**实验：App UI 自动化测试**

**一、实验概述**

本实验使用Appium作为自动化测试工具，利用Python的Unittest单元测试框架，实现对安卓Bilibili客户端的自动化测试。

Appium是一个用Node.js编写的、暴露REST API的WEB服务器。它接受来自客户端的连接，监听命令并在移动设备上执行，答复HTTP响应来描述执行结果。在selenium的基础上增加了对手机客户端的特定操作。

Unittest是Python自带的一个单元测试框架, 它能用于编写和运行重复的测试工作。Unittest为自动化测试用例开发和执行提供了丰富的断言方法, 从而判断测试用例的通过情况并最终生成测试结果。

**二、实验目的**

1. 熟悉Appium原理以及应用；

2. 熟悉Unittest测试框架；

3. 熟悉框架设计。

**三、实验要求**

对手机应用App的UI界面进行测试：

1）编写Python代码，利用Unittest测试框架，组织测试用例运行；

2）连接真机或模拟器，对App进行安装和卸载操作；

3）编写测试用例，测试搜索商品功能；

4）编写测试用例，测试添加购物车功能；

5）运行测试，测试步骤和结果截图。

**四、实验过程**

**4.1 测试环境搭建**

**4.1.1 Anaconda**

实验环境由Anaconda进行版本管理。

版本：conda 22.9.0

下载链接：https://www.anaconda.com/download/

**4.1.2 Python**

实验中使用Python编写代码，利用Python自带的单元测试库Unittest进行测试。

版本：Python 3.10.10

本实验中的Python利用Anaconda进行下载和管理，如需单独下载可以访问https://www.python.org/downloads/release/python-31010/

**4.1.3 雷电模拟器**

实验中采用雷电模拟器模拟安卓手机系统进行测试。

系统版本：Android 9

雷电模拟器版本：9.0.43

下载链接：https://www.ldmnq.com/

**4.1.4 Android SDK**

实验中使用Android SDK 辅助模拟器与Appium通信，控制App。

版本：Android-13.0（通过 Android Studio 安装）

下载链接：<https://developer.android.google.cn/studio>

**4.1.5 JDK**

如果要使用Android功能，那么JDK也是必须的。

版本：jdk-17.0.3.7

下载链接：<https://www.oracle.com/cn/java/technologies/downloads/>

**4.1.6 Appium**

实验中使用Appium控制模拟器和测试端口进行测试。

版本：1.22.3

通过以下命令分别安装 Appium、Appium-doctor和Appium-Python-Client：

npm install -g appium

npm install appium-doctor -g

pip install Appium-Python-Client

**4.2 实验对象选择**

在本实验中，我们选取Bilibili安卓客户端作为App自动化测试的实验对象，选择Bilibli作为测试对象有以下优点：

1）合适的界面复杂度：Bilibili客户端的主界面以及各类分支界面的排版合理，不同界面中有各种类型的界面元素，适合用于模拟复杂环境和测试。

2）较少的客户端权限：Bilibili客户端所需要的设备权限较少，在模拟器上运行时相比其他客户端可以简化操作和避免信息泄露。

3）较多的随机性事件：Bilibili客户端在使用时，会出现很多“突发状况”，比如在进入客户端时会随机弹出“青少年模式”的窗口及按钮、在进入视频时会出现广告信息干扰评论区的拉取等等，这些随机性事件可以用于检验我们自动化测试脚本的鲁棒性。

4）多样的客户端功能：Bilibili客户端不仅有视频搜索功能，还有“会员购”这样的购物界面。通过Bilibili客户端，可以同时完成多个功能的测试。

**4.3 获取App信息**

**4.3.1 获取设备和App基本信息**

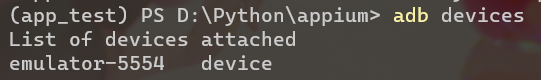
通过Appium启动App，需要在Appium中指定设备相关信息和应用相关信息。需要获取的信息包括操作系统、设备名、设备系统版本号、App包名和App主活动。

我们使用雷电模拟器在设备内的 “设置” 界面查找相关设备信息，如下图，可以得到操作系统为Android系统，操作系统版本为Andriod 9，因此版本号为9。

****

**图1** 雷电模拟器设置界面系统信息

再使用adb获取设备名，使用命令行命令adb devices获取到目前连接的所有设备，如下图的设备信息，雷电模拟器的设备名为emulator-5554 device。

****

**图2** 命令行通过adb获取设备信息

此外，还需要获取我们所要运行的App对应的包名与主活动名。首先需要在模拟器中运行对应应用，进入应用主界面（这里以Bilibili作为示例），在模拟器所在目录下使用命令行命令 adb shell dumpsys window | findstr mCurrentFocus 得到如下图结果，则如图红框标识内容即为App包名，蓝框标识内容即为App主活动名。

****

**图3** 命令行通过adb获取App信息

**4.3.2 获取App交互按钮信息**

为使后续实验中能够在Python脚本内控制App的各个控件，需要首先获取各个控件的id等相关信息。本实验使用Appium桌面版连接模拟器获取相关信息。

首先打开Appium桌面版，在设置相关信息的位置按操作系统名（platformName），设备系统版本号（platformVersion），设备名（deviceName），App包名（appPackage），App主活动名（appActivity）进行设置后，开启连接，模拟器会自动启动设定好的App并进入到主界面。以下是用于设置参数的json代码：

{

"platformName": "Android",

"platformVersion": "9",

"deviceName": "emulator-5554 device",

"appPackage": "tv.danmaku.bili",

"appActivity": "tv.danmaku.bili.MainActivityV2"

}

进入Appium桌面版的界面后，通过点击界面上相应的元素即可获得客户端元素的ID、xpath、ClassName以及继承关系等信息。

**4.4 使用Python连接Appium控制模拟器**

在 Python 脚本中调用os库可以使用系统命令，因此我们在Python中使用如下代码：先将可能正在后台运行的server端清除，以避免不必要的错误；然后打开appium，让appium使用4723端口。

os.system('adb kill-server')

os.system('appium -a localhost -p 4723')

此外，由于os.system( )会将未完成的系统调用阻塞，所以我们需要引入多线程机制，将appium挂载在后台进程，而在另一个线程中进行模块测试。

**4.5 测试Bilibili客户端**

**4.5.1 App的初始化和启动**

我们将Bilibili的driver控制端和实验中需要用到的控制函数封装为BiliOperator类，通过以下代码对driver端进行参数设置，并启动Bilibili客户端。

self.desired\_caps = {}

self.desired\_caps["platformName"] = "Android"

self.desired\_caps["platformVersion"] = "9"

self.desired\_caps["deviceName"] = "emulator-5554 device"

self.desired\_caps["appPackage"] = "tv.danmaku.bili"

self.desired\_caps["appActivity"] = "tv.danmaku.bili.MainActivityV2"

self.desired\_caps["unicodeKeyboard"] = True

self.desired\_caps["resetKeyboard"] = True

self.desired\_caps["noReset"] = True

self.desired\_caps["newCommandTimeout"] = 100000

self.host = '127.0.0.1'

self.port = 4723

self.driver = webdriver.Remote('http://localhost:4723/wd/hub', self.desired\_caps)

**4.5.2 使用Unittest自动化测试**

我们将控制App的类封装好后，使用一个类继承unittest的TestCase类，在其中设置不同的测试调用相应的控制App的类的方法，从而利用unittest进行自动化测试。

类的继承代码为：

class TestCalculator(unittest.TestCase):

在测试中用到的部分断言：

self.assertNotEqual(len(titles), 0)

self.assertNotEqual(len(descs), 0)

并在测试开始后，使用下面代码，运行unittest：

unittest.main()

**4.5.3 测试搜索功能**

在实验中，我们准备了一些关键词（如China Daily、SpaceX等），并遍历这些关键词进行搜索。此外，我们还通过桌面版Appium的Inspector窗口，提前获取了实验中所需元素的ID、xpath等属性。我们首先通过driver.find\_element选中搜索框，并且获取到搜索框的元素。通过向获取到的搜索框元素发送关键词本文，实现搜索框的输入，然后调用安卓键盘的回车键，开始搜索。进入搜索结果界面后，通过driver.find\_elements获得所有ID含有title的元素，遍历这些元素并且获取它们的text属性，从而获得搜索到的所有视频的标题。

sbox = self.driver.find\_element(By.ID, ('search\_src\_text'))

sbox.send\_keys(keyword)

time.sleep(1)

self.driver.press\_keycode(AndroidKey.ENTER)

time.sleep(3)

eles = self.driver.find\_elements(By.ID, 'title')

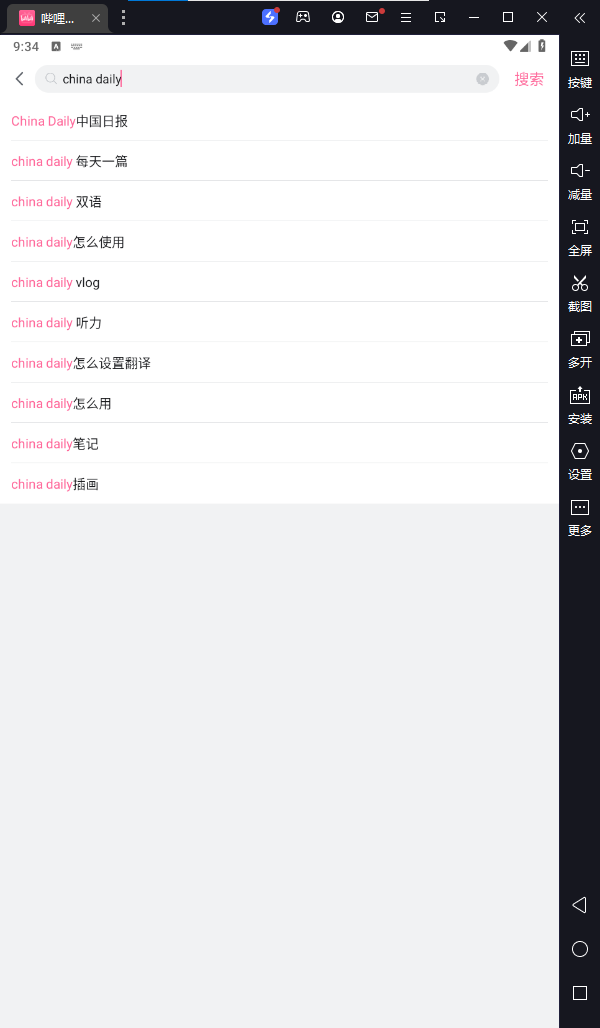
for ele in eles:

titles.append(ele.text)

time.sleep(15)

由于driver端只进行发送信号的工作，并不会检查server端对App操作的完成情况，因此在每次driver的操作后，都需要让进程阻塞一段时间，使得Appium对于App界面的屏幕操作（如点击等）、键盘操作（如回车等）和抓取操作能够充分完成。

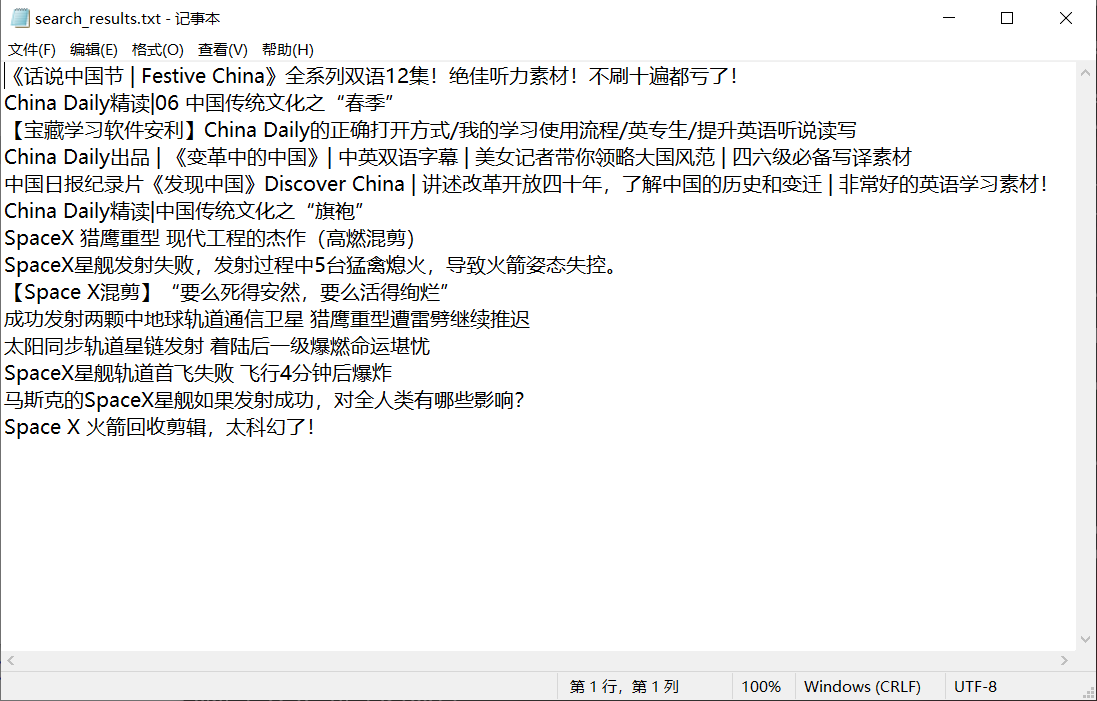
Bilibili视频搜索功能的测试如下图所示。



**图4** 视频搜索框



**图5** 视频搜索结果界面



**图6** 导出的视频搜索结果

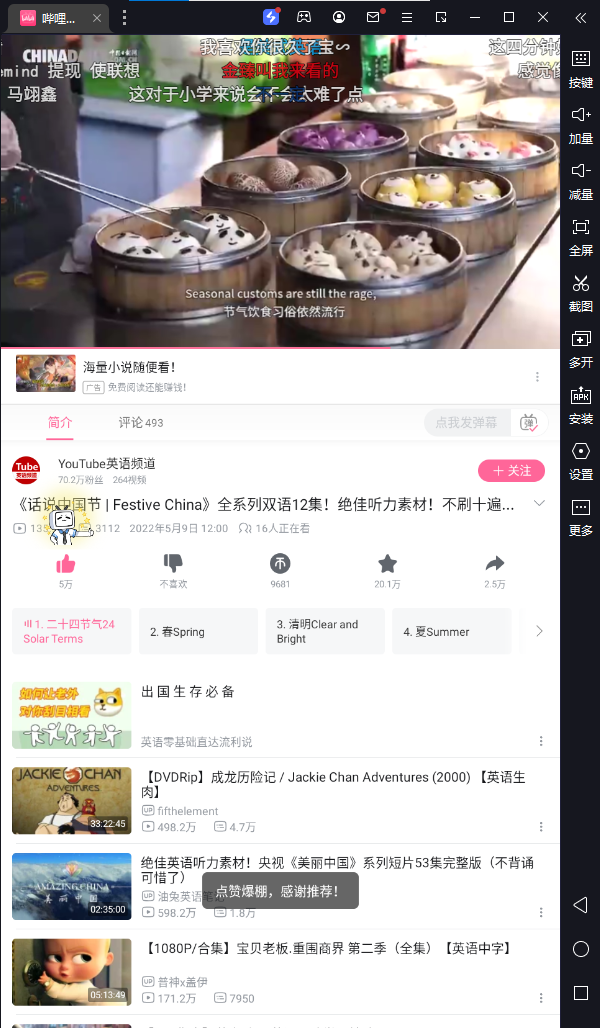
**4.5.4 测试点赞功能**

由以下代码实现点赞按钮的获取和点击：

self.driver.find\_element(By.ID, ('frame\_recommend')).click()

time.sleep(1)

点赞结果如图所示：



**图7** 自动点赞结果

**4.5.5 测试评论功能**

我们获取评论界面所有ClassName为TextView的元素，然后对元素进行处理和筛选，得到相应视频的评论区中已经加载出的评论。抓取和处理评论的代码如下：

self.driver.find\_element(By.ID, ('tab\_sub\_title')).click()

time.sleep(1)

desctext = self.driver.find\_elements(By.CLASS\_NAME, ('android.widget.TextView'))

foundcmt = False

for desc in desctext:

if not foundcmt:

if ":" in desc.text:

foundcmt = True

else:

continue

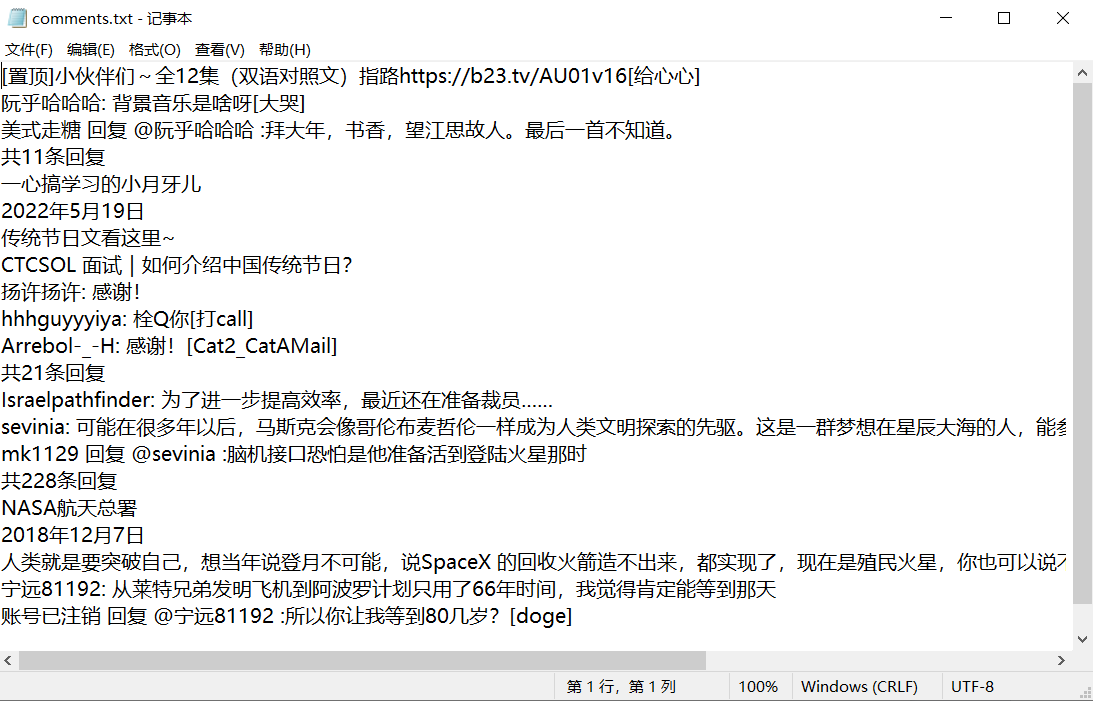
if len(desc.text) < 5:

continue

           descs.append(desc.text)

time.sleep(3)

评论的导出结果如下：



**图8** 自动导出评论结果（部分）

**4.5.6 测试会员购搜索和添加购物车功能**

除了对Bilibili的视频相关功能进行测试，我们还对Bilibili客户端的购物功能进行了自动化测试。首先用driver.press\_keycode(AndroidKey.BACK)模拟安卓机器的回退，回退至主界面后转移到Bilibili会员购界面。进入会员购界面后，通过点击搜索框并使用send\_keys(keyword)发送搜索信息，进入商品的搜索结果界面（注意，在此处不能使用 driver.press\_keycode(AndroidKey.ENTER)，经测试Bilibili客户端在会员购界面不能相应回车键）。随后，我们任意制定了一个商品的序号，选中进入。最后，通过筛选元素的方式找到购物车按钮，将该商品添加至购物车。

会员购搜索、选择商品和添加购物车的测试代码如下：

print("Searching for " + keyword + "...")

sbox = self.driver.find\_element(By.ID, ('search\_edit'))

sbox.send\_keys(keyword)

time.sleep(6)

self.driver.find\_element(By.ID, ('mall\_id\_search\_page\_actionbar\_commit')).click()

time.sleep(10)

self.driver.find\_elements(By.CLASS\_NAME, ('android.webkit.WebView'))

time.sleep(10)

eles = self.driver.find\_elements(By.CLASS\_NAME, ('android.widget.Image'))

eles[4].click()

time.sleep(8)

eles = self.driver.find\_elements(By.CLASS\_NAME, ('android.view.View'))

for ele in eles:

if u"购物车" in ele.text:

ele.click()

break

time.sleep(5)

self.driver.press\_keycode(AndroidKey.BACK)

time.sleep(1)

self.driver.press\_keycode(AndroidKey.BACK)

time.sleep(1)

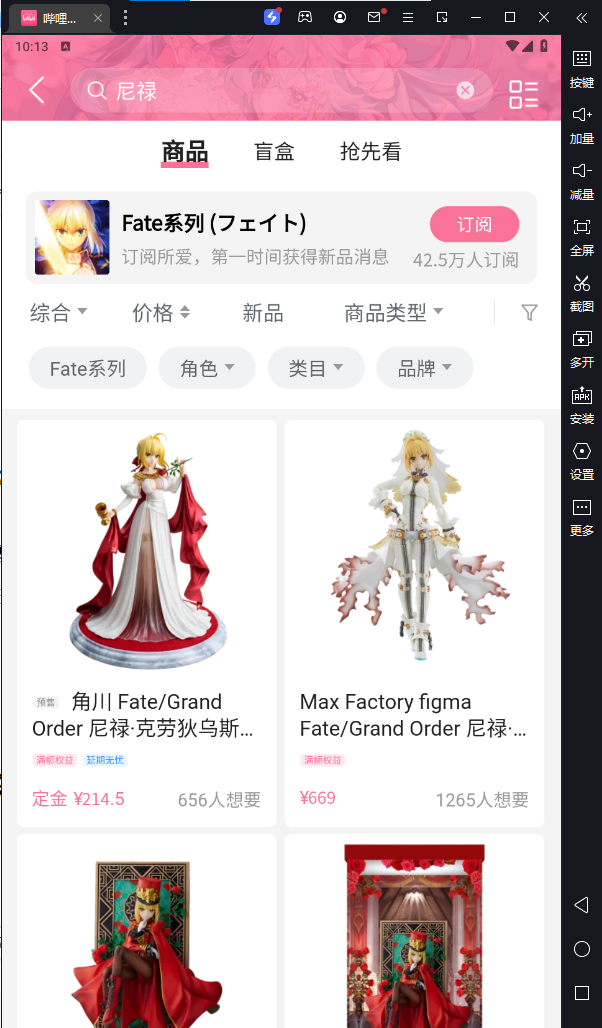
self.quit\_search()

self.quit\_buy()

Bilibili会员购的相关测试结果如下：



**图9** 搜索栏搜索商品



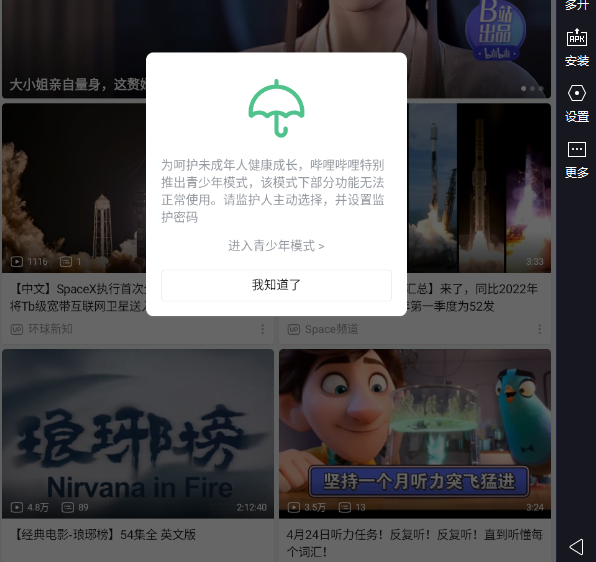
**图10** 商品的搜索结果界面

****

**图11** 添加至购物车功能

**4.5.7异常情况处理功能**

在使用Bilibili客户端的过程中，可能会随机弹出很多界面。



**图12** 青少年模式弹窗

我们的App自动化测试脚本具有对于这种异常功能的容错能力，比如对于进入界面时弹出的青少年模式选项的处理代码如下：

try:

iknow = self.driver.find\_element\_by\_id('text2')

if iknow:

           iknow = self.driver.find\_element\_by\_id('button')

           print('Adolescent protect found!')

iknow.click()

except:

print('Adolescent protect not found!')

通过在App启动时运行上述代码，使得自动化测试脚本不会因为选择青少年模式的弹窗而受到影响。

**4.6 实验中的问题和解决方式**

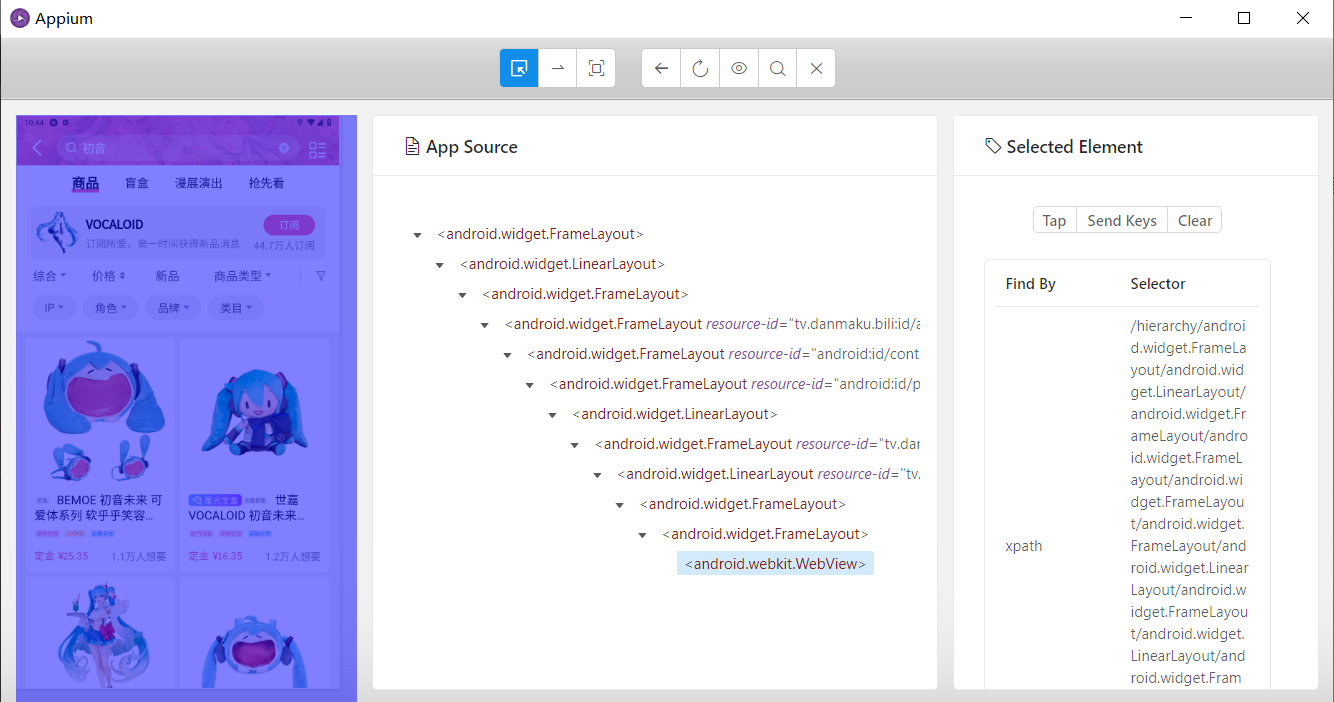
实验中，由于App本身功能复杂，我们在自动测试过程中遇到了一些问题，并采用了灵活高效的方式解决。

除了上述提到的青少年模式，Bilibili客户端在进入会员购页面是还会概率性展示一个广告页面，阻碍Appium找到所需的页面元素（存在遮挡）。我们观察到,该广告并没有退出按键，但该广告页面会自动在几秒后消失，因此选择在点击会员购按钮进入会员购分区后先等待数秒，再进行搜索操作，这样就能避免该广告页面带来的影响。



**图13** 会员购弹出广告

对于会员购，我们遇到的另一个问题是在进入搜索结果界面后，一直无法访问到界面元素导致退出。经过大量的实验，我们发现无论加载时间有多长、或是反复退出重进，都会有一个类名为WebView的List对象阻碍我们的脚本获取页面元素，因此我们使用self.driver.find\_elements(By.CLASS\_NAME, ('android.webkit.WebView'))这一段代码简单地对WebView类对象进行一次访问，使得它自动退出。



**图14** 阻碍元素访问的WebView对象

此外，我们还注意到，由于搜索功能需要访问互联网（无论是搜索视频还是搜索会员购商品），需要不定的加载时间，我们本来计划使用try关键字忙等访问，但后来发现由于访问页面元素本身较慢，以及Python调用Appium的调用情况，这样会大幅拖慢程序运行，我们采用两种方式解决：1.部分页面加载所需时间固定，采用忙等的方式，直接阻塞进程等待页面加载；2.对于加载时间不定的页面，采取等待反馈的方式，代码如下，其中func为任意反馈函数：

WebDriverWait(self.driver, timeout=10).until(func)

**五、实验总结**

通过本次实验，我们基本掌握了使用Python的单元测试框架Unittest进行自动化测试的方法，探索熟悉了使用Appium进行手机客户端自动化操作测试的流程，了解了自动化测试的规范，以及App上各类元素的构造访问格式和App的设计框架。通过编写自动化测试脚本和自动化测试程序，对手机App的自动化测试方法有了更为清晰和深刻的认识。

本实验中涉及的所有代码或文件均可在一同提交的压缩包中或者仓库https://github.com/Hibiki33/AppAutoTest中获取。本文中driver端调用的所有方法，可参考官方文档https://www.selenium.dev/zh-cn/documentation/webdriver获取详细信息。