UI自动化测试-Airtest-IDE-脚本编写

一、在AirtestIDE编写脚本

1. 准备工作

1）在电脑上安装好AirtestIDE

本文主要讲述如何在AirtstIDE的脚本编辑窗口编写自动化测试脚本，所以我们需要提前在电脑上安装好AirtestIDE，并且能正常打开和使用，详细教程请点击这里

2）学习一些Python基础

AirtestIDE内置了 Airtest和Poco 2个自动化测试框架，它们都是Python第三方库，因此在进行脚本的编写时，需要有 一定的Python基础

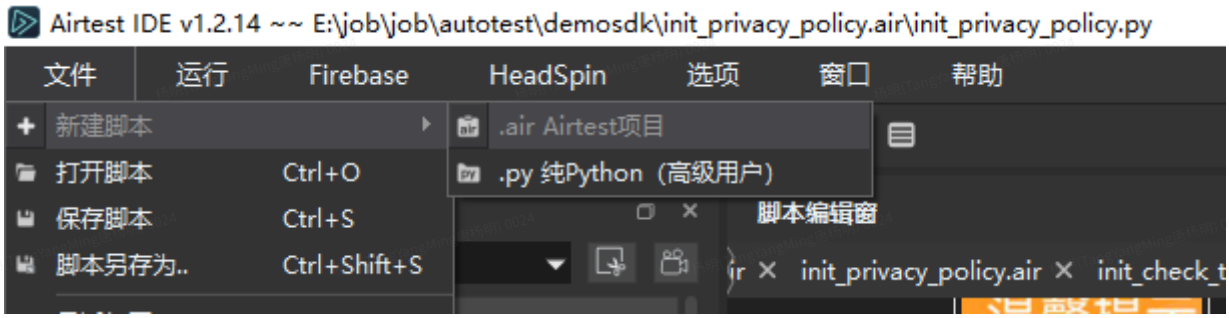
在脚本编写过程中，可以根据需求在里面混合使用 `Airtest` 和 `Poco` ，也可以自由加入自己想用的其他python第三方库完成更加强大的功能。

在开始脚本编写工作前，可以先读一读官网提供的 教程 ，可以帮助你更快地上手。

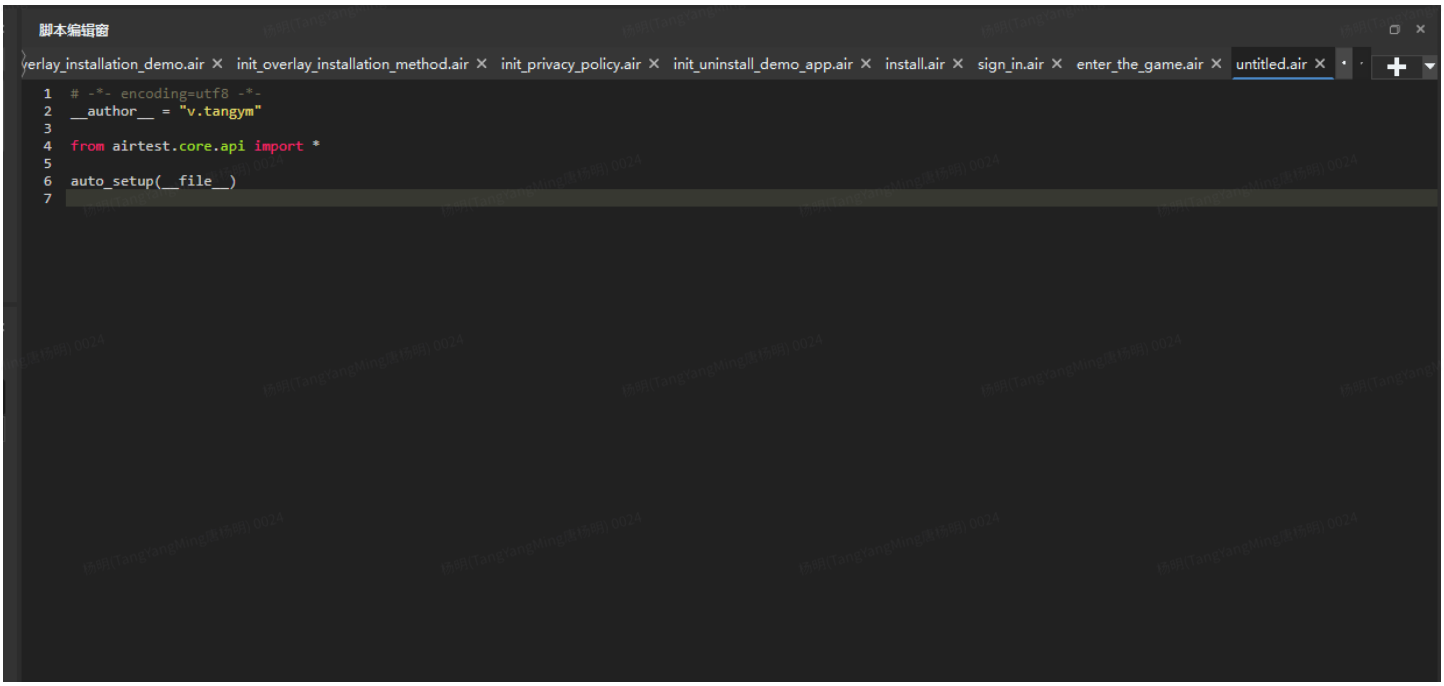
2. 在AirtestIDE上新建脚本

1）新建 .air 脚本（常用）

打开你的IDE，点击左上角的 `文件--新建脚本--.air Airtest项目` ，即可新建一个 `.air` 脚本。

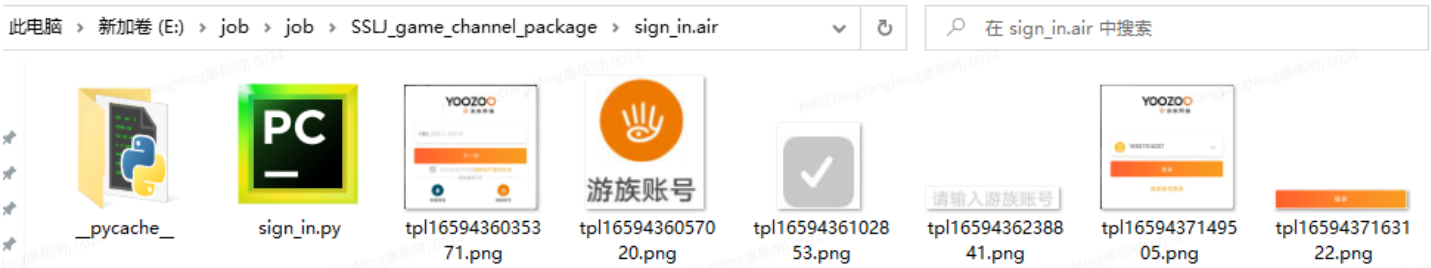


当你的 `.air` 脚本新建成功时，就能看到默认的初始化代码：



我们只需要新建 `.air` 脚本，而不是 `.py` 纯Python脚本，因为在IDE中运行 `.air` 脚本时，会自动帮我们连接设备窗口当前连接的设备，并且自动保存log内容，方便我们后续一键生成可视化的测试报告。但新建的 `.py` 纯Python脚本则不会自动帮我们处理这些，我们需要一项项在新建纯Python脚本时配置好，或者手动在脚本里面编写好，最终才能让脚本按照我们的期望跑起来并生成测试报告。

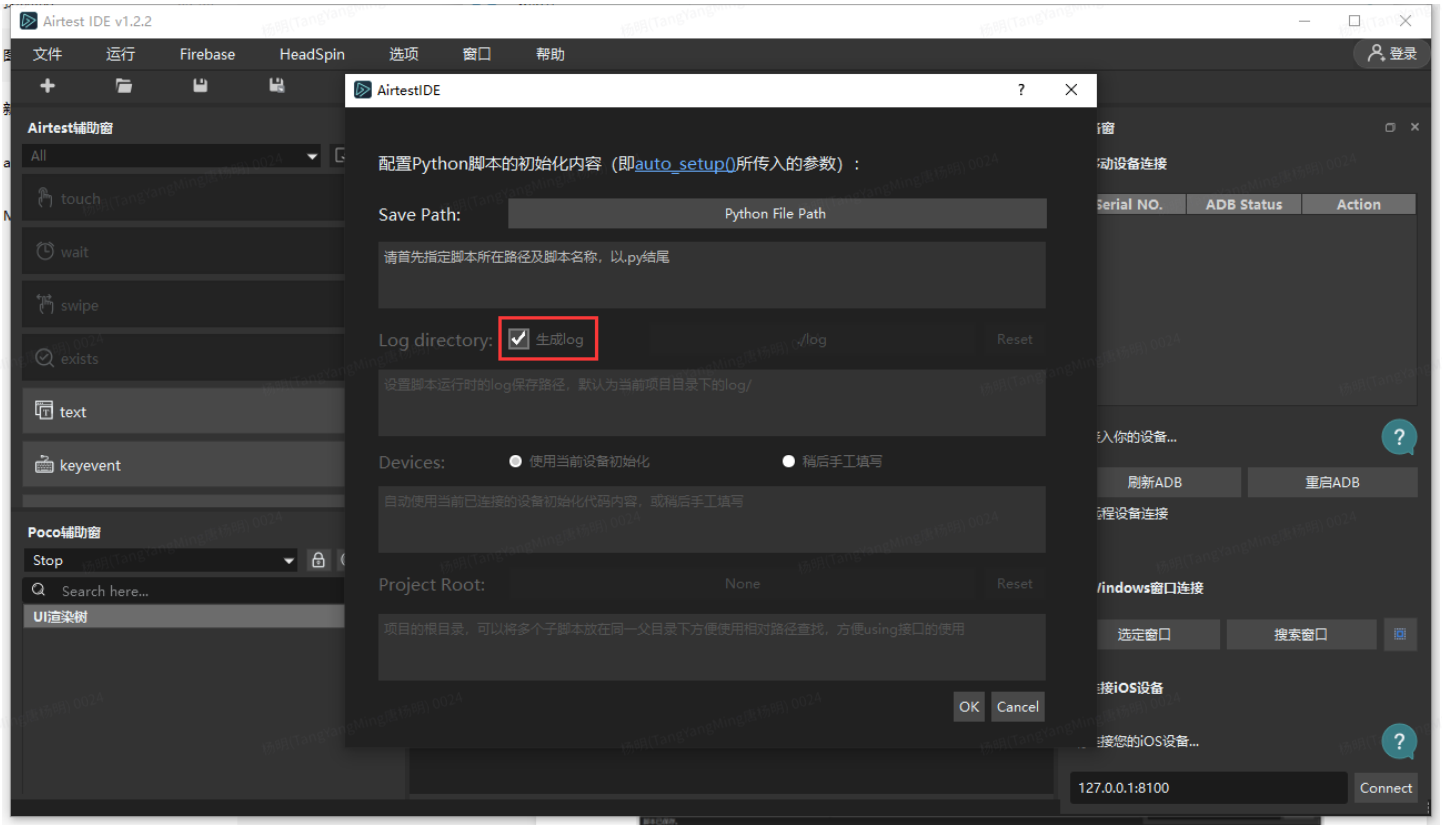
`.air` 脚本的初始化代码帮助我们从api中引入了Airtest的各个接口以及自动初始化设备。实际上，`.air` 脚本是一个文件夹，里面存放了与 `.air` 同名的 `.py` 文件，以及相关的图片文件。在运行脚本时，实际上依然使用了python调用了里面的 `.py` 文件，因为Airtest本质上是一个Python的第三方库。



2）新建 .py 纯Python脚本（了解）

点击左上角的 `文件--新建脚本--.py 纯Python（高级用户）`：

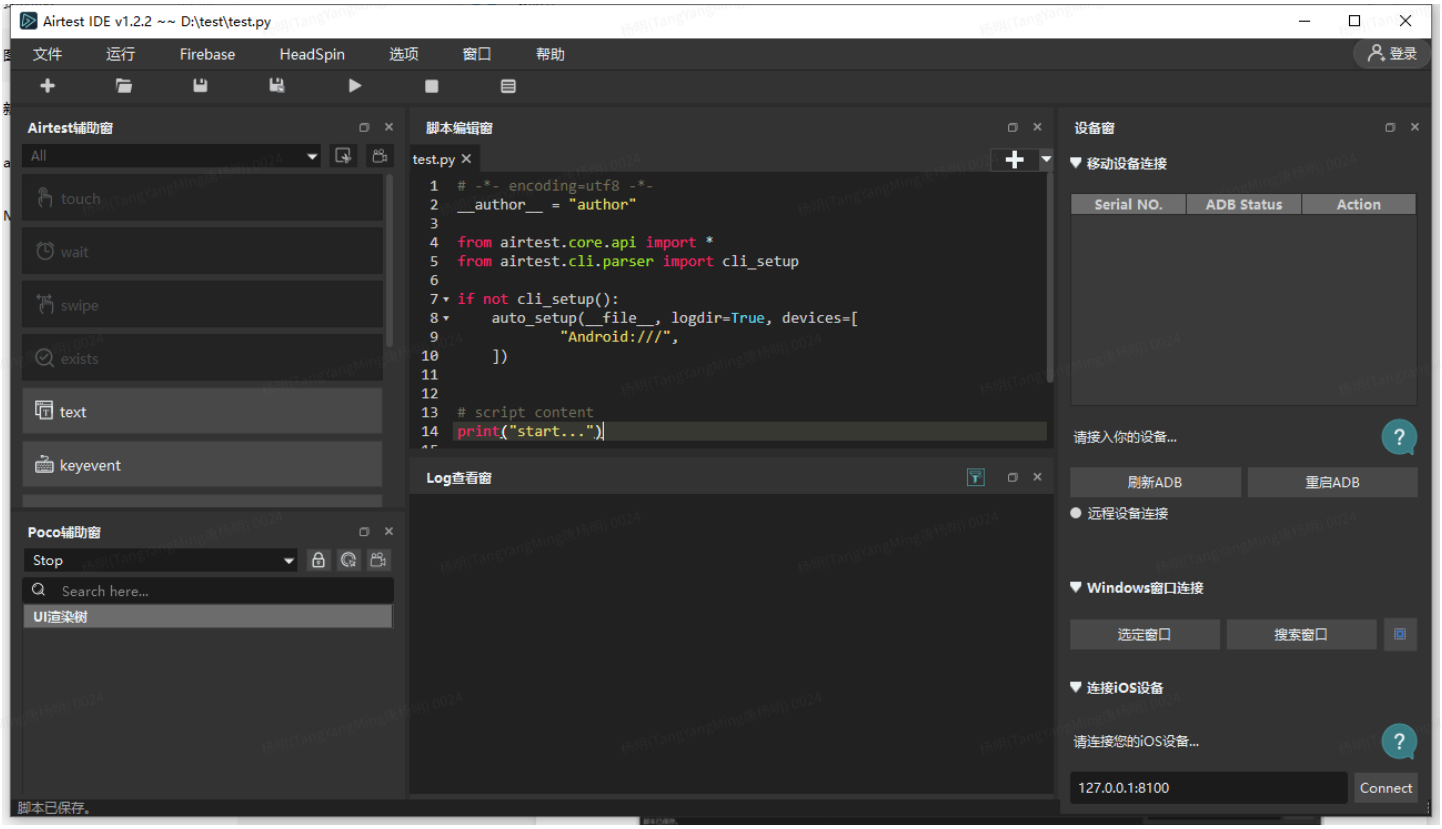
随后会弹出python文件的设置窗口：



里面的配置项有：

- Save Path： 文件保存路径，必填项，选择好路径后才能继续配置其他选项
- Log Directory： 脚本运行时的log保存路径，默认为.py文件所在目录下的 `log/` 目录
- Devices： 可以选择自动使用当前已连接的手机设备对devices参数进行初始化，也可以以后再填写
- Project Root： 项目的父目录，方便未来使用 `using` 语句引入同一父目录的其他子文件夹

在配置完所需要的选项后，点击ok按钮，将会自动以刚才的配置内容新建一份模板python文件，接下来只需要像编写一个普通 `py` 文件那样来编写脚本就可以了。当你的 `.py` 脚本新建成功时，也同样可以看到设置的一些初始化代码：



上图中这段初始化代码的意思是说，当使用 `python xxx.py` 来运行本文件，不带任何命令行参数时，则自动使用 auto_setup 这个接口来对Airtest相关的参数进行初始化。这样只需要在写py脚本时，填写好期望的参数就能直接用 `python xx.py` 指令来运行脚本。而原先的 `Airtest run xx.air --devices xx` 也不受影响，只要脚本检测到传入了命令行参数，就依然优先使用命令行参数来初始化Airtest。

3. 在AirtestIDE编写脚本

这里将以在AirtestIDE上新建 `.air` 脚本来演示如何编写自动化脚本并且设备处于已连接待测状态：

1）编写Airtest脚本

① Airtest脚本的初始化

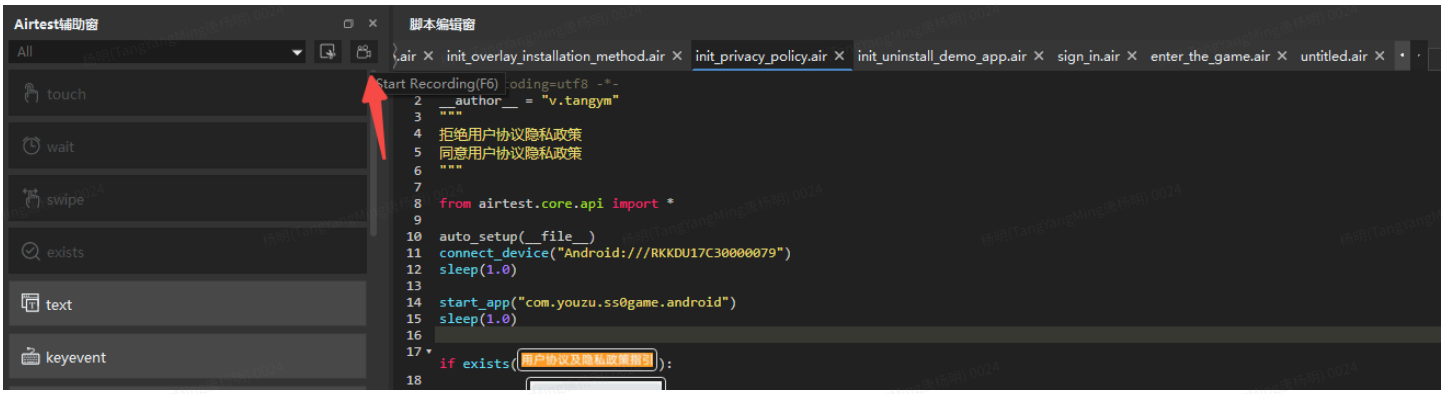
上文我们有提到，在IDE新建 `.air` 脚本时，会自动帮我们插入初始化代码：

```
# -*- encoding=utf8 -*-
__author__ = "v.tangym"

from Airtest.core.api import *

auto_setup(__file__)
```

这段代码已经帮我们引入了Airtest 的核心API，并且 `auto_setup` 会帮我们处理一些脚本初始化的内容以及连接设备窗口当前连接的设备，所以一些常用的Airtest接口我们就可以直接开始编写了

② Airtest的录制脚本功能

有一些操作，可以通过Airtest辅助窗提供的录制脚本功能，来录制我们期望的操作



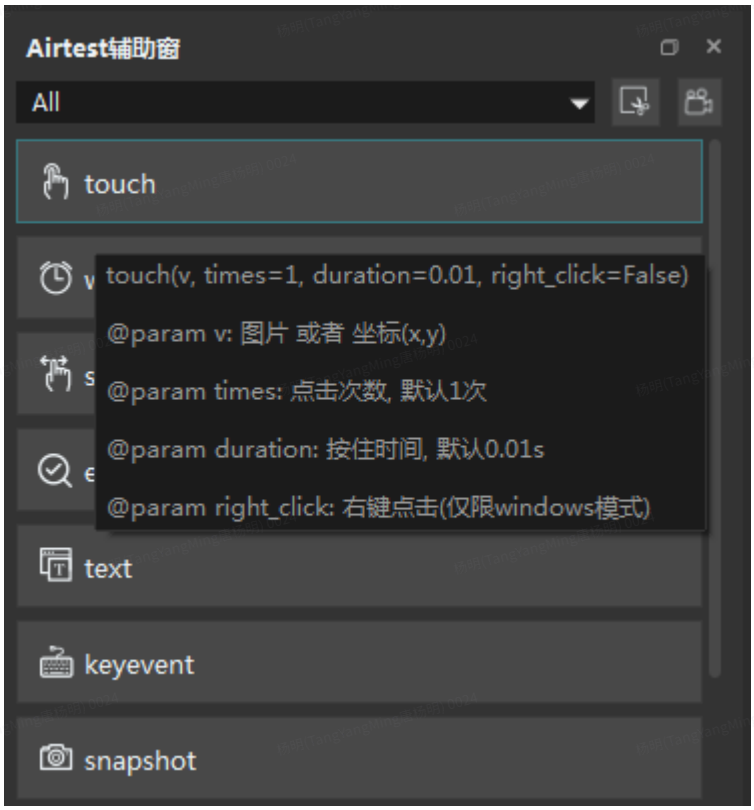
这个录制脚本并不完全是正确/最优的脚本，有时候录制脚本自动生成的截图，并不能很好地回放出来我们想要的操作，这时候我们可以通过Airtest辅助窗的其它快捷键，修改错误的操作，实现更加准确和更加丰富的脚本

③ Airtest辅助窗的其余功能

需要注意的是，Airtest辅助窗的API快捷键，只有在IDE的设备窗口连接上设备之后，才是可以使用状态，否都是置灰的不可用状态 （关于如何在AirtestIDE连接我们的待测设备，可以参考官方 设备连接章节 的内容，这里不再重复说明）。另外这里仅仅提供了部分的核心API的快捷键：

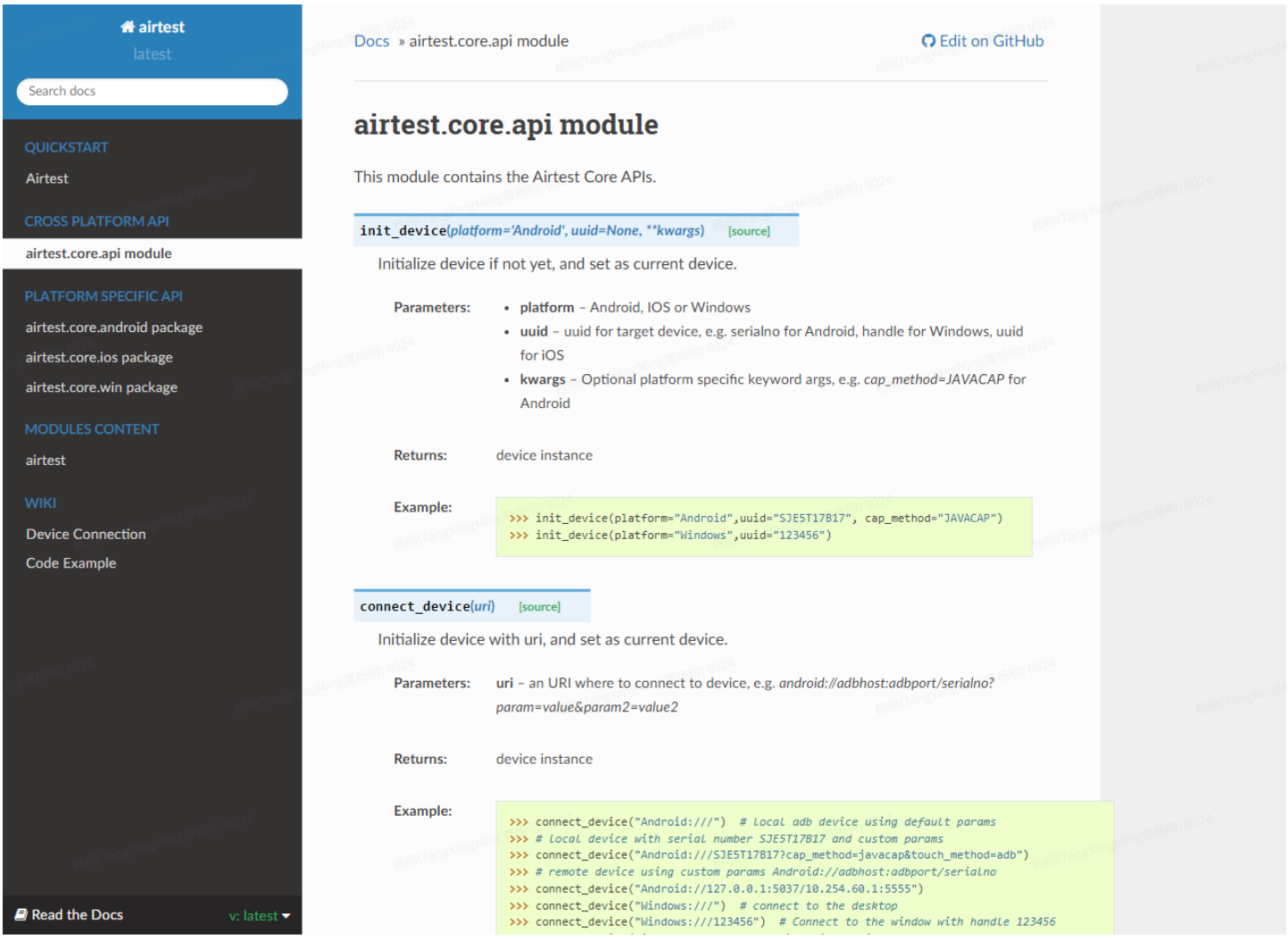
- 点击：`touch`
- 滑动：`swipe`
- 等待截图出现：`wait`
- 存在某个截图：`exists`
- 文本输入：`text`
- 关键词操作：`keyevent`
- 截图：`snapshot`
- 等待：`sleep`
- 断言：`assert_exists`、`assert_not_exists`、`assert_equal`、`assert_not_equal`

在使用Airtest辅助窗给出的API快捷键之前，我们还可以将鼠标移动到按钮上即可看到每个接口的常用参数与返回值信息，非常方便：



④ Airtest API的官方文档

当我们需要实现更多功能、编写更多复杂的Airtest脚本时，我们可以到 Airtest API 的官方文档查询更多我们想要的接口，这里我就不做过多的赘述



```
在设备上启动目标应用
start_app(package, activity=None)
package - 想要启动的应用包名package name，例如 com.netease.my
activity - 需要启动的activity，默认为None，意为main activity
start_app("com.youzu.ssljgj.hjy")

终止目标应用在设备上的运行
stop_app(package)
package - 需要终止运行的应用包名 package name，另见 start_app
stop_app("com.youzu.ssljgj.hjy")

清理设备上的目标应用数据
clear_app(package)
package - 包名 package name，另见 start_app
clear_app("com.youzu.ssljgj.hjy")

安装应用到设备上
install(filepath, **kwargs)
filepath - 需要被安装的应用路径
kwargs - 平台相关的参数 kwargs，请参考对应的平台接口文档
install(r"D:\demo\test.apk")
adb install -r -t D:\demo\test.apk
install(r"D:\demo\test.apk", install_options=["-r", "-t"])
install(r"E:\Downloads\apk\new\三十六计_成都欢聚游_1.0.04929_20220629_170434.apk")

卸载设备上的应用
uninstall(package)
package - 需要被卸载的包名 package name，另见 start_app
uninstall("com.youzu.ssljgj.hjy")

唤醒并解锁目标设备
wake()

返回HOME界面
home()

双击
double_click(v)
v - 点击位置，可以是一个 Template 图片实例，或是一个绝对坐标 (x, y)
double_click((100, 100))

在当前设备画面上进行一次滑动操作
swipe(v1, v2=None, vector=None, **kwargs)
有两种传入参数的方式
swipe(v1, v2=Template(...)) # 从 v1 滑动到 v2
swipe(v1, vector=(x, y)) # 从 v1 开始滑动，沿着vector方向
```

```
v1 – 滑动的起点，可以是一个Template图片实例，或是绝对坐标 (x, y)
v2 – 滑动的终点，可以是一个Template图片实例，或是绝对坐标 (x, y)
vector – 滑动动作的矢量坐标，可以是绝对坐标 (x,y) 或是屏幕百分比，例如 (0.5, 0.5)
**kwargs – 平台相关的参数 kwargs，请参考对应的平台接口文档
swipe((975, 825), (189, 825))

在设备屏幕上执行一个双指pinch捏合操作
pinch(in_or_out='in', center=None, percent=0.5)
in_or_out – 向内捏合或向外扩大，在[“in”, “out”] 中枚举一个值
center – pinch动作的中心位置，默认值为None则为屏幕中心点
percent – pinch动作的屏幕百分比，默认值为0.5

两指向屏幕中心点捏合:
pinch()
将(100, 100)作为中心点，向外扩张两指:
pinch('out', center=(100, 100))

在设备上执行keyevent按键事件
keyevent(keyname, **kwargs)
keyname – 平台相关的按键名称
**kwargs – 平台相关的参数 kwargs，请参考对应的平台接口文档
Android: 相当于执行了 adb shell input keyevent KEYNAME
keyevent("HOME")
The constant corresponding to the home key is 3
keyevent("3")  # same as keyevent("HOME")
keyevent("BACK")
keyevent("KEYCODE_DEL")

iOS: Only supports home/volumeUp/volumeDown:
keyevent("HOME")
keyevent("volumeUp")

在目标设备上输入文本，文本框需要处于激活状态。
text(text, enter=True, **kwargs)
text – 要输入的文本
enter – 是否在输入完毕后，执行一次 Enter ，默认是True
text – 要输入的文本
enter – 是否在输入完毕后，执行一次 Enter ，默认是True
在Android上，有时你需要在输入完毕后点击搜索按钮:
text("test", search=True)

设置一个等待sleep时间，它将会被显示在报告中
sleep(secs=1.0)
secs – sleep的时长
sleep(1)

等待当前画面上出现某个匹配的Template图片
```

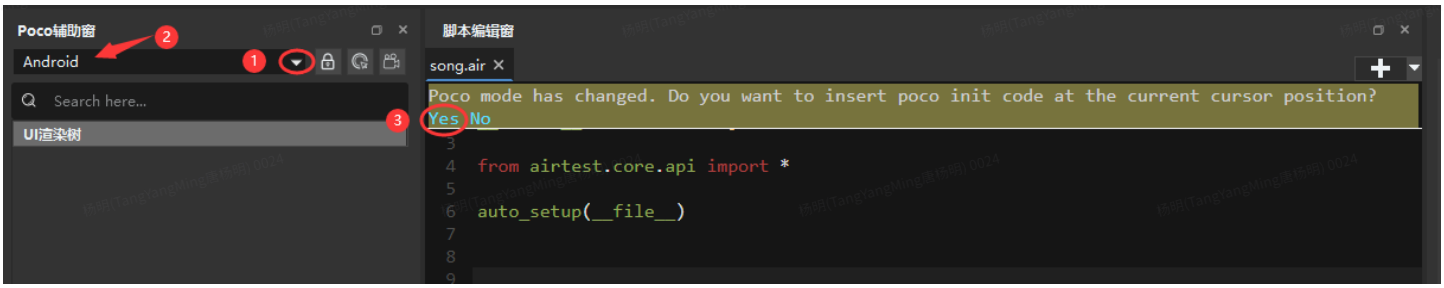
```
wait(v, timeout=None, interval=0.5, intervalfunc=None)
v – 要等待出现的目标Template实例
timeout – 等待匹配的最大超时时长，默认为None即默认取 ST.FIND_TIMEOUT 的值
interval – 尝试查找匹配项的时间间隔（以秒为单位）
intervalfunc – 在首次尝试查找匹配失败后的回调函数

检查设备上是否存在给定目标
exists(v)
v – 要检查的目标
if exists(Template(r"tpl1606822430589.png")):
    touch(Template(r"tpl1606822430589.png"))

pos = exists(Template(r"tpl1606822430589.png"))
if pos:
    touch(pos)

在设备屏幕上查找所有出现的目标并返回其坐标列表
find_all(v)
v – 寻找目标

设备屏幕上存在断言目标
assert_exists(v, msg='')
v – 要检查的目标
msg – 断言的简短描述，它将被记录在报告中
 assert_exists(Template(r"tpl1607324047907.png"), "assert exists")

设备屏幕上不存在断言目标
assert_not_exists(v, msg='')
v – 要检查的目标
msg – 断言的简短描述，它将被记录在报告中
assert_not_exists(Template(r"tpl1607324047907.png"), "assert not exists")

```

2）编写Poco脚本（目前的项目接触不到）

① POCO脚本的初始化

在IDE上编写poco脚本，我们首先要在IDE的Poco辅助窗内选择对应的poco模式，等待刷出待测设备/待测应用的控件树，然后借助这颗树的内容来找到我们的目标控件，并对目标控件进行定位和操作。

其中，在Poco辅助窗内选择对应的poco模式之后，IDE的脚本编辑窗口上方会出现亮黄色的提示，让我们选择是否插入Poco的初始化脚本，我们选择 `Yes` 即可自动插入poco的初始化语句：




```
song.air
# -*- encoding=utf8 -*-
__author__ = "AirtestProject"

from airtest.core.api import *

auto_setup(__file__)


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

这里选择Android模式，表示我们需要展示Android原生的控件树，或者是Android原生应用的控件树。其它模式分别对应了不同项目的poco模式：

② 未成功刷出UI树的解决办法

如果在辅助窗选择对应的模式后，等待几秒仍没有成功刷出UI树，可以检查下述问题：

- 明确待测应用是否为Android/iOS原生应用，非原生应用需接入 pocosdk 才能查看UI树，比如各种引擎渲染的游戏项目就需要接入
- 检查手机上是否已经自动安装上poco初始化相关的2个apk（`pocoservice-debug.apk`、`pocoservice-debug-androidTest.apk`），未安装要手动安装上
- 安装输入法 `Yosemite` 并设为默认输入法
- 部分厂商的手机需要额外的设置，请参考设备连接章节的内容

注意

需要注意的，1.2.12版本的IDE，仅需要安装 1个pocoservice.apk 即可，即 pocoui1.0.84 版本。安装此版本的pocoservice.apk之后，运行环境里面的pocoui也需要更到最新的1.0.84版本！

③ POCO脚本的录制功能

刷新出控件树之后，我们可以利用控件树手动编写脚本，还可以使用辅助窗提供的录制功能，帮助我们一键录制自动化脚本（此处演示的是unity游戏应用的脚本录制，该游戏项目已提前接入了pocosdk）：

④ POCO控件的检索功能

与Airtest的录制功能相似，Poco录制的脚本也不一定是最优脚本，我们还可以利用辅助窗提供的控件检索功能，来精确定位到每一个控件，查看它的详细属性，依此来编写控件的定位和操作脚本：

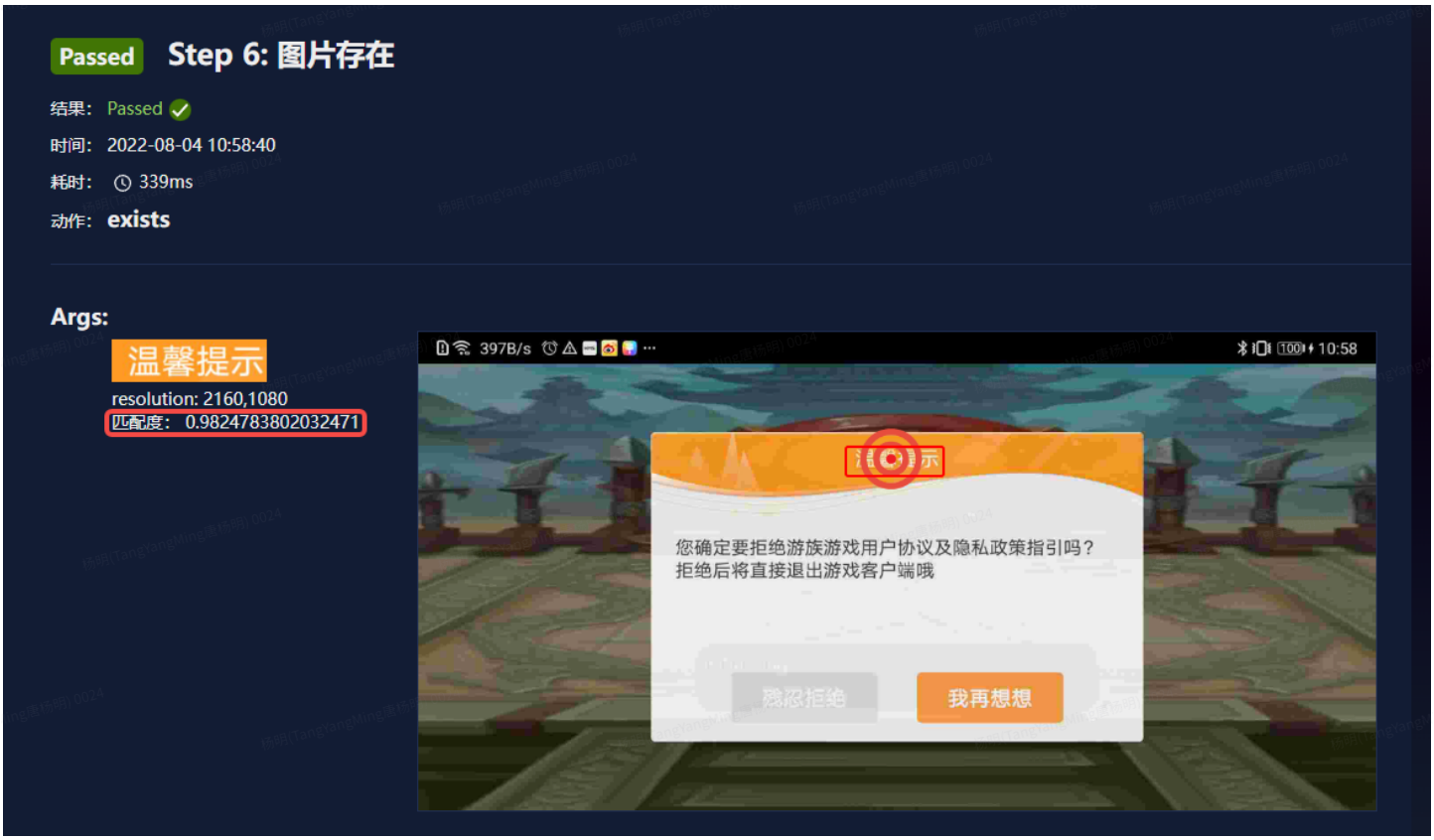
⑤ POCO API的官方文档

当我们想查找更多关于poco的API时，我们可以到 Poco API 的官网文档，查询更多我们想要的接口，以及各种引擎接入 pocosdk 的教程：

4.脚本编写的注意事项

1）截图操作

图片的匹配度需要保持在0.7以上 （可以通过测试报告查看）



2）脚本编写

根据习惯，定制用例函数，写用例根据实际，抽取公共函数

每个操作后，适当sleep，时间自己评估

方法调用 避免重复操作

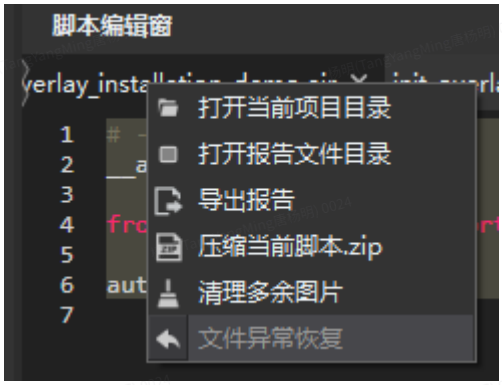
适量断言，避免用例失败，结果通过

5. 在AirtestIDE编写脚本的常见问题

1）文件异常恢复

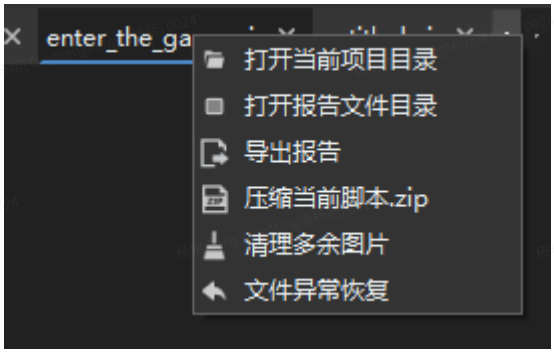
如果突然有一天打开脚本，发现脚本全都变成了红色的点（因未知错误导致的脚本异常）：

我们可以在脚本编辑窗里，右键单击脚本名称，然后选择 `文件异常恢复` ，看能不能尝试将脚本恢复成正常状态。



2）清理多余图片

在IDE编写Airtest的图片脚本时，我们可能反复截取某一个目标，直到能达到我们的脚本要求，但在脚本编辑窗反复截图、删除截图，截图文件是每次都会保存到我们的 `.air` 脚本文件夹里，有些脚本里面不需要的图，我们可以右键单击脚本名称，选择 `清除多余图片` ，可以帮助我们一键清理掉我们脚本里面已经没有用到的截图：



这样就不需要我们在文件夹里一张张去辨别哪些是脚本已经不用的截图了。

3）快速注释与取消注释

在IDE的脚本编辑窗口，选中1行或者多行脚本，按下 `ctrl+/` 键即可快速注释脚本。

同理，选中1行或者多行已经注释的脚本，再次按下 `ctrl+/` 键即可快速取消注释。

4）快速缩进与取消缩进

在IDE的脚本编辑窗口，选中1行或者多行脚本，按下 `Tab` 键即可快速实现缩进，需要多次缩进则按下多次 `Tab` 。

选中1行或者多行脚本，按下 `shift+Tab` 键即可快速实现取消缩进，需要取消多个缩进，则多次按下 `shift+Tab` 键即可。

5）缩进异常/不能对代码进行缩进

有时候在IDE编写脚本时，同学们可能感觉脚本编辑窗的缩进非常乱，不方便查看；或者在使用 `tab` 键进行缩进时发现缩进乱跳的情况，其实是因为脚本编辑窗口过小，脚本一行显示不完，换行显示时不够规范导致的。

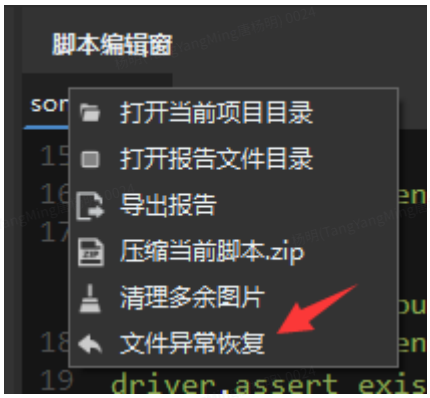
同学们可以将代码复制到 pycharm 或者 note++ 之类的编辑器上面，查看/修改缩进，再粘贴回IDE运行。后面我们也会尝试改建编辑窗的缩进显示。

6）脚本自动保存与缓存文件找回

在脚本编写工作中，我们十分推荐各位尽可能地利用一些版本管理软件，对编写的脚本进行管理，这样能避免很多脚本的丢失与多人协作的问题。

在使用AirtestIDE编写脚本时，我们将会每隔30秒对当前编辑窗口的脚本进行一次自动保存工作，避免遗忘保存导致的脚本遗失问题。

然而在一些极端情况下（电脑蓝屏、死机等），还有可能出现脚本编辑框显示错乱，导致原先的脚本丢失的问题。因此我们新增了一个脚本缓存找回功能，每当 手动选择保存 或是 运行脚本 时，都认为当前文件是一个完好的脚本文件，因此会自动在系统缓存目录下另存一份脚本的纯文本内容。如果遇到了异常情况，需要找回以前保存过的脚本的话，可以在脚本标题上点击鼠标右键：



随后的弹窗中，有3个可选项：

最左边的按钮是一键自动恢复，能将最近一次的缓存文件内容恢复到当前编辑窗口中。中间的按钮是打开缓存文件夹，打开后将会看到一些纯文本文件，命名规则是将脚本文件的路径中的 `:\/` 等符号替换成 `%` ，最后以时间戳进行结尾，可以自行找到编辑中的文件名对应的缓存文件后，用文本编辑器打开，就可以看到自动保存的内容了。

7）MacOS的脚本丢失

第一次打开AirtestIDE时，会默认帮忙在系统temp目录下新建一个空脚本，方便大家熟悉软件功能和上手编写脚本。请在脚本编写的工作中，尽量避免将脚本保存在系统目录、或无权限读写的目录。如果是MacOS，请 不要将脚本保存在AirtestIDE.app的目录里 （例如：`/Applications/AirtestIDE.app/Contents/MacOS` ），因为一旦重装应用，将会覆盖掉该目录，导致脚本丢失。

请务必选择一个合适的、有存取权限的目录来存放你的脚本。