在上一节课我们了解了 Appium 的用法，利用 Appium 可以方便地完成 App 的自动化控制，但在使用过程中或多或少还会有些不方便的地方，比如响应速度慢，提供的 API 功能有限等。

本课时我们再介绍另外一个更好用的自动化测试工具，叫作 airtest，它提供了一些更好用的 API，同时提供了非常强大的 IDE，开发效率和响应速度相比 Appium 也有提升。

Airtest 概况

AirtestProject 是由网易游戏推出的一款自动化测试框架，项目构成如下。

- Airtest：是一个跨平台的、基于图像识别的 UI 自动化测试框架，适用于游戏和 App，支持平台有 Windows、Android 和 iOS，基于 Python 实现。
- Poco：是一款基于 UI 控件识别的自动化测试框架，目前支持 Unity3D/cocos2dx/Android 原生 App/iOS 原生 App/微信小程序，也可以在其他引擎中自行接入 poco-sdk 来使用，同样是基于 Python 实现的。
- AirtestIDE：提供了一个跨平台的 UI 自动化测试编辑器，内置了 Airtest 和 Poco 的相关插件功能，能够使用它快速简单地编写 `Airtest`和 `Poco`代码。
- AirLab：真机自动化云测试平台，目前提供了 TOP100 手机兼容性测试、海外云真机兼容性测试等服务。
- 私有化手机集群技术方案：从硬件到软件，提供了企业内部私有化手机集群的解决方案。

总之，Airtest 建立了一个比较完善的自动化测试解决方案，利用 Airtest 我们自然就能实现 App 内可见即可爬的爬取。

本节内容

本节我们会简单介绍 Airtest IDE 的基本使用，同时介绍一些 Airtest 和 Poco 的基本 API 的用法，最后我们以一个实例来实现 App 的模拟和爬取。

这里使用的平台还是安卓平台，请确保现在你准备好了一台安卓的手机或模拟器。

Airtest 的安装

在 Airtest 的官方文档中已经详细介绍了 Airtest 的安装方式，包括 AirtestIDE、Airtest Python 库、Poco Python 库。

如果我们只使用 AirtestIDE 来实现自动化模拟和数据爬取的话是没问题的，因为它里面已经内置了 Python、 Airtest Python 库、Poco Python 库。AirtestIDE 提供了非常便捷的可视化点选和代码生成等功能，你没有任何 Python 代码基础的话，仅仅使用 AirtestIDE 就可以完成 App 的自动化控制和数据的爬取了。但是对于大量数据的爬取和页面跳转控制这样的场景来说，如果仅仅依靠可视化点选和自动生成的代码来进行 App 的自动化控制，其实是不灵活的。

进一步地，如果我们再加上一些代码逻辑的话，比如一些流程控制、循环控制语句，我们就可以实现批量数据的爬取了，这时候我们就需要依赖于 Airtest、Poco 以及一些自定义逻辑和第三方库来实现了。

所以，这里建议同时安装 AirtestIDE、Airtest、Poco。

AirtestIDE 的安装方式参见链接：https://airtest.doc.io.netease.com/tutorial/1_quick_start_guide/。

Airtest 的安装命令如下：

```
pip3 install airtest
```

Poco 的安装命令如下：

```
pip3 install pocoui
```

安装完成之后，可以在 AirtestIDE 中把 Python 的解释器更换成系统的 Python 解释器，而不再是 AirtestIDE 内置的 Python 解释器，修改方法参见 https://airtest.doc.io.netease.com/IDEdocs/run_script/1_useCommand_runScript/。

AirtestIDE 体验

在这里我以一台安卓手机来演示 AirtestIDE 的使用。

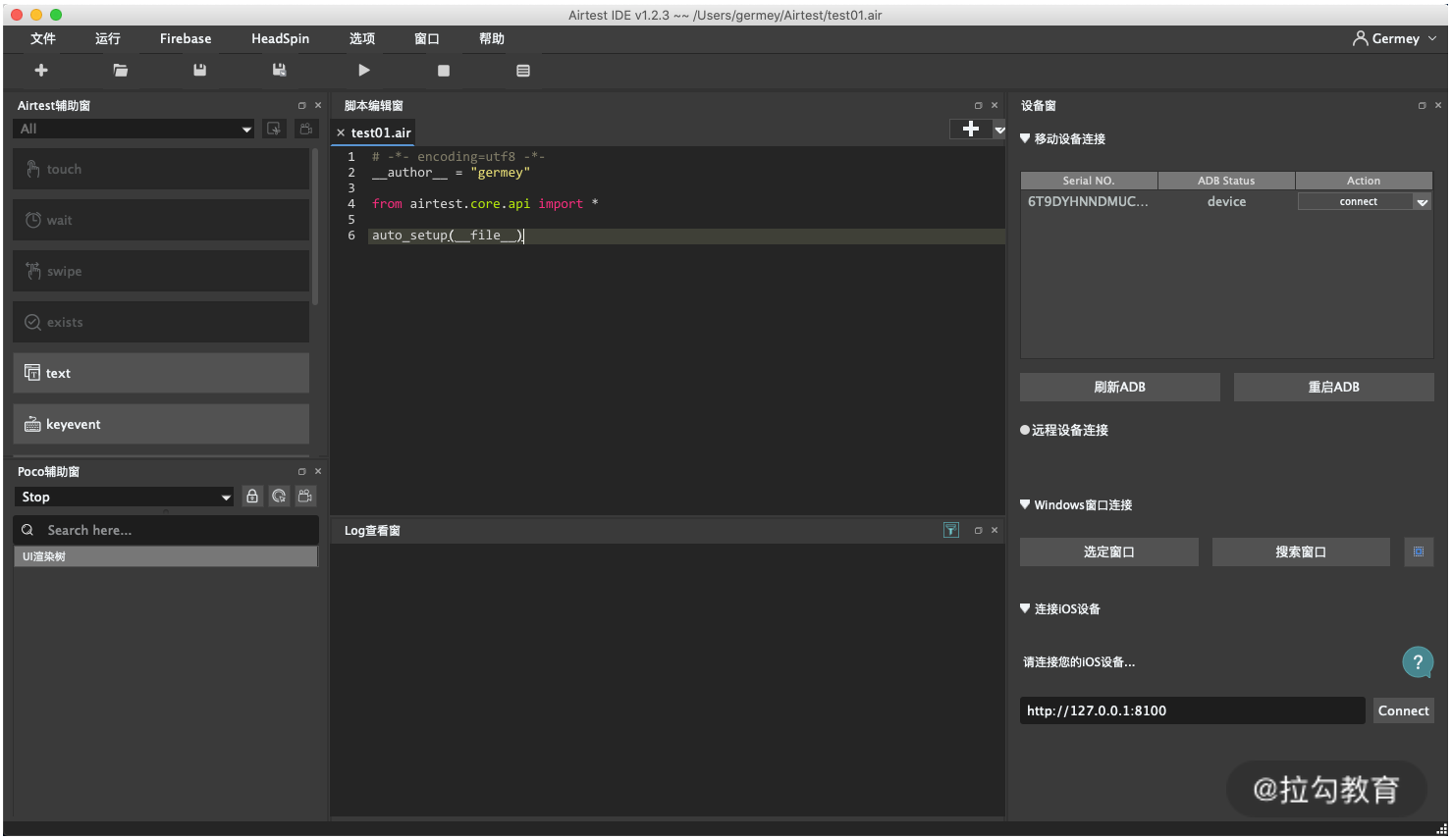
首先参考 https://airtest.doc.io.netease.com/tutorial/1_quick_start_guide/#_4 来完成手机的连接，确保使用 adb 可以正常获取到手机的相关信息，如：

```
adb devices
```

如果能正常输出手机相关信息，则证明连接成功，示例如下：

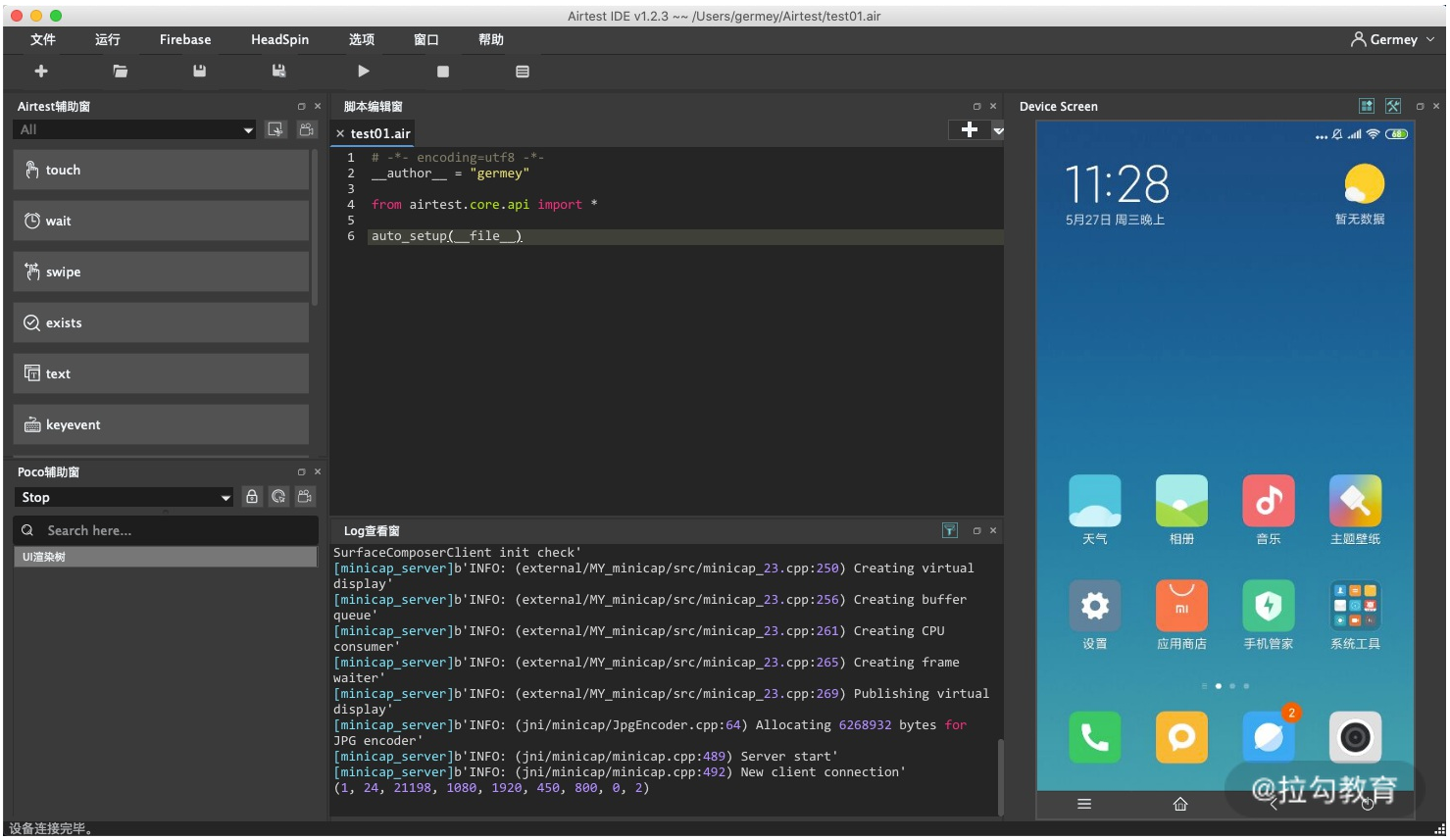
```
adb server version (40) doesn't match this client (41); killing...
* daemon started successfully
List of devices attached
6T9DYHNNDMUC8LBI    device
```

这里就能看到我的设备名称为 6T9DYHNNDMUC8LBI。
然后启动 AirtestIDE，新建一个脚本，界面如图所示：



这时候在右侧我们可以看到已经连接的设备，如果没有出现，可以查看 https://airtest.doc.io.netease.com/IDEdocs/device_connection/2_android_faq/ 来排查一些问题。

接下来我们点击设备列表右侧的 connect 按钮，就可以在 IDE 中看到手机的屏幕了，如图所示。



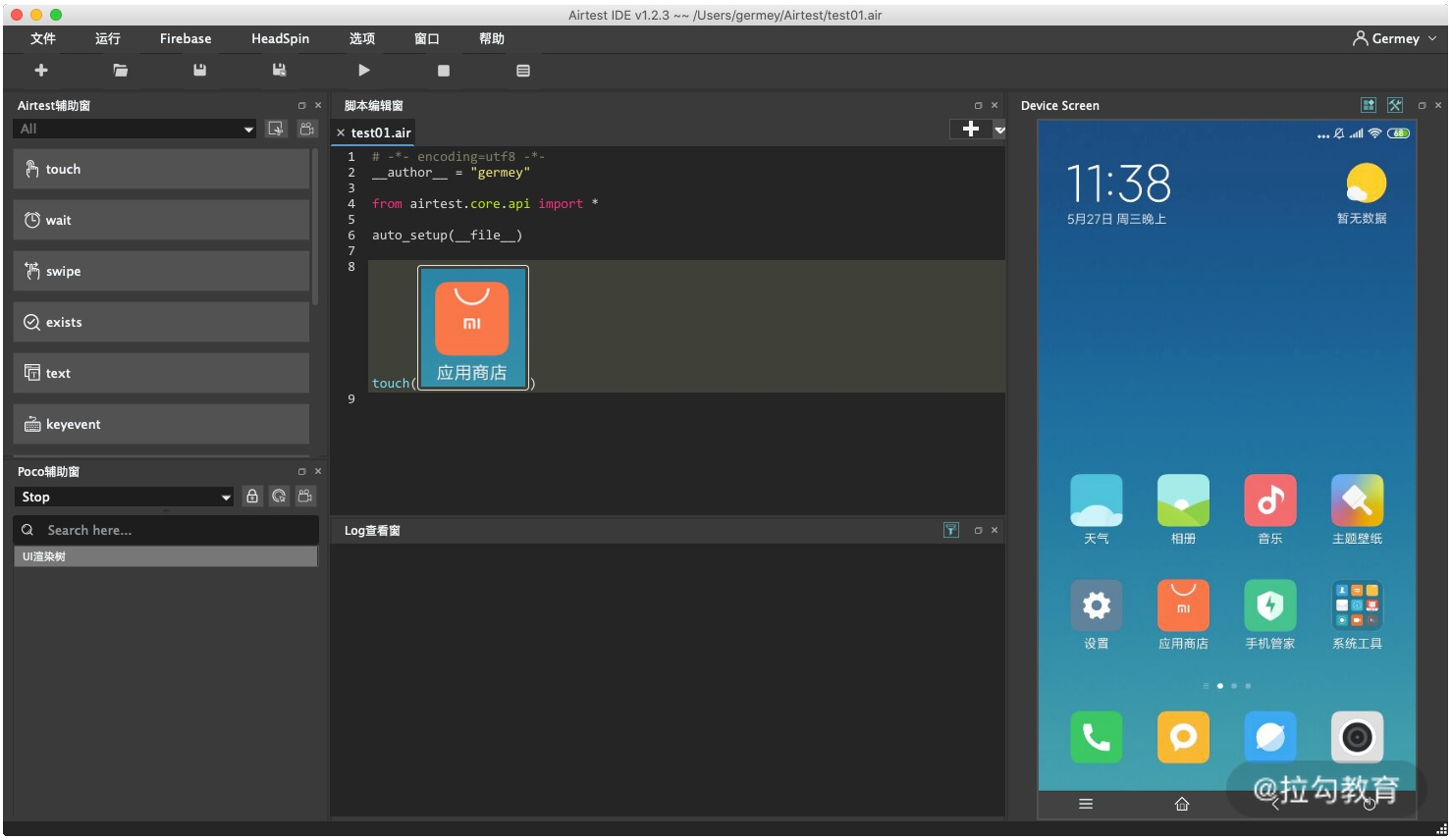
另外可以观察到，整个 IDE 被分成了三列。

- 左侧上半部分：Airtest 辅助窗，可以通过一些点选操作实现基于图像识别的自动化配置。
- 左侧下半部分：Poco 辅助窗，可以通过一些点选操作实现基于 UI 控件识别的自动化配置。
- 中间上半部分：代码区域，可以通过 Airtest 辅助窗和 Poco 辅助窗自动生成代码，同时也可以自己编写代码，代码是基于 Python 语言的。
- 中间下半部分：日志区域，会输出运行时、调试时的一些日志。
- 右侧部分：手机的屏幕。

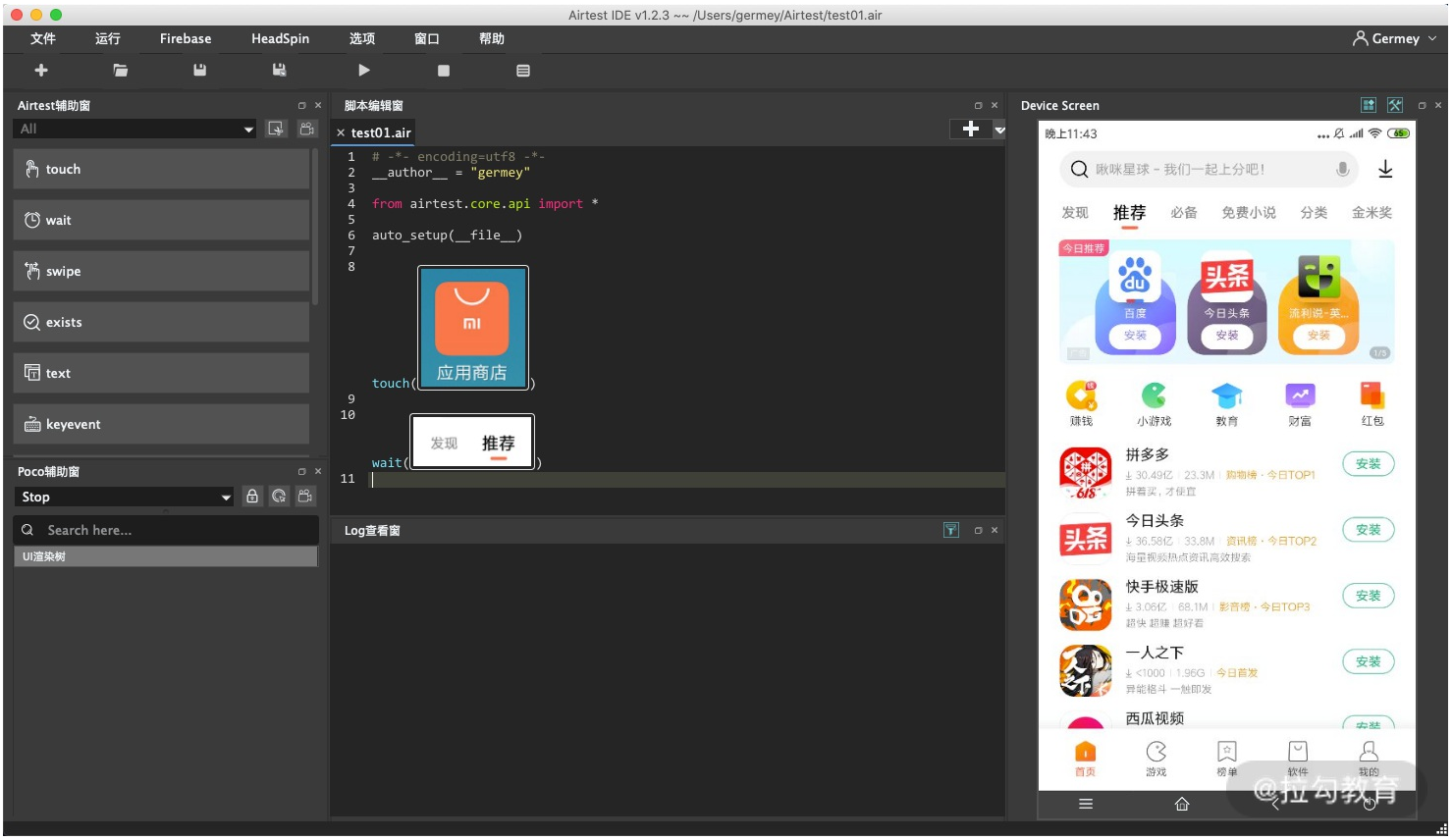
在这里我们可以通过鼠标直接点触右侧部分的手机屏幕，可以发现真机或模拟器的屏幕也会跟着变化，而且响应速度非常快。

接下来我们来实验一下 Airtest 辅助器。Airtest 可以基于图像识别来实现自动化控制，我们来体验一下。

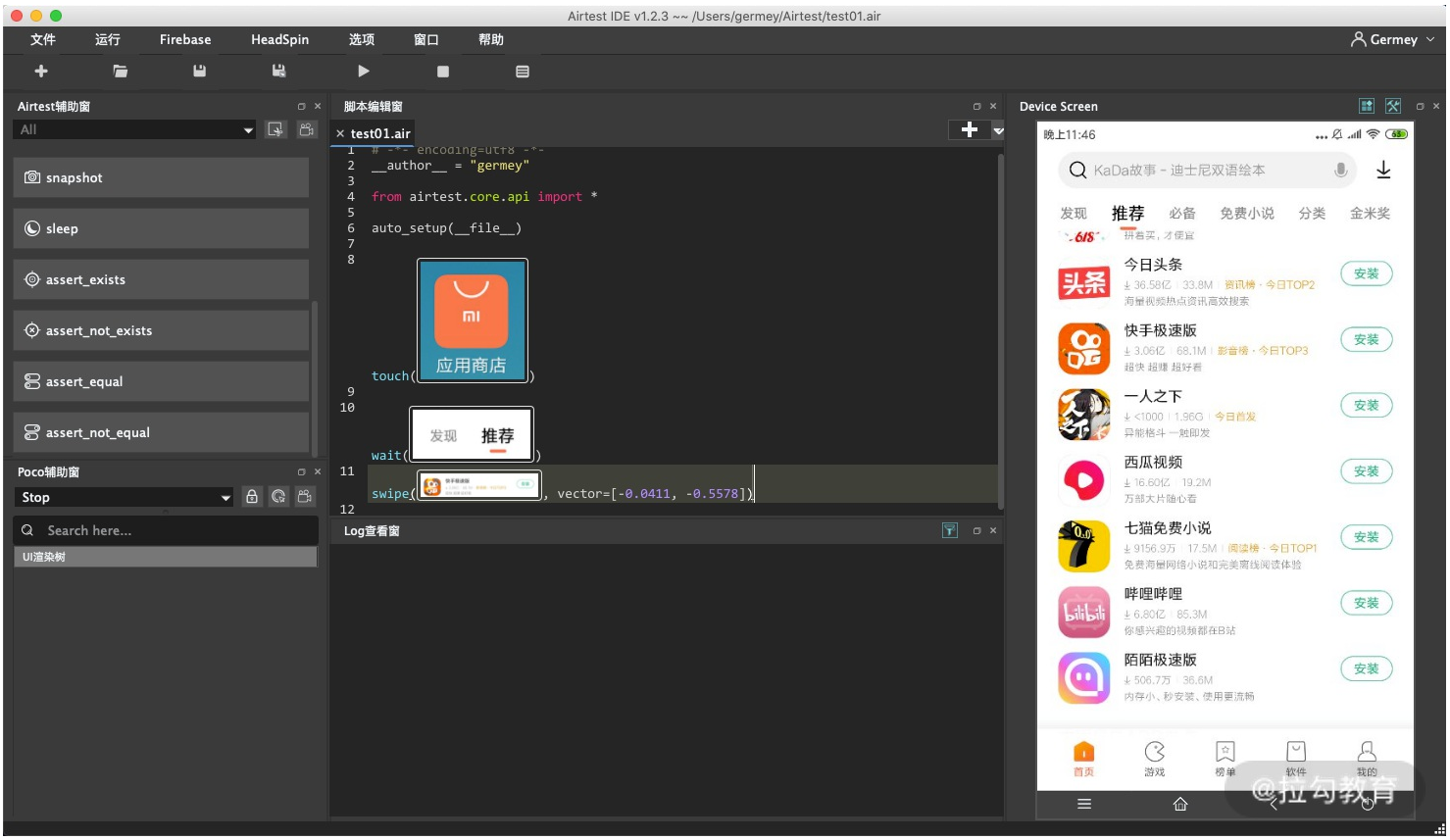
比如在这里我先点击左侧的 touch 按钮，其含义就是点击。这时候 AirtestIDE 会提示我们在右侧屏幕截图，比如这里我们截取“应用商店”，这时候我们可以发现 AirtestIDE 中便会出现了一行代码。代码的内容为 touch，然后其参数就是一张可视化的图片。



然后我们再选择 wait，其含义就是等待某个内容加载出来，同样地进行屏幕截图，如截取菜单栏的一部分，证明已经成功进入了应用商店首页。



再然后我们点击 swipe，其含义就是滑动屏幕，这时候 AirtestIDE 会提示我们先选择一个区域，再选择滑动到目标位置，如图所示。

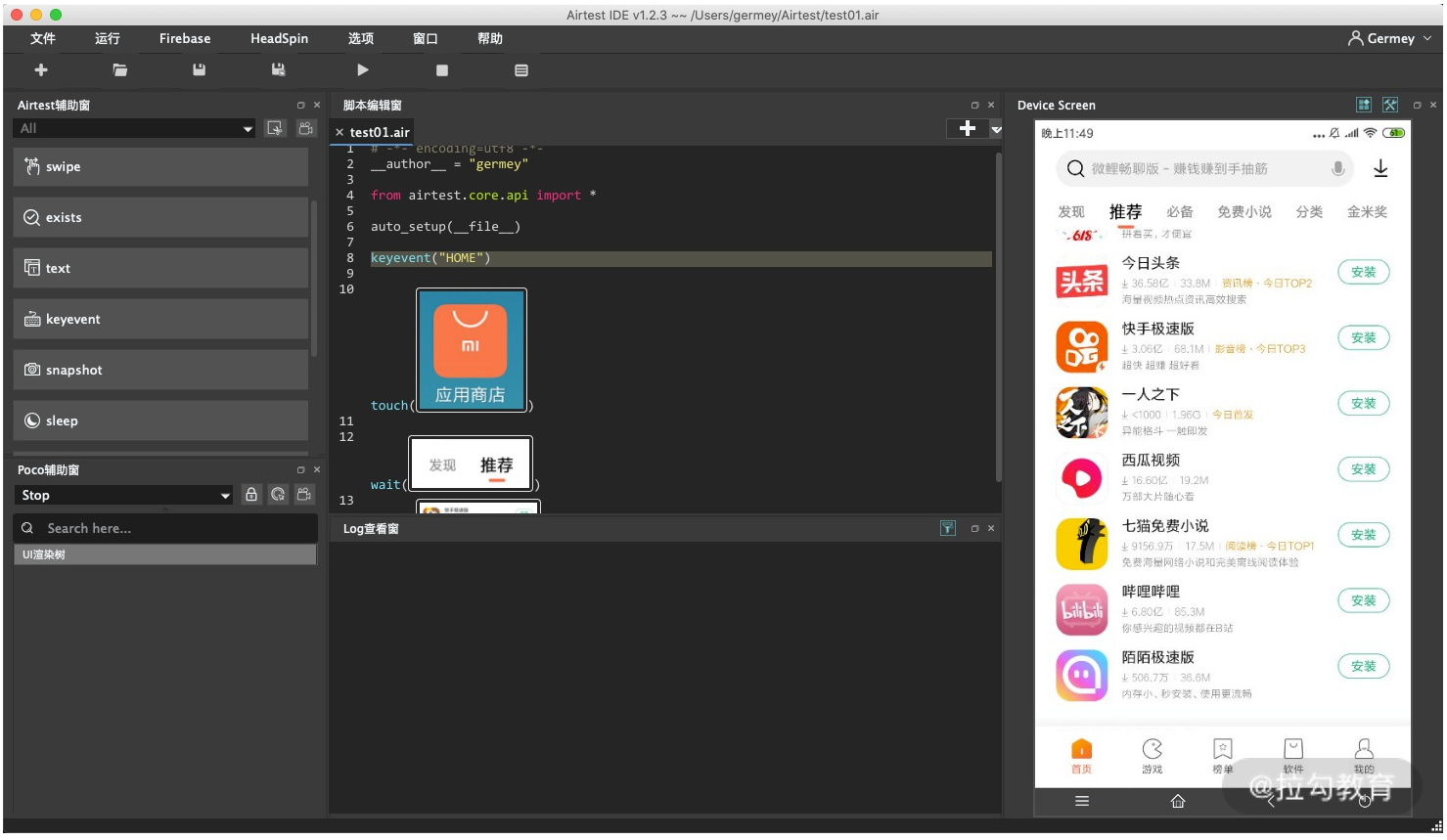


这里我们就通过一些可视化的配置完成了自动化的配置。

最后我们在代码的开头部分再加一个 keyevent，代表一些键盘事件，内容如下：

```
keyevent("HOME")
```

结果如下：



这样我们就能实现这样的自动化控制流程了：

1. 进入手机首页；
2. 点击“应用商店”；

3. 等待菜单内容加载出来；
4. 向上滑动屏幕。

怎么样，是不是很简单。如果你的手机内容和本示例不一样的话，可以灵活更换其中的配置内容。

这时候，我们点击运行按钮，即可发现 Airtest 便可以自动驱动手机完成一些自动化的操作了。以上便是 Airtest 提供的基于图像识别技术的自动化控制。

但很多情况下图像识别的速度可能不是很快，另外图像的截图也不一定是精确的，而且存在一定的风险，比如有的图像更换了，那可能就会影响自动化测试的流程。另外对于大批量的数据采集和循环控制，图像识别也不是一个好的方案。

所以，这里再介绍一个基于 Poco 的 UI 控件自动化控制，其实说白了就是基于一些 UI 名称和属性的选择器的自动化控制，有点类似于 Appium、Selenium 中的 XPath。

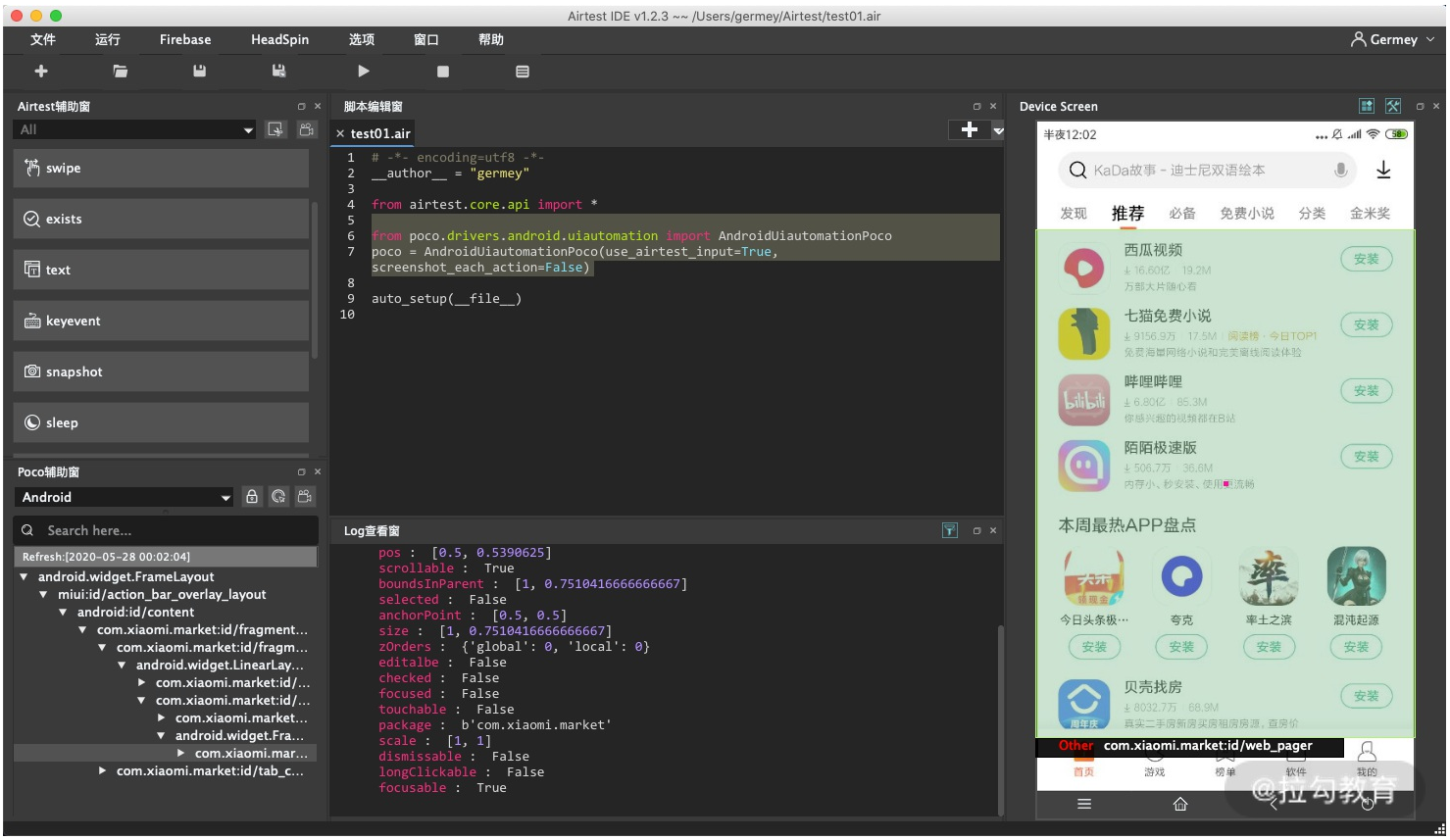
这里我们先点击左侧 Poco 辅助窗的下拉菜单，更换到 Android，这时候 AirtestIDE 会提示我们更新代码，点击确定之后可以发现其自动为我们添加了如下代码：

```
from poco.drivers.android.uiautomation import AndroidUiautomationPoco
poco = AndroidUiautomationPoco(use_airtest_input=True, screenshot_each_action=False)
```

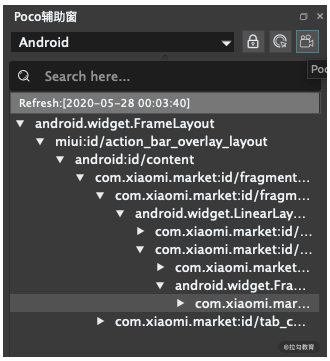
这其实就是导入了 Poco 的 AndroidUiautomationPoco 模块，然后声明了一个 poco 对象。

接下来我们就可以通过 poco 对象来选择一些内容了。

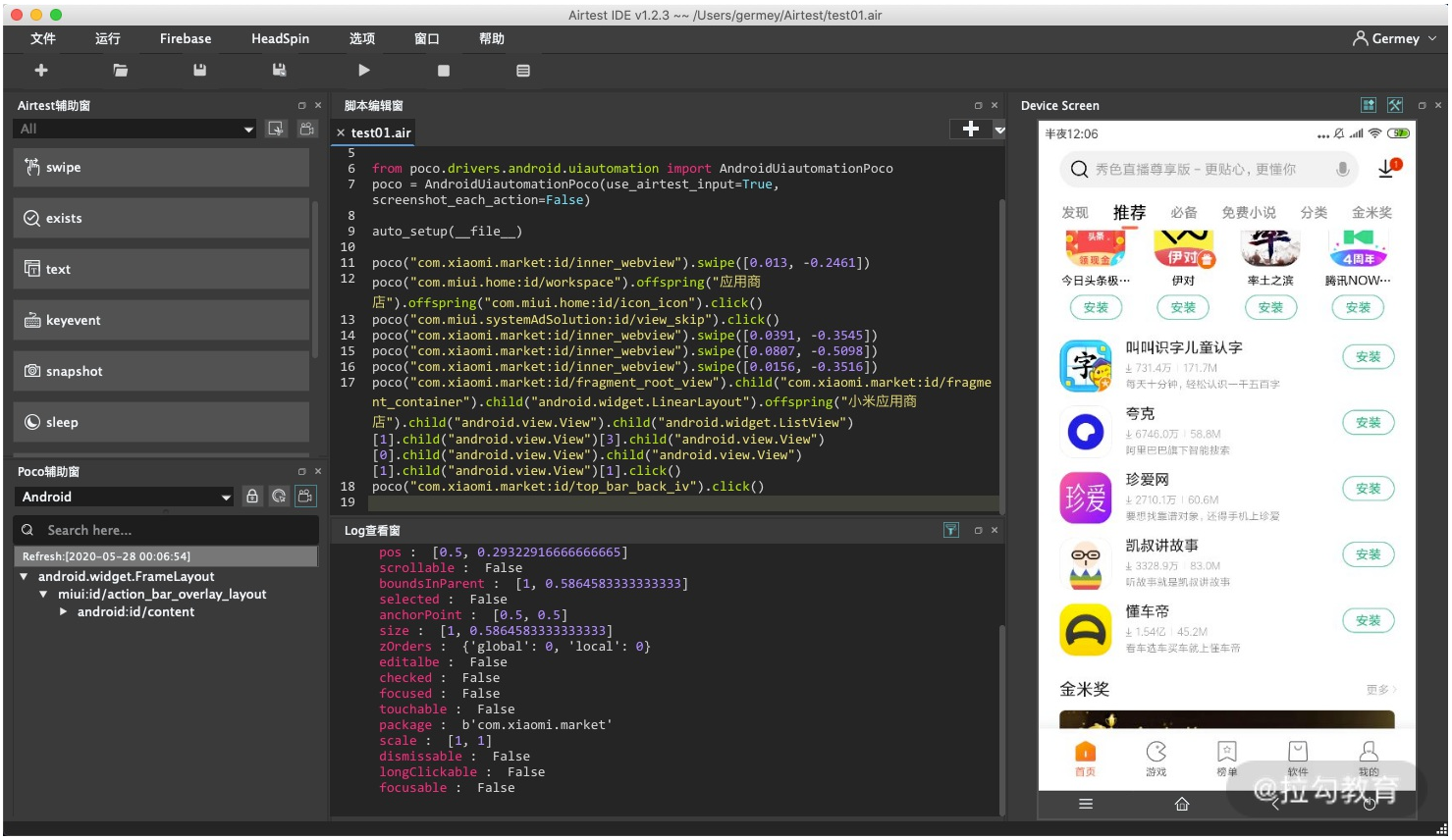
我们此时点击左侧的控件树，可以发现右侧的手机屏幕就有对应的高亮显示，如图所示。这就有点像浏览器开发者工具里面选取网页源代码，这里的 UI 控件树就类似于网页里面的 HTML DOM 树。



接着我们点击辅助窗的右上角的录制按钮，如图所示。



录制之后可以在右侧屏幕进行一些点选或滑动的一些操作，在代码区域就可以生成一些操作代码，如图所示。



这里也类似 Appium 里面录制并生成操作代码的过程。

比如这里经过我的一些操作，代码区域自动生成了如下代码：

```
poco("com.xiaomi.market:id/inner_webview").swipe([0.013, -0.2461])
poco("com.miui.home:id/workspace").offspring("应用商店").offspring("com.miui.home:id/icon_icon").click()
poco("com.miui.systemAdSolution:id/view_skip").click()
poco("com.xiaomi.market:id/inner_webview").swipe([0.0391, -0.3545])
poco("com.xiaomi.market:id/inner_webview").swipe([0.0807, -0.5098])
poco("com.xiaomi.market:id/inner_webview").swipe([0.0156, -0.3516])
poco("com.xiaomi.market:id/fragment_root_view").child("com.xiaomi.market:id/fragment_container").child("android.widget.LinearLayout").offspring("小米应用商店").child("android.view.View")
poco("com.xiaomi.market:id/top_bar_back_iv").click()
```

通过这些内容我们可以观察到有这样的规律：
poco 对象可以直接接收一个控件树选择器，然后就可以调用一些操作方法，如 swipe、click 等等完成一些操作。

另外 poco 对象还支持链式选择，如 poco 对象的调用返回结果后面紧跟了 child 方法、offspring 的方法的调用，同时还支持索引选择，其最终的返回结果依然可以调用一些操作方法，如 swipe、click 等完成一些操作。

所以，这里我们就可以初步得出如下结论：

- poco 对象支持通过传入一些 UI Path 来进行元素选择，最终会返回一个可操作对象。
- poco 对象返回的可操作对象支持链式选择，如选择其子孙节点、兄弟节点、父节点等等。

但其实可以观察到现在利用录制的方式自动生成的代码并不太规范，也不太灵活。既然已经是纯编程方式实现自动化控制，那么我们有必要来了解下 Poco 的一些具体用法。

Poco

Poco 是一款基于 UI 控件识别的自动化测试框架，目前支持 Unity3D/cocos2dx/Android 原生 App/iOS 原生 App/微信小程序，同样是基于 Python 实现的。

其 GitHub 地址为：https://github.com/AirtestProject/Poco。

首先可以看下 Poco 这个对象，其 API 为：

```
class Poco(agent, **options)
```

一般来说我们会使用它的子类，比如安卓就会使用 AndroidUiautomationPoco 来声明一个 poco 对象，这个就相当于手机操作的句柄，类似于是 Selenium 中的 webdriver 对象，通过调用它的一些选择器和操作方法就可以完成手机的一些操作。
用法类似如下：

```
poco = Poco(...)
close_btn = poco('close', type='Button')
```

这里我们可以发现，poco 本身就是一个对象，但它是可以直接调用并传入 UI 控件的名称的，这归根结底是因为其实现了一个 `__call__` 方法，实现如下：

```
def __call__(self, name=None, **kw):
    if not name and len(kw) == 0:
        warnings.warn("Wildcard selector may cause performance trouble. Please give at least one condition to shrink range of results")
    return UIObjectProxy(self, name, **kw)
```

可以看到其就是返回了一个 UIObjectProxy 对象，这个就对应页面中的某个 UI 组件，如一个输入框、一个按钮，等等。

接下来我们再看下 UIObjectProxy 的实现，其文档地址为：https://poco.readthedocs.io/en/latest/source/poco.proxy.html。

这里我们可以看到它实现了 `__getitem__`、`__iter__`、`__len__` 等方法，另外观察到其还实现了 child、children、offspring 方法，这也就是 UIObjectProxy 可以实现链式调用和索引操作以及循环遍历的原因。

接下来我们再介绍几个比较常用的方法。

child

选择子节点，第一个参数是 name，即 UI 控件的名称，如 android.widget.LinearLayout 等等，另外还可以额外传入一些属性来进行辅助选择。

其返回结果同样是 UIObjectProxy 类型。

parent

选择父节点，无需参数，可以直接返回当前节点的父节点，同样是 UIObjectProxy 类型。

sibling

选择兄弟节点，第一个参数是 name，即 UI 控件的名称，另外还可以额外传入一些属性来进行辅助选择。

其返回结果同样是 UIObjectProxy 类型。

click、rclick、double_click、long_click

点击、右键点击、双击、长按操作，UIObjectProxy 对象直接调用即可。其接受参数 focus 指定点击偏移位置，sleep_interval 代表点击完成之后等待的秒数。

swipe

滑动操作，其接收参数 direction 代表滑动方向，focus 代表滑动焦点偏移量，duration 代表完成滑动所需时间。

wait

等待此节点出现，其接收参数 timeout 代表最长等待时间。

attr

获取节点的属性，其接收参数 name 代表属性名，如 visable、text、type、pos、size 等等。

get_text

获取节点的文本值，这个方法非常有用，利用它我们就可以获得某个文本节点内部的文本数据。

另外还有很多方法，这里暂时介绍这么多，更多的方法可以参考官方文档介绍： https://poco.readthedocs.io/en/latest/source/poco.proxy.html。

实战爬取

最后我们以一个 App 为例来完成数据的爬取。其下载地址为：https://app7.scrape.center/。

首先将 App 安装到手机上，进行简单的抓包发现其数据接口带有加密，同时 App 的逆向分析也有一定的难度，所以这里我们来采取 Airtest 来实现模拟爬取。

我们的目标就是要把所有的电影名称抓取下来，如图所示：

整体思路如下：

- 由于存在大量相似的节点，所以需要用循环的方式来遍历每个节点。
- 遍历节点之后获取到其真实的 TextView 节点，利用 get_text 方法提取文本值。
- 初始数据只有 10 条，数据的加载需要连续不断上拉，因此需要增加滑动操作。
- 提取的数据可能有重复，所以需要增加去重相关操作。
- 最后加载完毕之后，检测数据量不再发生变化，停止抓取。

由于整体思路比较简单，这里直接将代码实现如下：

```
from airtest.core.api import *
from poco.drivers.android.uiautomation import AndroidUiautomationPoco
PACKAGE_NAME = 'com.goldze.mvvmhabit'
poco = AndroidUiautomationPoco()
poco.device.wake()
stop_app(PACKAGE_NAME)
start_app(PACKAGE_NAME)
auto_setup(__file__)
screenWidth, screenHeight = poco.get_screen_size()
viewed = []
current_count, last_count = len(viewed), len(viewed)
while True:
    last_count = len(viewed)
    result = poco('android.support.v7.widget.RecyclerView').child('android.widget.LinearLayout')
    result.wait(timeout=10)
    for item in result:
        text_view = item.child(type='android.widget.TextView')
        if not text_view.exists():
            continue
```

```
        name = text_view.get_text()
        if not name in viewed:
            viewed.append(name)
            print('名称', name)
    current_count = len(viewed)
    print('开始滑动')
    swipe((screenWidth * 0.5, screenHeight * 0.7), vector=[0, -0.8], duration=3)
    print('滑动结束')
    sleep(5)

    if current_count == last_count:
        print('数量不再有变化，抓取结束')
        break
```

整体思路如下：

- 首先在最开始的时候我们声明了 AndroidUiautomationPoco 对象，赋值为 poco，即获得了 App 的操作句柄。
- 接着调用了 stop_app 和 start_app 并传入包名实现了 App 的重启，确保是从头开始抓取的。
- 接着我们定义了一个无限循环，提取的是 android.support.v7.widget.RecyclerView 里面的 android.widget.LinearLayout 子节点，会一次性命中多个。
- 然后我们利用 for 循环遍历了每个节点，获取到了其中的 android.widget.TextView 节点，并用 get_text 提取了文本值，保存到 viewed 变量里面并去重。
- 遍历完成一遍之后，调用 swipe 方法滑动手机，进行上拉加载，同时滑动完毕之后等待一段时间。
- 重复以上步骤，直到 viewed 的数量不再变化，终止抓取。

运行如上代码便可以发现控制台输出了如下结果：

```
名称 霸王别姬
名称 这个杀手不太冷
名称 肖申克的救赎
名称 泰坦尼克号
名称 罗马假日
名称 唐伯虎点秋香
名称 乱世佳人
名称 喜剧之王
名称 楚门的世界
开始滑动
滑动结束
名称 狮子王
名称 V字仇杀队
开始滑动
滑动结束
名称 少年派的奇幻漂流
名称 美丽心灵
名称 初恋这件小事
名称 借东西的小人阿莉埃蒂
名称 一一
...
```

最后所有的电影名称就被我们提取出来了。

总结

以上我们便讲解了 AirtestIDE、Airtest、Poco 的基本用法，并用它们来完成了一个 App 数据的简单爬取。