本课时我们主要学习如何使用 Appium。

Appium 是一个跨平台移动端自动化测试工具，可以非常便捷地为 iOS 和 Android 平台创建自动化测试用例。它可以模拟 App 内部的各种操作，如点击、滑动、文本输入等，只要我们手工操作的动作 Appium 都可以完成。在前面我们了解过 Selenium，它是一个网页端的自动化测试工具。Appium 实际上继承了 Selenium，Appium 也是利用 WebDriver 来实现 App 的自动化测试的。对 iOS 设备来说，Appium 使用 UIAutomation 来实现驱动。对于 Android 来说，它使用 UiAutomator 和 Selendroid 来实现驱动。

Appium 相当于一个服务器，我们可以向 Appium 发送一些操作指令，Appium 就会根据不同的指令对移动设备进行驱动，完成不同的动作。

对于爬虫来说，我们用 Selenium 来抓取 JavaScript 渲染的页面，可见即可爬。Appium 同样也可以，用 Appium 来做 App 爬虫不失为一个好的选择。

下面我们来了解 Appium 的基本使用方法。

本节目标

我们以 Android 平台的一个示例 apk 演示 Appium 启动和操作 App 的方法，主要目的是了解利用 Appium 进行自动化测试的流程以及相关 API 的用法。

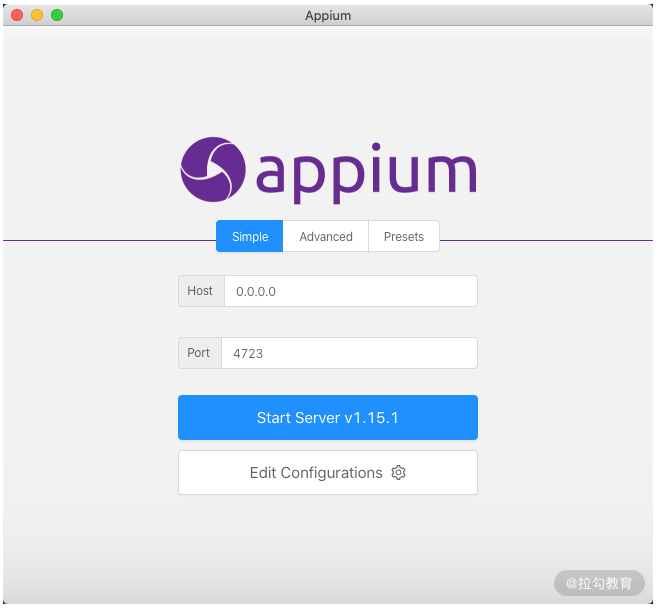
准备工作

请确保 PC 已经安装好 Appium、Android 开发环境和 Python 版本的 Appium API，安装方法可以参考 https://cuiqingcai.com/5407.html。另外，Android 手机安装好示例安装包，下载地址为：https://app5.scrape.center/。

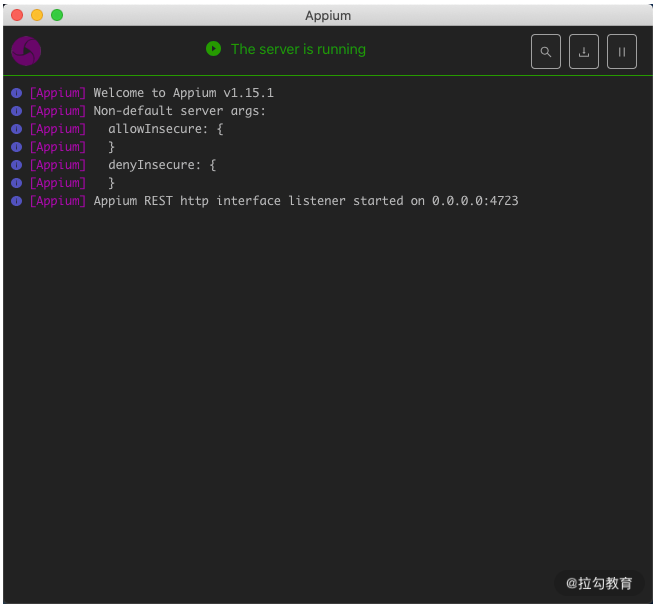
启动 APP

Appium 启动 App 的方式有两种：一种是用 Appium 内置的驱动器来打开 App，另一种是利用 Python 程序实现此操作。下面我们分别进行说明。

首先打开 Appium，启动界面如图所示。



直接点击 Start Server 按钮即可启动 Appium 的服务，相当于开启了一个 Appium 服务器。我们可以通过 Appium 内置的驱动或 Python 代码向 Appium 的服务器发送一系列操作指令，Appium 就会根据不同的指令对移动设备进行驱动，完成不同的动作。启动后运行界面如图所示。



Appium 运行之后正在监听 4723 端口。我们可以向此端口对应的服务接口发送操作指令，此页面就会显示这个过程的操作日志。

将 Android 手机通过数据线和运行 Appium 的 PC 相连，同时打开 USB 调试功能，确保 PC 可以连接到手机。

可以输入 adb 命令来测试连接情况，如下所示：

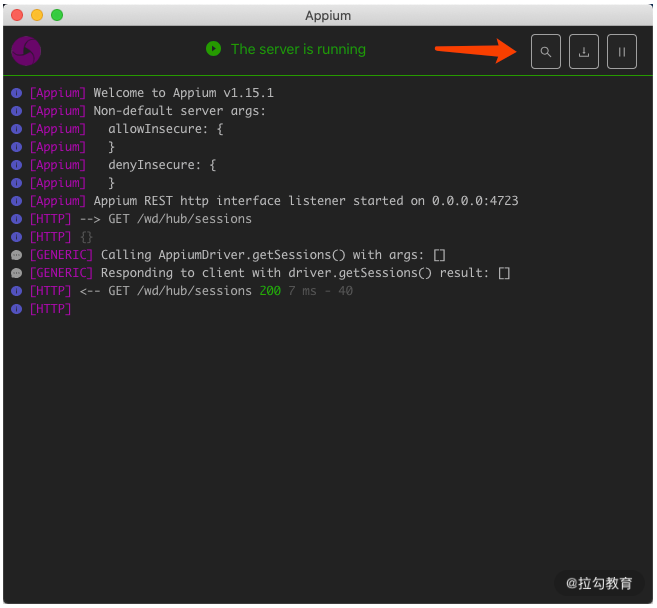
```
adb devices -l
```

如果出现如下类似结果，就说明 PC 已经正确连接手机。

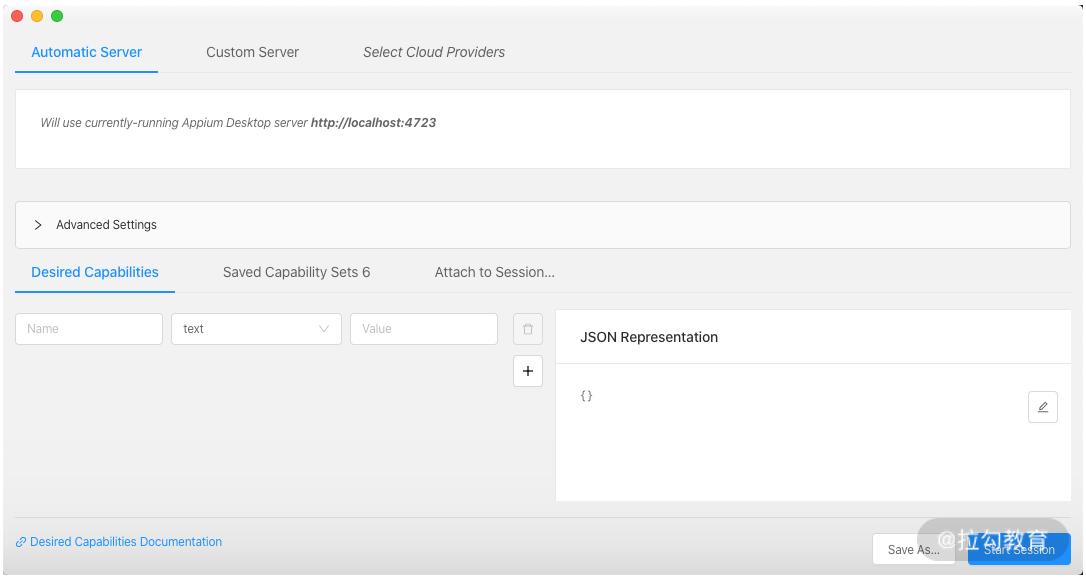
```
List of devices attached
emulator-5554          device product:cancro model:MuMu device:x86 transport_id:231
```

第一个字段是设备的名称，就是后文需要用到的 deviceName 变量。我使用的是模拟器，所以此处名称为 emulator-5554。
如果提示找不到 adb 命令，请检查 Android 开发环境和环境变量是否配置成功。如果可以成功调用 adb 命令但不显示设备信息，请检查手机和 PC 的连接情况。

接下来用 Appium 内置的驱动器打开 App，点击 Appium 中的 Start New Session 按钮，如图所示。



这时会出现一个配置页面，如图所示。

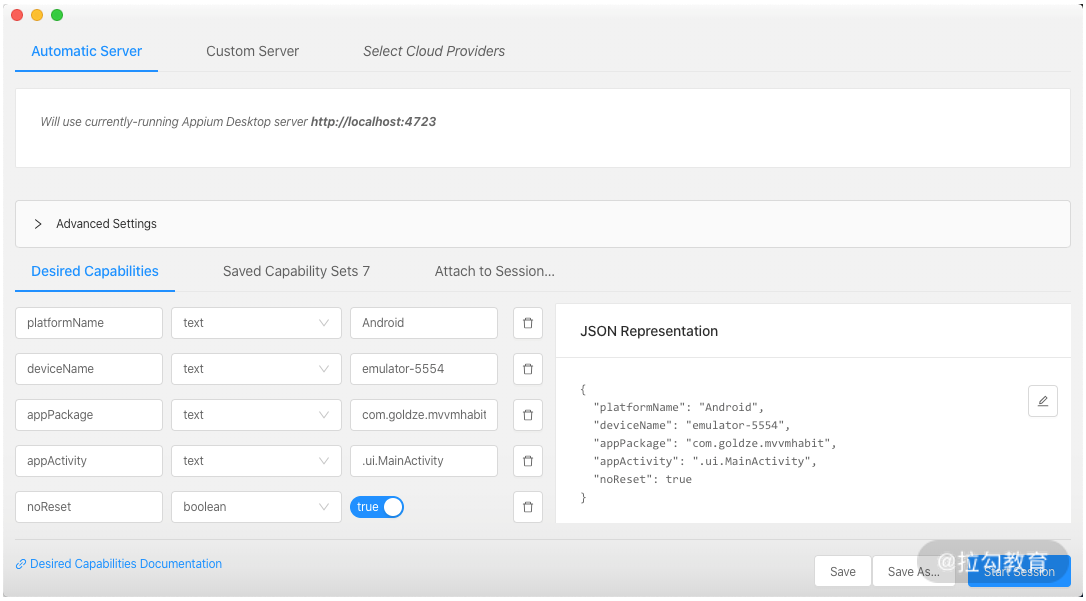


需要配置启动 App 时的 Desired Capabilities 参数，它们分别是 platformName、deviceName、appPackage、appActivity。

- platformName：平台名称，需要区分是 Android 还是 iOS，此处填写 Android。
- deviceName：设备名称，是手机的具体类型。
- appPackage：App 程序包名。
- appActivity：入口 Activity 名，这里通常需要以 . 开头。
- noReset：在打开 App 时不重置 Session，这里设置为 true。

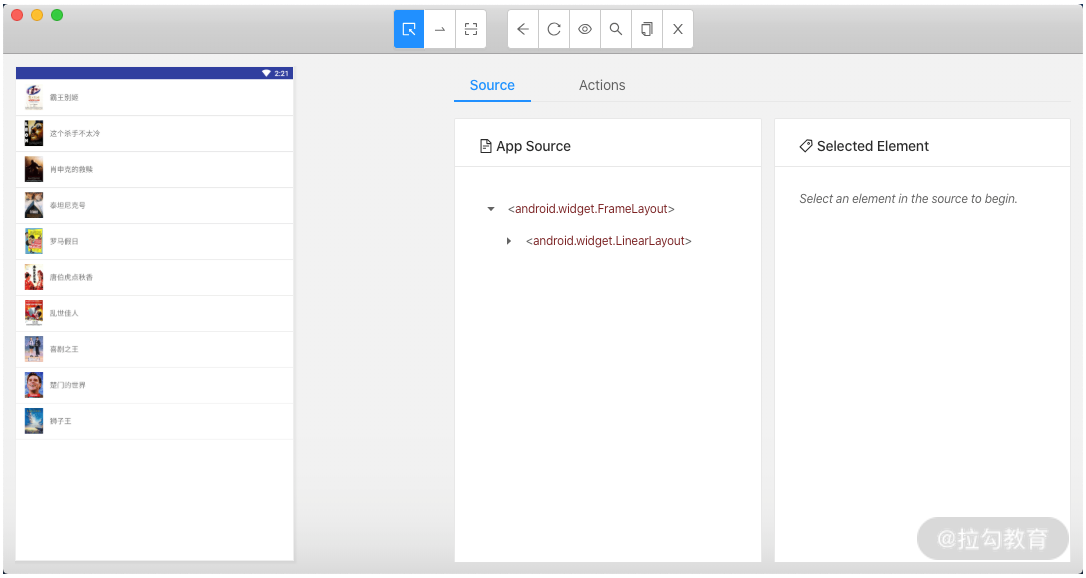
在当前配置页面的左下角也有配置参数的相关说明，链接是 https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/caps.md。

我们在 Appium 中加入上面 5 个配置，如图所示。

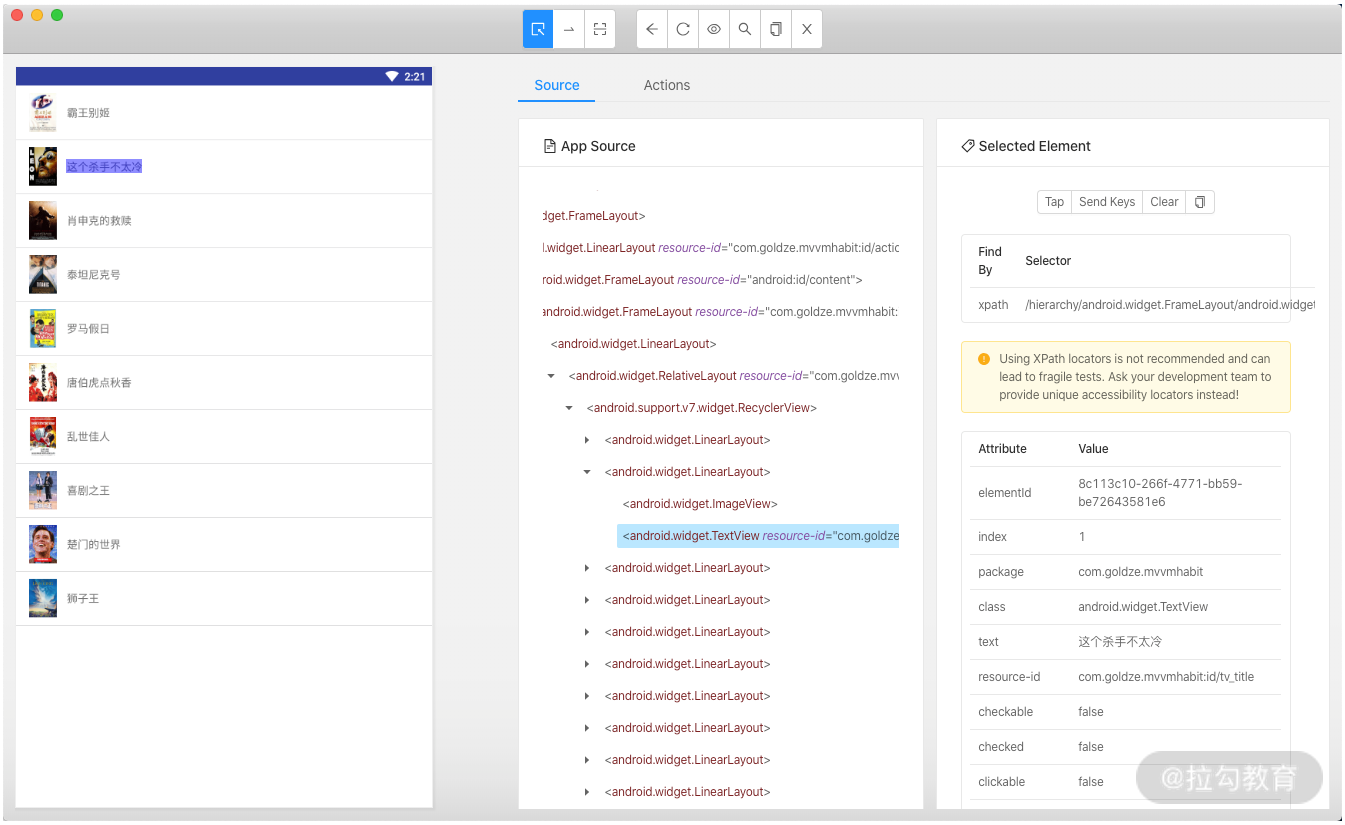


点击保存按钮，保存下来，我们以后可以继续使用这个配置。

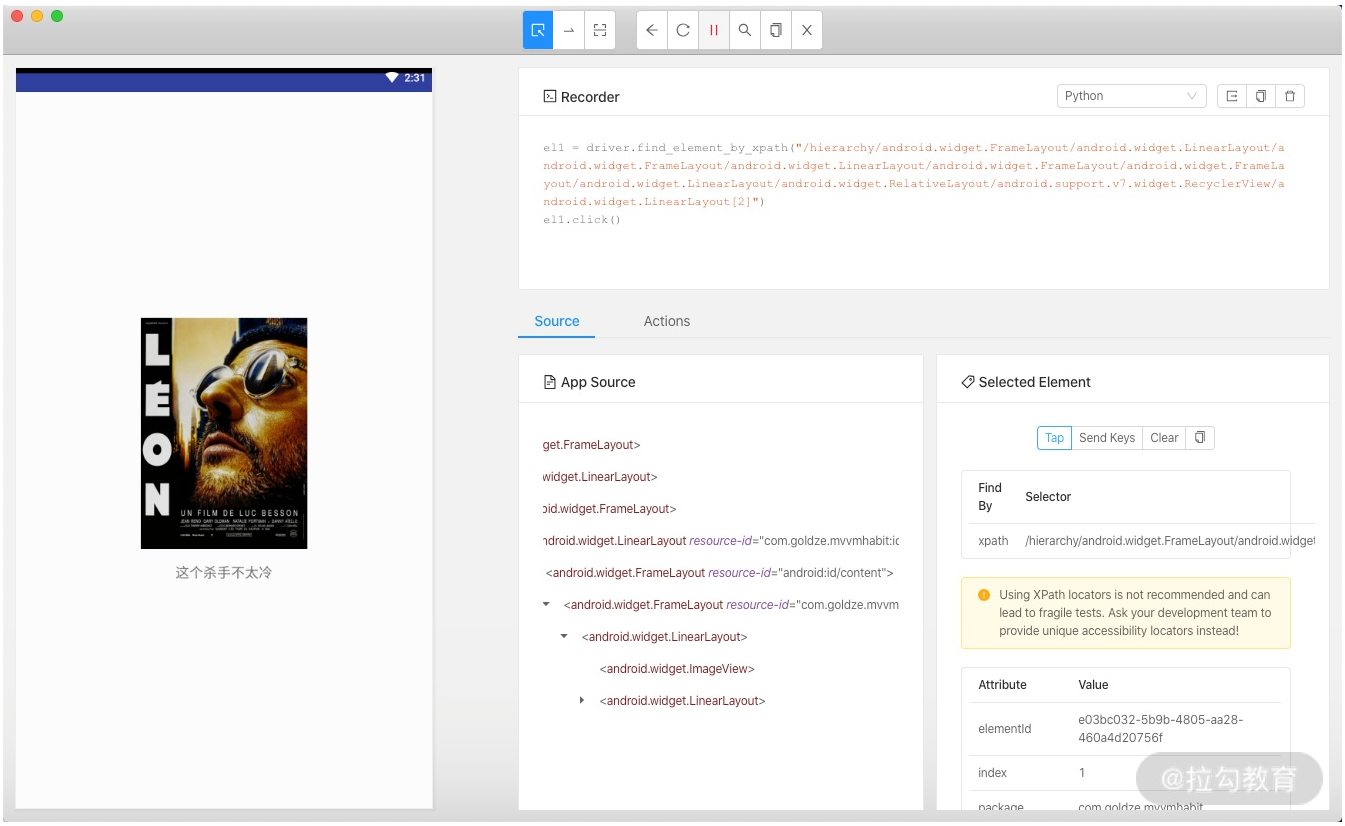
点击右下角的 Start Session 按钮，即可启动 Android 手机上的 App 并进入启动页面。同时 PC 上会弹出一个调试窗口，从这个窗口我们可以预览当前手机页面，并可以查看页面的源码，如图所示。



点击左栏中屏幕的某个元素，如选中一个条目，它就会高亮显示。这时中间栏就显示了当前选中的元素对应的源代码，右栏则显示了该元素的基本信息，如元素的 id、class、text 等，以及可以执行的操作，如 Tap、Send Keys、Clear，如图所示。



点击中间栏最上方的第三个录制按钮，Appium 会开始录制操作动作，这时我们在窗口中操作 App 的行为都会被记录下来，Recorder 处可以自动生成对应语言的代码。例如，我们点击录制按钮，然后选中其中一个条目，点击 Tap 操作，即模拟了按钮点击功能，这时手机和窗口的 App 都会跳转到对应的详情页面，同时中间栏会显示此动作对应的代码，如图所示。



我们可以在此页面点击不同的动作按钮，即可实现对 App 的控制，同时 Recorder 部分也可以生成对应的 Python 代码。

下面我们看看使用 Python 代码驱动 App 的方法。首先需要在代码中指定一个 Appium Server，而这个 Server 在刚才打开 Appium 的时候就已经开启了，是在 4723 端口上运行的，配置如下所示：

```
server = 'http://localhost:4723/wd/hub'
```

用字典来配置 Desired Capabilities 参数，代码如下所示：

```
desired_caps = {
    'platformName': 'Android',
    'deviceName': 'emulator-5554',
    'appPackage': 'com.goldze.mvvmhabit',
    'appActivity': '.ui.MainActivity'
}
```

新建一个 Session，这类似点击 Appium 内置驱动的 Start Session 按钮相同的功能，代码实现如下所示：

```
from appium import webdriver
from selenium.webdriver.support.ui import WebDriverWait

driver = webdriver.Remote(server, desired_caps)
```

配置完成后运行，就可以启动 App了。但是现在仅仅是可以启动 App，还没有做任何动作。
再用代码来模拟刚才演示的两个动作：点击某个条目，然后返回。

看看刚才 Appium 内置驱动器内的 Recorder 录制生成的 Python 代码，自动生成的代码非常累赘，例如点击某个条目然后返回的代码如下所示：

```
el1 = driver.find_element_by_xpath("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/
el1.click()
driver.back()
```

我们稍微整理修改一下，然后再加上获取文本的操作，完整的代码如下所示：

```
from appium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

server = 'http://localhost:4723/wd/hub'
desired_caps = {
    'platformName': 'Android',
    'deviceName': 'emulator-5554',
    'appPackage': 'com.goldze.mvvmhabit',
    'appActivity': '.ui.MainActivity',
    'adbExecTimeout': 200000,
}
driver = webdriver.Remote(server, desired_caps)
wait = WebDriverWait(driver, 1000)
item_element = wait.until(EC.presence_of_element_located(
    (By.XPATH, '//android.support.v7.widget.RecyclerView/android.widget.LinearLayout[2]')))
item_title_element = item_element.find_element_by_xpath('//android.widget.TextView')
text = item_title_element.text
print('text', text)
item_element.click()
driver.back()
```

运行此代码，这时即可观察到手机上首先弹出了 App，然后模拟点击了其中一个条目，然后返回了主页。同时还输出了提取到的节点内的文本。
这样我们就成功使用 Python 代码实现了 App 的操作。

API

接下来看看使用代码如何操作 App、总结相关 API 的用法。这里使用的 Python 库为 AppiumPythonClient，其 GitHub 地址为 https://github.com/appium/python-client，此库继承自 Selenium，使用方法与 Selenium 有很多共同之处。

初始化

需要配置 Desired Capabilities 参数，完整的配置说明可以参考 https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/caps.md，一般来说我们配置几个基本参数即可：

```
from appium import webdriver

server = 'http://localhost:4723/wd/hub'
desired_caps = {
    'platformName': 'Android',
    'deviceName': 'emulator-5554',
    'appPackage': 'com.goldze.mvvmhabit',
    'appActivity': '.ui.MainActivity'
}
driver = webdriver.Remote(server, desired_caps)
```

这里配置了启动 App 的 Desired Capabilities，这样 Appium 就会自动查找手机上的包名和入口类，然后将其启动。包名和入口类的名称可以在安装包中的 AndroidManifest.xml 文件获取。
如果要打开的 App 没有事先在手机上安装，我们可以直接指定 App 参数为安装包所在路径，这样程序启动时就会自动向手机安装并启动 App，如下所示：

```
from appium import webdriver
server = 'http://localhost:4723/wd/hub'
desired_caps = {
    'platformName': 'Android',
    'deviceName': 'emulator-5554',
    'app': './app.apk'
}
driver = webdriver.Remote(server, desired_caps)
```

程序启动的时候就会寻找 PC 当前路径下的 apk 安装包，然后将其安装到手机中并启动。

查找元素

我们可以使用 Selenium 中通用的查找方法来实现元素的查找，如下所示：

```
el = driver.find_element_by_id('com.package.name:id/path')
```

在 Selenium 中，其他查找元素的方法同样适用，在此不再赘述。
在 Android 平台上，我们还可以使用 UIAutomator 来进行元素选择，如下所示：

```
el = self.driver.find_element_by_android_uiautomator('new UiSelector().description("Animation")')
els = self.driver.find_elements_by_android_uiautomator('new UiSelector().clickable(true)')
```

在 iOS 平台上，我们可以使用 UIAutomation 来进行元素选择，如下所示：

```
el = self.driver.find_element_by_ios_uiautomation('.elements()[0]')
els = self.driver.find_elements_by_ios_uiautomation('.elements()')
```

还可以使用 iOS Predicates 来进行元素选择，如下所示：

```
el = self.driver.find_element_by_ios_predicate('wdName == "Buttons"')
els = self.driver.find_elements_by_ios_predicate('wdValue == "SearchBar" AND isWDDivisible == 1')
```

也可以使用 iOS Class Chain 来进行选择，如下所示：

```
el = self.driver.find_element_by_ios_class_chain('XCUIElementTypeWindow/XCUIElementTypeButton[3]')
els = self.driver.find_elements_by_ios_class_chain('XCUIElementTypeWindow/XCUIElementTypeButton')
```

但是此种方法只适用于 XCUITest 驱动，具体可以参考：https://github.com/appium/appium-xcuitest-driver。

点击

点击可以使用 tap 方法，该方法可以模拟手指点击（最多五个手指），可设置按时长短（毫秒），代码如下所示：

```
tap(self, positions, duration=None)
```

参数：

- positions，点击的位置组成的列表。
- duration，点击持续时间。

实例如下：

```
driver.tap([(100, 20), (100, 60), (100, 100)], 500)
```

这样就可以模拟点击屏幕的某几个点。
另外对于某个元素如按钮来说，我们可以直接调用 cilck方法实现模拟点击，实例如下所示：

```
button = find_element_by_id('com.tencent.mm:id/btn')
button.click()
```

这样获取元素之后，然后调用 click方法即可实现该元素的模拟点击。

屏幕拖动

可以使用 scroll 方法模拟屏幕滚动，用法如下所示：

```
scroll(self, origin_el, destination_el)
```

可以实现从元素 origin_el 滚动至元素 destination_el。
参数：

- original_el，被操作的元素。
- destination_el，目标元素。

实例如下：

```
driver.scroll(el1,el2)
```

我们还可以使用 swipe 模拟从 A 点滑动到 B 点，用法如下：

```
swipe(self, start_x, start_y, end_x, end_y, duration=None)
```

参数：

- start_x，开始位置的横坐标。
- start_y，开始位置的纵坐标。
- end_x，终止位置的横坐标。
- end_y，终止位置的纵坐标。
- duration，持续时间，毫秒。

实例如下：

```
driver.swipe(100, 100, 100, 400, 5000)
```

这样可以实现在 5s 由 (100,100) 滑动到 (100,400)。
另外可以使用 flick 方法模拟从 A 点快速滑动到 B 点，用法如下：

```
flick(self, start_x, start_y, end_x, end_y)
```

参数：

- start_x，开始位置的横坐标。
- start_y，开始位置的纵坐标。
- end_x，终止位置的横坐标。
- end_y，终止位置的纵坐标。

实例如下：

```
driver.flick(100, 100, 100, 400)
```

拖拽

可以使用 drag_and_drop 实现某个元素拖动到另一个目标元素上。

用法如下：

```
drag_and_drop(self, origin_el, destination_el)
```

可以实现元素 origin_el 拖拽至元素 destination_el。
参数：

- original_el，被拖拽的元素。
- destination_el，目标元素。

实例如下所示：

```
driver.drag_and_drop(el1, el2)
```

文本输入

可以使用 set_text 方法实现文本输入，如下所示：

```
el = find_element_by_id('com.tencent.mm:id/cjk')
el.set_text('Hello')
```

我们选中一个文本框元素之后，然后调用 set_text 方法即可实现文本输入。

动作链

与 Selenium 中的 ActionChains 类似，Appium 中的 TouchAction 可支持的方法有 tap、press、long_press、release、move_to、wait、cancel 等，实例如下所示：

```
el = self.driver.find_element_by_accessibility_id('Animation')
action = TouchAction(self.driver)
action.tap(el).perform()
```

首先选中一个元素，然后利用 TouchAction 实现点击操作。
如果想要实现拖动操作，可以用如下方式：

```
els = self.driver.find_elements_by_class_name('listView')
a1 = TouchAction()
a1.press(els[0]).move_to(x=10, y=0).move_to(x=10, y=-75).move_to(x=10, y=-600).release()
a2 = TouchAction()
a2.press(els[1]).move_to(x=10, y=10).move_to(x=10, y=-300).move_to(x=10, y=-600).release()
```

利用以上 API，我们就可以完成绝大部分操作。
更多的 API 操作可以参考 https://testerhome.com/topics/3711。

结语

本节中，我们主要了解了 Appium 的操作 App 的基本用法以及常用 API 的用法。利用它我们就可以对 App 进行可视化操作并像 Selenium 一样提取页面信息了。