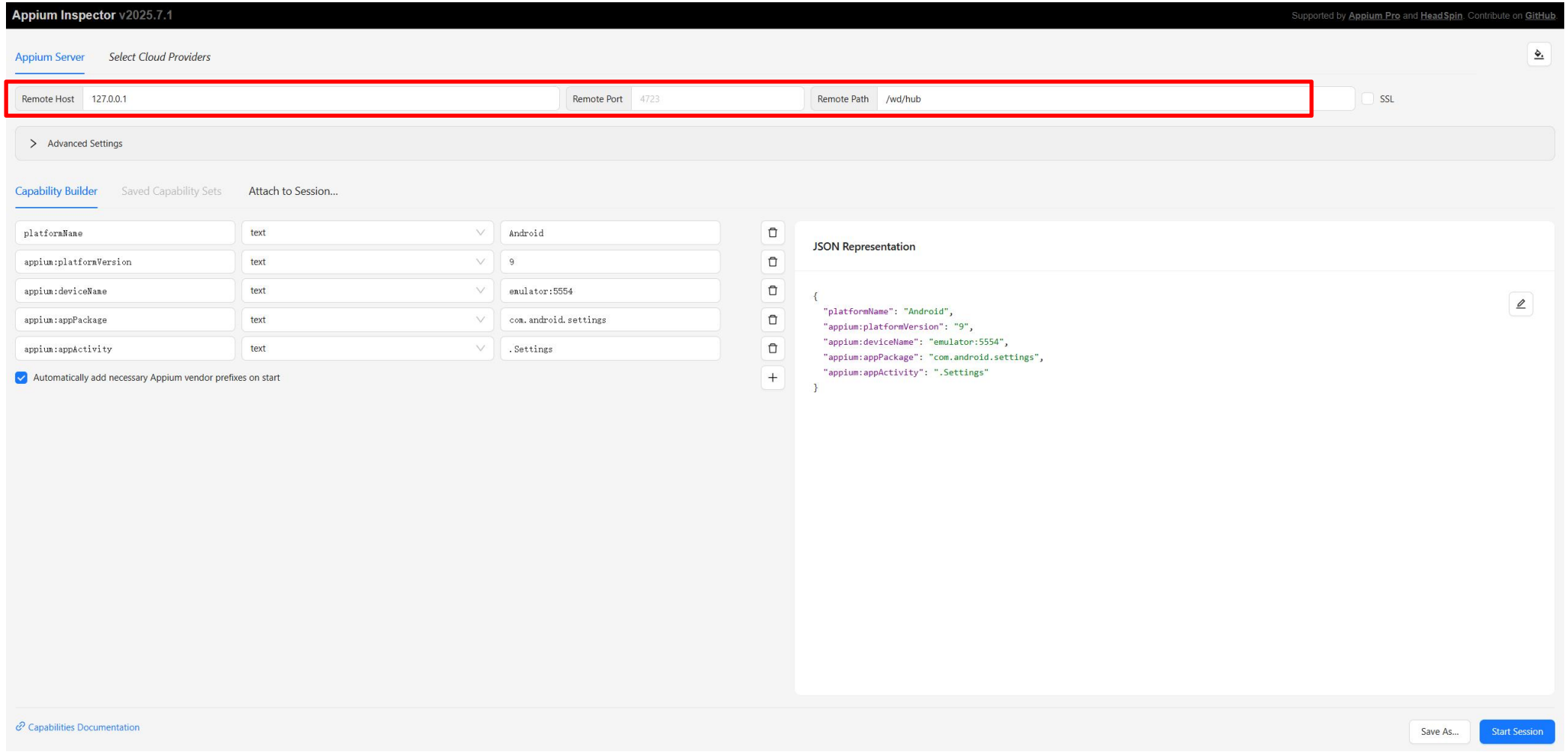


7.3.5 Appium Inspector工具的简单使用

- localhost（127.0.0.1）：表示 Appium Server 运行在本地计算机（与模拟器 / 设备同一台电脑），如果 Appium Server 部署在其他服务器上，这里需要替换为对应服务器的 IP 地址（例如 http://192.168.1.100:4723/wd/hub）。
- 4723：是 Appium Server 的默认端口号。启动 Appium Server 时，默认会监听这个端口（可在启动时自定义端口，例如4725，此时地址需改为http://localhost:4725/wd/hub）
- /wd/hub：是 Appium 遵循 WebDriver 协议的固定路径，必须包含，不可省略

7.3.5 Appium Inspector工具的简单使用

接下来在“Capability Builder”中设置连接设备所需的参数（如平台、设备名、应用包名等），生成 JSON 格式的配置用于启动会话。最后点击“Start Session”。



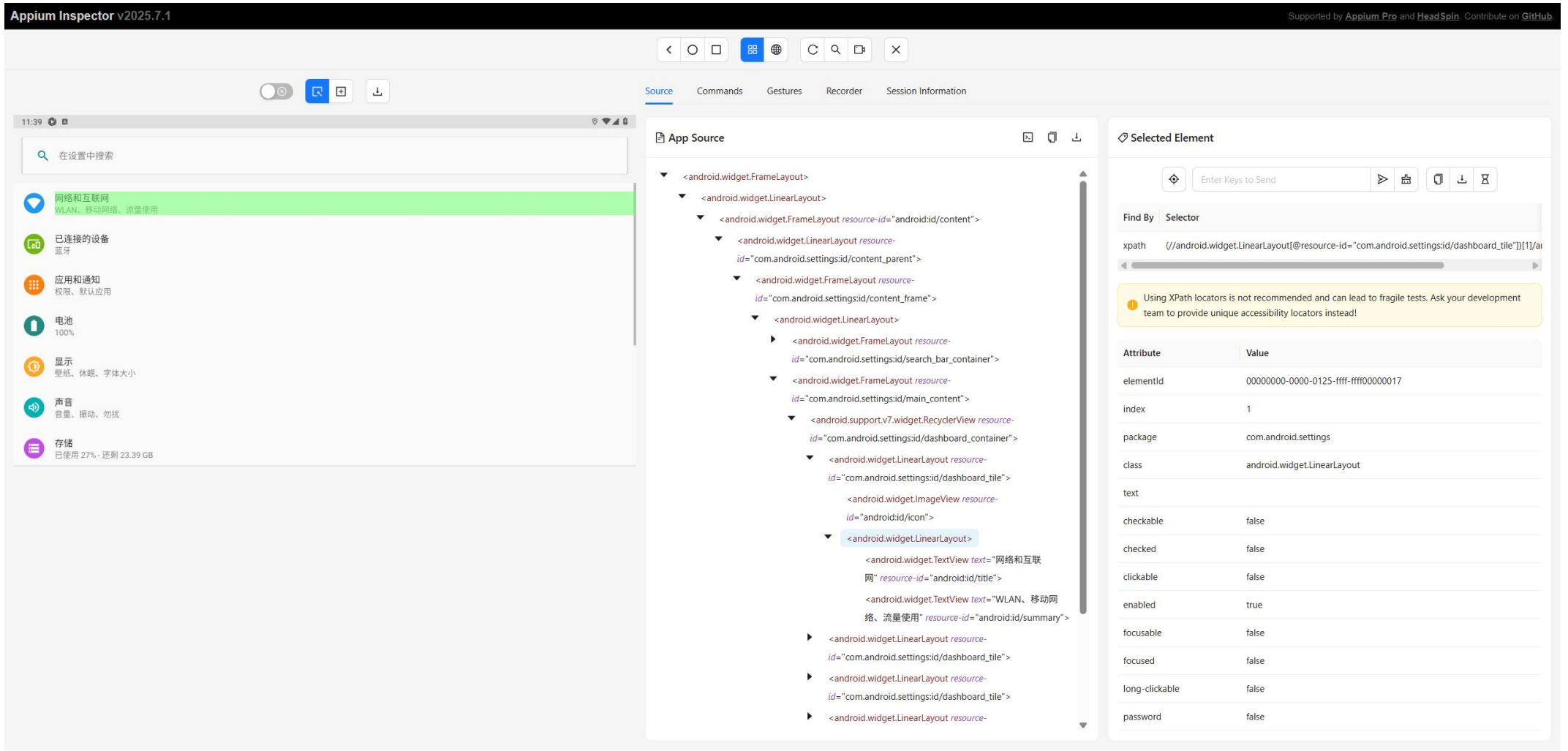
7.3.5 Appium Inspector工具的简单使用

- **platformName**：固定值为 Android（因为是安卓设备），无需额外获取
- **platformVersion**（安卓系统版本）：adb shell getprop ro.build.version.release
- **deviceName**（设备名称 / 端口）：adb devices，雷电模拟器默认端口通常是 5554，所以设备名常写为 emulator:5554 或 emulator-5554
- **appPackage 和 appActivity（应用包名和启动页）**：这两个参数用于指定要启动的应用（这里是 “设置”）

```
adb shell dumpsys window | findstr mCurrentFocus
```

7.3.5 Appium Inspector工具的简单使用

下面使用Appium Inspector工具获取雷电模拟器中“设置”App的界面布局信息。

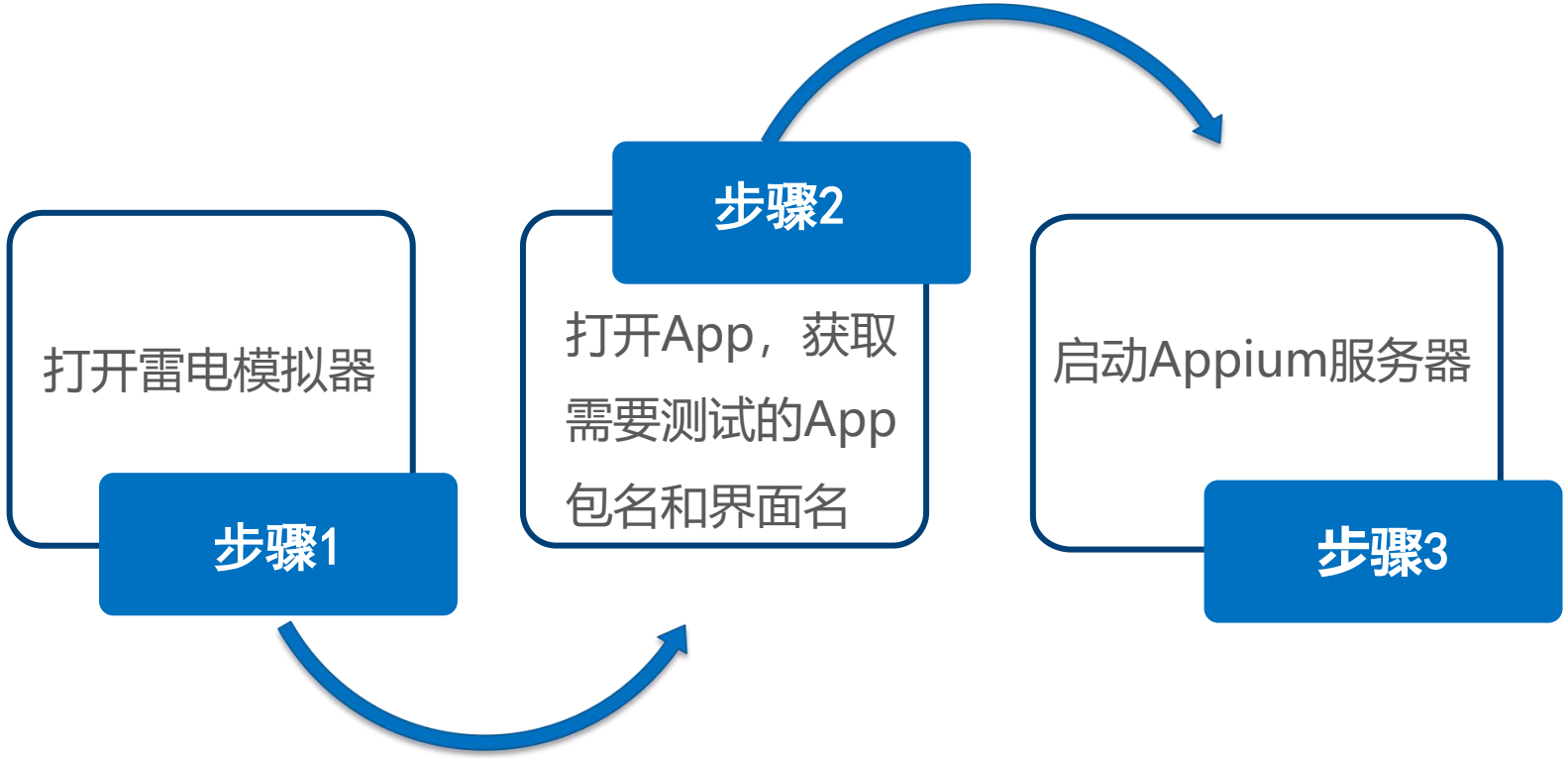


7.3.6 App测试入门示例

- 掌握App测试入门示例，能够编写App自动化测试脚本

7.3.6 App测试入门示例

App测试的入门示例具体实现步骤如下。



7.3.6 App测试入门示例

下面以雷电模拟器中的“设置”App为例，演示如何打开“设置”App。首先创建一个名为Test1.py的文件，在该文件中编写自动化测试代码实现自动打开“设置”App界面。

```
import time
from appium import webdriver
from appium.options.android import UiAutomator2Options
desired_caps = dict()
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '9'
desired_caps['deviceName'] = 'emulator:5554'
desired_caps['appPackage'] = 'com.android.settings'
desired_caps['appActivity'] = '.Settings'
Options = UiAutomator2Options().load_capabilities(desired_caps)
driver = webdriver.Remote("http://localhost:4723/wd/hub", options = Options)
time.sleep(3)
driver.quit()
```

7.4 Appium的基本应用

7.4.1 Appium元素定位

- 掌握Appium元素定位的方法，能够使用Appium定位App界面中的元素

1.通过 resource-id定位

如果测试的App界面元素有resource-id属性，并且该属性唯一，则可以调用find_element方法定位元素，该方法的语法格式如下。

```
find_element(AppiumBy.ID, “resource_id”)
```

2.通过 class定位

在Appium中通过class定位的方法也是通过调用find_element方法定位元素，该方法的语法格式如下。

```
find_element (AppiumBy.CLASS_NAME, “class_name”)
```

上述方法中的参数class_name表示元素的class属性值，需要注意的是，在App的同一个界面中，通常有多个class，因此在调用该方法定位元素时，需要验证元素的class是否唯一。

3.通过 content-desc定位

当测试的App界面元素有content-desc属性时，还是调用find_elment方法定位元素，该方法的语法格式如下。

```
find_element (AppiumBy.ACCESSIBILITY_ID, “accessibility_id”)
```

上述方法中的参数accessibility_id的取值是属性content-desc的值。

4.通过 xpath定位

xpath定位是根据元素的路径表达式选取XML文档中的节点或节点集，如果需要通过xpath定位元素时，依然调用find_element方法，该方法的语法格式如下。

```
find_element (AppiumBy.XPATH, “xpath”)
```

上述方法中的参数xpath表示元素的相对路径或绝对路径。在xpath中，绝对路径是从HTML根节点开始，相对路径则是从任意节点开始。

7.4.1 Appium元素定位

下面以雷电模拟器中的“设置”App为例，演示如何调用Appium元素定位方法来定位“设置”App中设置界面的元素，并对这些元素进行相应的单击与输入操作，以定位放大镜图标、输入框和返回图标等元素为例进行演示。创建一个Test2.py文件，在该文件中对设置界面元素进行定位与操作。

7.4.1 Appium元素定位


```
import time
from appium import webdriver
from appium.options.android import UiAutomator2Options
from appium.webdriver.common.appiumby import AppiumBy
desired_caps = dict()
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '9'
desired_caps['deviceName'] = 'emulator:5554'
desired_caps['appPackage'] = 'com.android.settings'
desired_caps['appActivity'] = '.Settings'
desired_caps['automationName'] = 'UiAutomator2'  # 明确指定自动化引擎
Options = UiAutomator2Options().load_capabilities(desired_caps)
driver = webdriver.Remote("http://localhost:4723/wd/hub", options = Options)
```

7.4.1 Appium元素定位


```
# 通过resource-id定位放大镜图标并单击
driver.find_element(AppiumBy.ID,"com.android.settings:id/search_bar").click()
# 通过class定位输入框并输入“电池"
input_box = driver.find_element(AppiumBy.ID,"android:id/search_src_text")
input_box.send_keys("电池")
# 通过xpath定位返回图标并单击
return_icon = driver.find_element(AppiumBy.XPATH,"//android.widget.ImageButton[@content-desc='向上导航‘]“).click
# 通过content-desc定位放大镜图标并单击
driver.find_element(AppiumBy.ACCESSIBILITY_ID,"向上导航").click()

# 等待3秒后退出“设置”App并关闭驱动
time.sleep(3)
driver.quit()
```

7.4.2 Appium元素操作

- 掌握Appium元素操作的方法，能够使用Appium操作App界面中的元素

7.4.2 Appium元素操作

Appium 中基本元素操作通过定位元素（如 ID、XPath、ClassName 等方式）后进行，常用方法包括：click () 点击元素、send_keys () 输入文本、clear () 清空输入框、get_text () 获取元素文本、is_displayed（）判断元素是否可见、get_attribute () 获取元素属性，这些方法通过客户端库（如 Appium-Python-Client）调用，实现对移动应用界面元素的自动化控制。

Appium元素操作的常用方法和属性如下表。

方法/属性	说明
is_displayed()	该方法用于判断元素是否可见，返回结果为布尔值
is_enabled()	该方法用于判断元素是否可用，返回结果为布尔值
is_selected()	该方法用于判断元素是否被选中，返回结果为布尔值
text	该属性用于获取元素的text值，返回结果为元素的text属性值

方法/属性	说明
tag_name	该属性用于获取元素的标签名，App中的原生应用没有标签名，默认为None
size	该属性用于获取元素的宽和高，返回结果为字典类型的数据
location	该属性用于获取元素的坐标，返回结果为字典类型的数据
rect	该属性用于获取元素的宽、高和坐标，返回结果为字典类型的数据

7.4.2 Appium元素操作

下面以雷电模拟器中的“设置”App为例，演示常用的方法和属性。创建一个Test3.py文件，在该文件中首先判断“设置”App中设置界面的蓝牙“开启/关闭”按钮是否可见、可用、被选中，并依次输出判断结果，然后获取蓝牙“开启/关闭”按钮的text值、标签名、宽高、坐标，并依次输出获取结果。

7.4.1 Appium元素定位


```
import time
from appium import webdriver
from appium.options.android import UiAutomator2Options
from appium.webdriver.common.appiumby import AppiumBy
desired_caps = dict()
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '9'
desired_caps['deviceName'] = 'emulator:5554'
desired_caps['appPackage'] = 'com.android.settings'
desired_caps['appActivity'] = '.Settings'
desired_caps['automationName'] = 'UiAutomator2'  # 明确指定自动化引擎
Options = UiAutomator2Options().load_capabilities(desired_caps)
driver = webdriver.Remote("http://localhost:4723/wd/hub", options = Options)
```

7.4.1 Appium元素定位


```
# 1. 定位并点击网络选项
Internet_Options = driver.find_element(AppiumBy.XPATH,"//*[@text='网络和互联网']").click()
time.sleep(2)
# 2. 定位网络开关按钮
Internet_button = driver.find_element(AppiumBy.ID, "com.android.settings:id/switchWidget")
# 3. 执行各项属性检查并输出结果
print("网络开关是否可见：", Internet_button.is_displayed())
print("网络开关是否可用：", Internet_button.is_enabled())
print("网络开关是否被选中（开启状态）：", Internet_button.is_selected())
print("网络开关的文本内容：", Internet_button.text)
print("网络开关的标签名：", Internet_button.tag_name)
print("网络开关的宽高：", Internet_button.size)
print("网络开关的坐标：", Internet_button.location)
print("网络开关的宽高+坐标：", Internet_button.rect)
```

7.4.3 Appium手势操作

- 掌握Appium手势操作的方法，能够对App界面中的元素进行手势操作

7.4.3 Appium手势操作

在使用智能手机、平板电脑等移动设备时，为了保护个人隐私，通常会设置锁屏密码，如果设置数字锁屏密码，则需要通过手指按下屏幕中的数字按键，输入正确的密码完成开锁；如果设置图案锁屏密码，则需要通过手指完成按下、移动、抬起的操作，最后绘制开锁的图案。除此之外，在使用移动设备时，还会经常进行轻敲、长按、拖曳等操作，在Appium中这些操作统一称为手势操作。

1.轻敲操作

轻敲操作指的是模拟手指对某个元素或点按下并快速抬起的操作，实现轻敲操作时需要调用mobile: clickGesture方法，该方法的语法格式如下所示。

```
driver.execute_script("mobile: clickGesture", { "x": 300,  "y": 600})
```

参数 x、y 为屏幕绝对坐标；也可传入 "elementId" 对元素中心点击。

2.长按操作

长按操作是模拟手指按下元素或坐标后，等待一段时间的操作。例如，长按某个按钮一段时间后会弹出菜单。实现长按操作时需要调用mobile: longClickGesture方法，该方法的语法格式如下所示。

```
driver.execute_script(“mobile: longClickGesture”,
{“x”: 300, “y”: 600, “duration”: 2000})
```

参数 duration 为按住时长，单位毫秒；可结合 "elementId" 对元素长按。

3.滑动操作

Appium提供了2个方法实现滑动操作，这2个方法分别是mobile: swipeGesture方法和mobile: scrollGesture方法，其中scroll()方法实现的滑动操作也可以称为滚动操作。

（1）通过mobile: swipeGesture方法实现滑动操作

mobile: swipeGesture 是一种用于模拟手指在屏幕限定区域内快速滑动的手势命令。

执行时，手指会按照指定的方向与距离完成一次滑移操作，并在抬起后产生惯性滚动效果，适用于快速翻页、快速浏览列表等场景

```
driver.execute_script("mobile: swipeGesture", {   "left": 100, "top": 500, "width": 900,
"height": 200,    "direction": "down",    "percent": 0.8,    "speed": 5000})
```

其中，参数 left、top、width、height 用于定义滑动手势的可视区域，表示手指滑动操作的有效范围；

参数 direction 表示滑动方向，可选值为 "up"、"down"、"left" 或 "right"，用于控制手指滑动的朝向；

参数 percent 表示滑动距离，单位为可视区域高度或宽度的倍数，例如 0.8 表示滑动距离为区域高度的80%；

参数 speed 为可选参数，用于设置滑动速度，单位为像素/秒，数值越大滑动越快。

（2）通过mobile: scrollGesture方法实现滑动操作

mobile: scrollGesture 是一种用于模拟手指在屏幕限定区域内持续滑动的手势命令。

与快速滑动不同，该命令更适合用于需要多次滑动或平滑滚动到指定位置的场景，如长列表滚动、网页内容浏览等。

```
driver.execute_script("mobile: scrollGesture", {  "left": 100,  "top": 100,  "width": 900,  "height": 1600,  "direction": "down",  "percent": 0.5})
```

其中，参数 left、top、width、height 表示滑动区域的位置和大小；

参数 direction 表示滑动方向（up、down、left、right）；

参数 percent 表示滑动距离，单位为可视区域高度的倍数；

可选参数 speed 用于控制滑动速度，单位为像素/秒。

注意：它也支持“起点+终点”形式的参数

```
driver.execute_script('mobile: swipeGesture', { 'startX': 900,'startY': 500,'endX': 100,'endY': 500, 'duration': 500  # 0.5秒内完成滑动})
```

4.拖曳操作

mobile: dragGesture 是一种用于模拟手指在屏幕上拖拽元素或对象的手势命令。

执行时，手指会从一个起始位置按下并持续移动至目标位置，再抬起手指，整个过程无惯性，适用于拖拽排序、拖拽文件、拖拽图标等场景。

```
driver.execute_script("mobile: dragGesture", {    "startX": 100,    "startY": 500,    "endX": 300,    "endY": 500,    "steps": 50})
```

其中，参数 startX 和 startY 表示拖拽起始位置的屏幕坐标；

参数 endX 和 endY 表示拖拽目标位置的屏幕坐标；

参数 steps 为可选参数，用于控制拖拽过程的平滑程度，表示从起始位置到目标位置中间插入的步数，数值越大拖拽过程越平滑，一般建议使用 50 至 100 之间。

5.复杂路径手势

PointerInput 是一种低级手势模拟方式，可用于构建包含多个中间点、停顿、长按、曲线等复杂路径的触摸操作。

适用于需要精确控制手势轨迹的场景，如绘制图案、密码手势解锁、连续折线滑动等。

该方式通过构建 ActionBuilder 对象并添加多个触摸事件实现，语法格式如下：

7.4.3 Appium手势操作


```
pointer = PointerInput(interaction.POINTER_TOUCH, "finger")
actions = ActionBuilder(driver, mouse=pointer)
actions.pointer_action.move_to_location(240, 976)
actions.pointer_action.pointer_down()
actions.pointer_action.move_to_location(840, 1576)
actions.pointer_action.pause(0.5)
actions.pointer_action.release()
actions.perform()
```

- PointerInput：用于定义一个 “指针输入源”，这里指定为 interaction.POINTER_TOUCH（触摸类型），名称为 “finger”
- ActionBuilder：用于构建一系列连续的手势动作，将指针输入与 Appium 驱动（driver）绑定，后续的手势操作都通过它来定义
- 其中，move_to_location() 用于移动手指到指定坐标；pointer_down() 表示手指按下；pointer_up() 表示手指抬起；pause() 表示手指停留时间，单位为秒；release() 表示释放当前手势。

7.4.3 Appium手势操作

下面以雷电模拟器中的“设置”App为例，演示如何在程序中调用Appium手势操作的方法。创建Test4.py文件，然后在该文件中实现以下操作。

- 在设置界面滑动屏幕找到安全选项并点击。
- 在安全界面点击屏幕锁定。
- 在选择屏幕锁定方式界面点击图案选项。
- 在绘制解锁图案界面中绘制“L”解屏图案，等待5秒后退出雷电模拟器。

7.4.1 Appium元素定位


```
import time
from appium import webdriver
from appium.options.android import UiAutomator2Options
from appium.webdriver.common.appiumby import AppiumBy
desired_caps = dict()
desired_caps['platformName'] = 'Android'
desired_caps['platformVersion'] = '9'
desired_caps['deviceName'] = 'emulator:5554'
desired_caps['appPackage'] = 'com.android.settings'
desired_caps['appActivity'] = '.Settings'
desired_caps['automationName'] = 'UiAutomator2'  # 明确指定自动化引擎
Options = UiAutomator2Options().load_capabilities(desired_caps)
driver = webdriver.Remote("http://localhost:4723/wd/hub", options = Options)
```

7.4.1 Appium元素定位


```
# 1. 滑动屏幕找“安全”选项
driver.swipe(start_x=900, start_y=250, end_x=900, end_y=100, duration=1000)
driver.swipe(start_x=900, start_y=500, end_x=900, end_y=250, duration=1000)
time.sleep(1)
# 2. 点击“安全”选项
driver.find_element(AppiumBy.XPATH, "//*[@text='安全性和位置信息']").click()
time.sleep(2)
# 3. 点击“屏幕锁定”选项
driver.find_element(AppiumBy.XPATH, "//*[@text='屏幕锁定']").click()
time.sleep(2)
# 5. 选择“图案”锁屏方式
driver.find_element(AppiumBy.XPATH, "//*[@text='图案']").click()
time.sleep(3)  # 等待图案绘制界面加载完成
```

7.4.1 Appium元素定位


```
 # 初始化指针输入
pointer = PointerInput(interaction.POINTER_TOUCH, "finger")
actions = ActionBuilder(driver, mouse=pointer)
# 开始手势
actions.pointer_action.move_to_location(810, 540)  # 移动到起始点
actions.pointer_action.pointer_down()              # 按下
# 每次移动后暂停0.5秒
actions.pointer_action.move_to_location(810, 680)
actions.pointer_action.pause(0.5)
actions.pointer_action.move_to_location(810, 820)
actions.pointer_action.pause(0.5)
actions.pointer_action.move_to_location(950, 820)
actions.pointer_action.pause(0.5)
actions.pointer_action.move_to_location(1090, 820)
actions.pointer_action.pause(0.5)
actions.pointer_action.release()
actions.perform()
time.sleep(5)
```

谢谢