airtest Documentation

Game-Netease

2022 年 05 月 18 日

快速开始

CHAPTER 1

Airtest

跨平台的 UI 自动化测试框架，适用于游戏和 App

Airtest 是一个跨平台的 UI 自动化测试框架，适用于游戏和 App。

如果你是 Airtest 新手用户，从 官网 开始上手吧。

以下文档会介绍 Airtest 的基本用法，以及提供 API 文档。

1.1 Airtest

跨平台的 UI 自动化框架，适用于游戏和 App

1.1.1 快速开始

- **跨平台：** Airtest 可以在几乎所有平台上执行游戏和 App 的自动化。
- **一次编写，随处运行：** Airtest 提供了跨平台的 API，囊括了应用安装、模拟输入以及断言等。由于使用图像识别技术来定位 UI 元素，因此无需嵌入任何代码即可对游戏和应用进行自动化操作。
- **可扩展性：**通过使用 Airtest 提供的命令行与 python API 接口，可以很轻松地在大规模设备集群上运行脚本。提供的 HTML 报告包含了详细操作步骤和截屏，能够迅速定位到失败点。同时，网易也提供了 Airlab 云测试平台服务。
- **AirtestIDE：** AirtestIDE 是一个强大的 GUI 工具，可以帮助你录制和调试测试脚本。AirtestIDE 提供了完整的自动化工作流程支持：录制脚本-> 真机回放-> 生成报告。

从官网开始上手吧

1.1.1.1 支持平台

- Android
- iOS
- Windows
- Unity
- Cocos2dx
- 白鹭引擎
- 微信小程序

Read more

1.1.2 安装

这部分说明了如何在本地 python 环境中安装 Airtest 测试框架。如果你需要使用 GUI 工具，请从 官网 直接下载 AirtestIDE 使用。

1.1.2.1 系统要求

- 操作系统
  - Windows
  - MacOS X
  - Linux
- Python2.7 & Python3.3+

1.1.2.2 安装 Python Package

使用 `pip` 来管理安装包和自动安装所有依赖。

```
pip install -U airtest
```

你也可以直接从 Git 仓库安装。

```
git clone https://github.com/AirtestProject/Airtest.git
pip install -e airtest
```

因为 Airtest 还在快速开发中，这里使用 `-e` 来安装源码。以后你就可以直接使用 `git pull` 更新代码目录来升级 Airtest 了。

1.1.3 文档

完整的 Airtest 框架文档请查阅 readthedocs。

1.1.4 例子

Airtest 提供了简洁而且平台无关的 API。这部分介绍了如何使用这些 API 来编写一个自动化脚本，步骤如下：

1. 通过 ADB 连接一台安卓手机
2. 安装应用 APK
3. 运行应用并截图
4. 模拟用户输入（点击、滑动、按键）
5. 卸载应用

```
from airtest.core.api import *

# connect an android phone with adb
init_device("Android")
# or use connect_device api
# connect_device("Android:///")

install("path/to/your/apk")
start_app("package_name_of_your_apk")
touch(Template("image_of_a_button.png"))
swipe(Template("slide_start.png"), Template("slide_end.png"))
assert_exists(Template("success.png"))
keyevent("BACK")
home()
uninstall("package_name_of_your_apk")
```

更多 API 和使用方法，请参考完整的 Airtest Python API reference ，或者直接看看 API code

1.1.5 基本使用方法

Airtest 希望提供平台无关的 API，让你的测试代码可以运行在不同平台的设备和应用上。

1. 使用 connect_device 来连接任意 Android/iOS 设备或者 Windows 窗口。

2. 使用 模拟操作 的 API 来自动化你的游戏或者 App。
3. 千万 **不要**忘记声明断言 来验证测试结果。

1.1.5.1 连接设备

使用 connect_device 来连接任意 Android/iOS 设备或者 Windows 窗口。

```
connect_device("platform://host:port/uuid?param=value&param2=value2")
```

- platform: Android/iOS/Windows…
- host: Android 平台是 adb host，iOS 下是 iproxy host，其他平台请留空
- port: Android 下是 adb port，iOS 下填写 iproxy port，其他平台请留空
- uuid: 目标设备的 uuid，例如 Android 下是序列号，windows 下是窗口句柄，iOS 是 uuid
- param: 设备初始化的配置字段，例如 cap_method/ori_method/…
- value: 设备初始化字段的值。

查看 connect_devce 获取更多信息。

连接安卓设备

1. 通过 usb 将手机与电脑相连
2. 命令行输入 `adb devices` 命令，确保手机连接状态是 `device`
3. 在 Airtest 中连接设备
4. 如果你连接了多个设备，或者有远程设备，那么使用参数来指定要连接的设备

```
# connect an android phone with adb
init_device("Android")

# or use connect_device api with default params
connect_device("android:///")

# connect a remote device using custom params
connect_device("android://adbhost:adbport/1234566?cap_method=javacap&touch_method=adb")
```

连接 iOS 设备

根据 iOS-Tagent 的操作指引来配置环境。

```
# connect a local ios device
connect_device("ios:///")
```

连接 windows 应用

```
# connect local windows desktop
connect_device("Windows:///")

# connect local windows application
connect_device("Windows:///?title_re=unity.*")
```

Airtest 使用了 `pywinauto` 作为操作 Windows 应用的底层库，更多窗口搜索的参数请看 pywinauto documentation 。

1.1.5.2 模拟输入

支持以下常用 API：

- touch
- swipe
- text
- keyevent
- snapshot
- wait

支持更多 API，其中部分是平台相关的 API，请查看 API reference 。

在使用这些通用 API 时，Airtest 会自动根据当前平台调用对应的操作，例如以下代码示例：

```
from airtest.core.api import *  # import the general APIs such as touch/swipe/...

connect_device("Android:///")
touch((100, 100))

connect_device("Windows:///")
touch((100, 100))
```

Airtest 还针对各个平台，提供了一些平台专用 API，可以分别在每个平台的模块 API 中查询到：

- 通用 APIs: API reference
- Android: airtest.core.android.Android class

- Windows: airtest.core.win.Windows class
- iOS: airtest.core.ios.IOS class

```
>>> from airtest.core.api import *
>>> connect_device("Android:///")
>>> dev = device()  # get current device, which is an Android object
>>> print(type(dev))
<class 'airtest.core.android.android.Android'>
>>> dev.get_top_activity()  # use Android ADB to get the top activity
('com.google.android.apps.nexuslauncher', '.NexusLauncherActivity', '2720')
```

1.1.5.3 声明断言

Airtest 提供了以下断言方法：

- assert_exists
- assert_not_exists
- assert_equal
- assert_not_equal

当断言失败，会抛出 `AssertsionError`。所有断言都会在 html 报告中显示。

1.1.6 用命令行运行 `.air` 脚本

使用 AirtestIDE 你可以非常轻松地录制一个自动化脚本并保存为 `.air` 目录结构。Airtest 命令行则让你能够脱离 IDE，在不同宿主机器和被测设备上运行测试脚本。

你可以在命令行参数中指定连接的被测设备，这样就可以运行在不同的手机平台和宿主机器上。只要你的测试代码本身是平台无关的，你就可以在一个平台上录制脚本，然后在不同平台上运行。

下面的例子介绍了命令行的基本用法。可以配合我们提供的示例 `airtest/playground/test_blackjack.air/` 来学习使用：

1.1.6.1 运行自动化用例

```
# run automated cases and scenarios on various devices
> airtest run "path to your .air dir" --device Android:///
> airtest run "path to your .air dir" --device Android://adbhost:adbport/serialno
> airtest run "path to your .air dir" --device Windows:///?title_re=Unity.*
> airtest run "path to your .air dir" --device iOS:///
...
```

(下页继续)

(续上页)

```
# show help
> airtest run -h
usage: airtest run [-h] [--device [DEVICE]] [--log [LOG]]
                   [--recording [RECORDING]]
                   script

positional arguments:
  script                air path

optional arguments:
  -h, --help            show this help message and exit
  --device [DEVICE]     connect dev by uri string, e.g. Android:///
  --log [LOG]           set log dir, default to be script dir
  --recording [RECORDING]
                        record screen when running
  --compress
                        set snapshot quality, 1-99
  --no-image [NO_IMAGE]
                        Do not save screenshots
```

1.1.6.2 生成报告

```
> airtest report "path to your .air dir"
log.html
> airtest report -h
usage: airtest report [-h] [--outfile OUTFILE] [--static_root STATIC_ROOT]
                      [--log_root LOG_ROOT] [--record RECORD [RECORD ...]]
                      [--export EXPORT] [--lang LANG]
                      script

positional arguments:
  script                script filepath

optional arguments:
  -h, --help            show this help message and exit
  --outfile OUTFILE     output html filepath, default to be log.html
  --static_root STATIC_ROOT
                        static files root dir
  --log_root LOG_ROOT   log & screen data root dir, logfile should be
```

(下页继续)

(续上页)

```
                        log_root/log.txt
  --record RECORD [RECORD ...]
                        custom screen record file path
  --export EXPORT       export a portable report dir containing all resources
  --lang LANG           report language
```

1.1.6.3 获取脚本信息

```
# print case info in json if defined, including: author, title, desc
> python -m airtest info "path to your .air dir"
{"author": ..., "title": ..., "desc": ...}
```

1.1.7 引用其他的 .air 脚本

可以将一些通用的操作写在一个 .air 脚本里，然后在其他脚本中 import 它。Airtest 提供了 using 接口，能够将需要引用的脚本加入 sys.path 里，其中包含的图片文件也会被加入 Template 的搜索路径中。

```
from airtest.core.api import using
using("common.air")

from common import common_function

common_function()
```

1.2 airtest.core.api module

这个模块包含了 Airtest 核心 API。

init_device(*platform='Android'*, *uuid=None*, ***kwargs*)

初始化设备，并设置为当前设备。

参数

- **platform** – Android, IOS or Windows
- **uuid** – 目标设备的 uuid，例如 Android 的序列号，Windows 的窗口句柄，或 iOS 的 uuid
- **kwargs** – 可选的平台相关的参数，例如 Android 下的“cap_method=JAVACAP“参数

返回 device 对象

示例

```
>>> init_device(platform="Android",uuid="SJE5T17B17", cap_method=
↪"JAVACAP")
>>> init_device(platform="Windows",uuid="123456")
```

connect_device(*uri*)

用 URI 字符串来初始化设备，并且设置为当前设备。

参数 **uri** – 一个用于初始化设备的 URI 字符串，例如 android://adbhost:adbport/serialno?param=value&param2=value2

返回 device 对象

示例

```
>>> connect_device("Android:///")  # local adb device using default␣
↪params
>>> # local device with serial number SJE5T17B17 and custom params
>>> connect_device("Android:///SJE5T17B17?cap_method=javacap&touch_
↪method=adb")
>>> # remote device using custom params Android://adbhost:adbport/
↪serialno
>>> connect_device("Android://127.0.0.1:5037/10.254.60.1:5555")
>>> connect_device("Windows:///")  # connect to the desktop
>>> connect_device("Windows:///123456")  # Connect to the window with␣
↪handle 123456
>>> connect_device("iOS:///127.0.0.1:8100")  # iOS device
>>> connect_device("iOS:///http://localhost:8100/?mjpeg_port=9100")  #␣
↪iOS with mjpeg port
```

device()

返回当前正在使用中的设备。

返回 当前设备实例

示例

```
>>> dev = device()
>>> dev.touch((100, 100))
```

set_current(*idx*)

设置当前设备。

参数 **idx** – uuid 或已初始化的设备列表中的编号，从 0 开始

引发 `IndexError` – 当查找不到设备时

返回 None

支持平台 Android, iOS, Windows

示例

```
>>> # switch to the first phone currently connected
>>> set_current(0)
>>> # switch to the phone with serial number serialno1
>>> set_current("serialno1")
```

`auto_setup`(*basedir=None*, *devices=None*, *logdir=None*, *project_root=None*, *compress=None*)

自动配置运行环境，如果当前没有连接设备的话，就默认尝试连接 Android 设备。

参数

- `basedir` – 设置当前脚本的所在路径，也可以直接传 `__file__` 变量进来
- `devices` – 一个内容为 `connect_device uri` 字符串的列表
- `logdir` – 可设置脚本运行时的 log 保存路径，默认值为 None 则不保存 log，如果设置为 True 则自动保存在 <basedir>/log 目录中。
- `project_root` – 用于设置 PROJECT_ROOT 变量，方便 `using` 接口的调用
- `compress` – 屏幕截图的压缩比率，在 [1, 99] 范围内的整数，默认是 10

示例

```
>>> auto_setup(__file__)
>>> auto_setup(__file__, devices=["Android://127.0.0.1:5037/SJE5T17B17
↪"],
...             logdir=True, project_root=r"D:\test\logs", compress=90)
```

`shell`(*cmd*)

在目标设备上运行远程 shell 指令

参数 `cmd` – 需要在设备上运行的指令，例如 `ls /data/local/tmp`

返回 shell 指令的输出内容

支持平台 Android

示例

```
>>> # Execute commands on the current device adb shell ls
>>> print(shell("ls"))
```

```
>>> # Execute adb instructions for specific devices
>>> dev = connect_device("Android:///device1")
>>> dev.shell("ls")
```

```
>>> # Switch to a device and execute the adb command
>>> set_current(0)
>>> shell("ls")
```

start_app(*package*, *activity=None*)

在设备上启动目标应用

参数

- **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

返回 None

支持平台 Android, iOS

示例

```
>>> start_app("com.netease.cloudmusic")
>>> start_app("com.apple.mobilesafari")  # on iOS
```

stop_app(*package*)

终止目标应用在设备上的运行

参数 **package** – 需要终止运行的应用包名 package name，另见 start_app

返回 None

支持平台 Android, iOS

示例

```
>>> stop_app("com.netease.cloudmusic")
```

clear_app(*package*)

清理设备上的目标应用数据

参数 **package** – 包名 package name，另见 start_app

返回 None

支持平台 Android

示例

```
>>> clear_app("com.netease.cloudmusic")
```

install(*filepath*, ***kwargs*)

安装应用到设备上

参数

- **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

返回 None

支持平台 Android

示例

```
>>> install(r"D:\demo\test.apk")
>>> # adb install -r -t D:\demo\test.apk
>>> install(r"D:\demo\test.apk", install_options=["-r", "-t"])
```

uninstall(*package*)

卸载设备上的应用

参数 **package** – 需要被卸载的包名 package name，另见 start_app

返回 None

支持平台 Android

示例

```
>>> uninstall("com.netease.cloudmusic")
```

snapshot(*filename=None*, *msg=''*, *quality=None*, *max_size=None*)

对目标设备进行一次截图，并且保存到文件中。

参数

- **filename** – 保存截图的文件名，默认保存路径为“ST.LOG_DIR“中
- **msg** – 截图文件的简短描述，将会被显示在报告页面中
- **quality** – 图片的质量，[1,99] 的整数，默认是 10
- **max_size** – 图片的最大尺寸，例如 1200

返回 { “screen” : filename, “resolution” : resolution of the screen} or None

支持平台 Android, iOS, Windows

示例

```
>>> snapshot(msg="index")
>>> # save the screenshot to test.jpg
>>> snapshot(filename="test.png", msg="test")
```

可以设置截图的画质和大小

```
>>> # Set the screenshot quality to 30
>>> ST.SNAPSHOT_QUALITY = 30
>>> # Set the screenshot size not to exceed 600*600
>>> # if not set, the default size is the original image size
>>> ST.IMAGE_MAXSIZE = 600
>>> # The quality of the screenshot is 30, and the size does not exceed␣
↪600*600
>>> touch((100, 100))
>>> # The quality of the screenshot of this sentence is 90
>>> snapshot(filename="test.png", msg="test", quality=90)
>>> # The quality of the screenshot is 90, and the size does not exceed␣
↪1200*1200
>>> snapshot(filename="test2.png", msg="test", quality=90, max_
↪size=1200)
```

wake()

唤醒并解锁目标设备

返回 None

支持平台 Android

示例

```
>>> wake()
```

注解: 在部分品牌手机上可能无法生效

---

home()

返回 HOME 界面。

返回 None

支持平台 Android, iOS

示例

```
>>> home()
```

touch(*v*, *times=1*, ***kwargs*)

在当前设备画面上进行一次点击

参数

- **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)
- **times** – 点击次数
- **kwargs** – 平台相关的参数 kwargs，请参考对应的平台接口文档

返回 finial position to be clicked, e.g. (100, 100)

支持平台 Android, Windows, iOS

示例 点击绝对坐标:

```
>>> touch((100, 100))
```

点击图片的中心位置:

```
>>> touch(Template(r"tpl1606730579419.png", target_pos=5))
```

点击两次:

```
>>> touch((100, 100), times=2)
```

在 Android 和 Windows 下，可以设置点击持续时间:

```
>>> touch((100, 100), duration=2)
```

右键点击（Windows):

```
>>> touch((100, 100), right_click=True)
```

click(*v*, *times=1*, ***kwargs*)

在当前设备画面上进行一次点击

参数

- **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)
- **times** – 点击次数
- **kwargs** – 平台相关的参数 kwargs，请参考对应的平台接口文档

返回 finial position to be clicked, e.g. (100, 100)

支持平台 Android, Windows, iOS

示例 点击绝对坐标:

```
>>> touch((100, 100))
```

点击图片的中心位置:

```
>>> touch(Template(r"tpl1606730579419.png", target_pos=5))
```

点击两次:

```
>>> touch((100, 100), times=2)
```

在 Android 和 Windows 下，可以设置点击持续时间:

```
>>> touch((100, 100), duration=2)
```

右键点击（Windows):

```
>>> touch((100, 100), right_click=True)
```

double_click(*v*)

双击

参数 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)

返回 实际点击位置坐标 (x, y)

示例

```
>>> double_click((100, 100))
>>> double_click(Template(r"tpl1606730579419.png"))
```

swipe(*v1*, *v2=None*, *vector=None*, ***kwargs*)

在当前设备画面上进行一次滑动操作。

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。

参数

- **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

引发 Exception – 当没有足够的参数来执行滑动时引发异常

返回 原点位置和目标位置

支持平台 Android, Windows, iOS

示例

```
>>> swipe(Template(r"tpl1606814865574.png"), vector=[-0.0316, -0.3311])
>>> swipe((100, 100), (200, 200))
```

自定义滑动持续时间和经过几步到达终点:

```
>>> # swiping lasts for 1 second, divided into 6 steps
>>> swipe((100, 100), (200, 200), duration=1, steps=6)
```

pinch(*in_or_out='in'*, *center=None*, *percent=0.5*)

在设备屏幕上执行一个双指 pinch 捏合操作

参数

- **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

返回 None

支持平台 Android

示例 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))
```

keyevent(*keyname*, ***kwargs*)

在设备上执行 keyevent 按键事件

参数

- **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

返回 None

支持平台 Android, Windows, iOS

示例

- 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")
```

参见:

Module *airtest.core.android.adb.ADB.keyevent* 相当于调用 android.adb.keyevent()

Android Keyevent Android.KeyEvent 的参考文档

- Windows: 使用 pywinauto.keyboard 进行按键点击:

```
>>> keyevent("{DEL}")
>>> keyevent("%{F4}")  # close an active window with Alt+F4
```

参见:

Module *airtest.core.win.win.Windows.keyevent*

pywinauto.keyboard Documentation for pywinauto.keyboard

- iOS: Only supports home/volumeUp/volumeDown:

```
>>> keyevent("HOME")
>>> keyevent("volumeUp")
```

text(*text*, *enter=True*, ***kwargs*)

在目标设备上输入文本，文本框需要处于激活状态。

参数

- **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

返回 None

支持平台 Android, Windows, iOS

示例

```
>>> text("test")
>>> text("test", enter=False)
```

在 Android 上，有时你需要在输入完毕后点击搜索按钮:

```
>>> text("test", search=True)
```

参见:

Module *airtest.core.android.ime.YosemiteIme.code*

如果希望输入其他按键，可以用这个接口:

```
>>> text("test")
>>> device().yosemite_ime.code("3")  # 3 = IME_ACTION_SEARCH
```

Ref: Editor Action Code

`sleep`(*secs=1.0*)

设置一个等待 sleep 时间，它将会被显示在报告中

参数 **secs** – sleep 的时长

返回 None

支持平台 Android, Windows, iOS

示例

```
>>> sleep(1)
```

`wait`(*v*, *timeout=None*, *interval=0.5*, *intervalfunc=None*)

等待当前画面上出现某个匹配的 Template 图片

参数

- **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

引发 *TargetNotFoundError* – 在超时后仍未找到目标则触发

返回 匹配目标的坐标

支持平台 Android, Windows, iOS

示例

```
>>> wait(Template(r"tpl1606821804906.png"))  # timeout after ST.FIND_
↪TIMEOUT
>>> # find Template every 3 seconds, timeout after 120 seconds
>>> wait(Template(r"tpl1606821804906.png"), timeout=120, interval=3)
```

你可以在每次查找目标失败时，指定一个回调函数:

```
>>> def notfound():
>>>     print("No target found")
>>> wait(Template(r"tpl1607510661400.png"), intervalfunc=notfound)
```

`exists`(*v*)

检查设备上是否存在给定目标

参数 v – 要检查的目标

返回 如果未找到目标，则返回 False，否则返回目标的坐标

支持平台 Android, Windows, iOS

示例

```
>>> if exists(Template(r"tpl1606822430589.png")):
>>>     touch(Template(r"tpl1606822430589.png"))
```

因为 `exists()` 会返回坐标，我们可以直接点击坐标来减少一次图像查找

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

`find_all`(*v*)

在设备屏幕上查找所有出现的目标并返回其坐标列表

参数 v – 寻找目标

返回 结果列表，[{ 'result' : (x, y), 'rectangle' : ((left_top, left_bottom, right_bottom, right_top)), 'confidence' : 0.9}, ···]

支持平台 Android, Windows, iOS

示例

```
>>> find_all(Template(r"tpl1607511235111.png"))
[{'result': (218, 468), 'rectangle': ((149, 440), (149, 496), (288,␣
↪496), (288, 440)),
'confidence': 0.9999996423721313}]
```

assert_exists(*v*, *msg=''*)

设备屏幕上存在断言目标

参数

- **v** – 要检查的目标
- **msg** – 断言的简短描述，它将被记录在报告中

引发 **AssertionError** – 如果断言失败

返回 目标坐标

支持平台 Android, Windows, iOS

示例

```
>>> assert_exists(Template(r"tpl1607324047907.png"), "assert exists")
```

assert_not_exists(*v*, *msg=''*)

设备屏幕上不存在断言目标

参数

- **v** – 要检查的目标
- **msg** – 断言的简短描述，它将被记录在报告中

引发 **AssertionError** – 如果断言失败

返回 None.

支持平台 Android, Windows, iOS

示例

```
>>> assert_not_exists(Template(r"tpl1607324047907.png"), "assert not␣
↪exists")
```

assert_equal(*first*, *second*, *msg=''*)

断言两个值相等

参数

- **first** – 第一个值
- **second** – 第二个值
- **msg** – 断言的简短描述，它将被记录在报告中

引发 **AssertionError** – 如果断言失败

返回 None

支持平台 Android, Windows, iOS

示例

```
>>> assert_equal(1, 1, msg="assert 1==1")
```

assert_not_equal(*first*, *second*, *msg=''*)

断言两个值不相等

参数

- **first** – 第一个值
- **second** – 第二个值
- **msg** – 断言的简短描述，它将被记录在报告中

引发 **AssertionError** – 如果断言异常

返回 None

支持平台 Android, Windows, iOS

示例

```
>>> assert_not_equal(1, 2, msg="assert 1!=2")
```

1.3 airtest.core.android package

This package provide Android Device Class.

为 Android 平台提供的接口，请参考：airtest.core.android.Android class

1.3.1 Subpackages

1.3.1.1 airtest.core.android.touch_methods package

Submodules

airtest.core.android.touch_methods.base_touch module

retry_when_connection_error(*func*)

class BaseTouch(*adb*, *backend=False*, *size_info=None*, *input_event=None*, **args*, ***kwargs*)

基类：object

A super class for Minitouch or Maxtouch

install_and_setup()

Install and setup airtest touch

返回 None

`uninstall()`

Uninstall airtest touch

返回 None

`install()`

Install airtest touch

返回 None

`setup_server()`

Setip touch server and adb forward

返回 server process

`safe_send(data)`

Send data to client

参数 `data` – data to send

Raises `Exception` – when data cannot be sent

返回 None

`setup_client_backend()`

Setup backend client thread as daemon

返回 None

`setup_client()`

Setup client

返回 None

`teardown()`

Stop the server and client

返回 None

`transform_xy(x, y)`

Transform coordinates (x, y) according to the device display

参数

- `x` – coordinate x
- `y` – coordinate y

返回 transformed coordinates (x, y)

`perform(motion_events, interval=0.01)`

Perform a sequence of motion events including: UpEvent, DownEvent, MoveEvent, SleepEvent

参数

- **motion_events** – a list of MotionEvent instances
- **interval** – minimum interval between events

返回 None

touch(*tuple_xy*, *duration=0.01*)

Perform touch event

minitouch protocol example:

```
d 0 10 10 50
c
<wait in your own code>
u 0
c
```

参数

- **tuple_xy** – coordinates (x, y)
- **duration** – time interval for touch event, default is 0.01

返回 None

swipe_along(*coordinates_list*, *duration=0.8*, *steps=5*)

Perform swipe event across multiple points in sequence.

参数

- **coordinates_list** – list of coordinates: [(x1, y1), (x2, y2), (x3, y3)]
- **duration** – time interval for swipe duration, default is 0.8
- **steps** – size of swipe step, default is 5

返回 None

swipe(*tuple_from_xy*, *tuple_to_xy*, *duration=0.8*, *steps=5*)

Perform swipe event.

参数

- **tuple_from_xy** – start point
- **tuple_to_xy** – end point
- **duration** – time interval for swipe duration, default is 0.8
- **steps** – size of swipe step, default is 5

返回 None

two_finger_swipe(*tuple_from_xy*, *tuple_to_xy*, *duration=0.8*, *steps=5*, *offset=(0, 50)*)

Perform two finger swipe action

minitouch protocol example:

```
d 0 0 0 50
d 1 1 0 50
c
m 0 20 0 50
m 1 21 0 50
c
m 0 40 0 50
m 1 41 0 50
c
m 0 60 0 50
m 1 61 0 50
c
m 0 80 0 50
m 1 81 0 50
c
m 0 100 0 50
m 1 101 0 50
c
u 0
u 1
c
```

参数

- **tuple_from_xy** – start point
- **tuple_to_xy** – end point
- **duration** – time interval for swipe duration, default is 0.8
- **steps** – size of swipe step, default is 5
- **offset** – coordinate offset of the second finger, default is (0, 50)

返回 None

pinch(*center=None*, *percent=0.5*, *duration=0.5*, *steps=5*, *in_or_out='in'*)

Perform pinch action

minitouch protocol example:

```
d 0 0 100 50
d 1 100 0 50
c
m 0 10 90 50
m 1 90 10 50
c
m 0 20 80 50
m 1 80 20 50
c
m 0 20 80 50
m 1 80 20 50
c
m 0 30 70 50
m 1 70 30 50
c
m 0 40 60 50
m 1 60 40 50
c
m 0 50 50 50
m 1 50 50 50
c
u 0
u 1
c
```

参数

- **center** – the center point of the pinch operation
- **percent** – pinch distance to half of screen, default is 0.5
- **duration** – time interval for swipe duration, default is 0.8
- **steps** – size of swipe step, default is 5
- **in_or_out** – pinch in or pinch out, default is 'in'

返回 None

Raises TypeError – An error occurred when center is not a list/tuple or None

operate(*args*)

Perform down, up and move actions

参数 **args** – action arguments, dictionary containing type and x, y coordinates, e.g.:

```
{
"type" : "down",
"x" : 10,
"y" : 10
}
```

Raises `RuntimeError` – is invalid arguments are provided

返回 None

`class MotionEvent`

基类: `object`

Motion Event to be performed by Minitouch/Maxtouch

`getcmd(transform=None)`

`class DownEvent(coordinates, contact=0, pressure=50)`

基类: *airtest.core.android.touch_methods.base_touch.MotionEvent*

`getcmd(transform=None)`

`class UpEvent(contact=0)`

基类: *airtest.core.android.touch_methods.base_touch.MotionEvent*

`getcmd(transform=None)`

`class MoveEvent(coordinates, contact=0, pressure=50)`

基类: *airtest.core.android.touch_methods.base_touch.MotionEvent*

`getcmd(transform=None)`

`class SleepEvent(seconds)`

基类: *airtest.core.android.touch_methods.base_touch.MotionEvent*

`getcmd(transform=None)`

airtest.core.android.touch_methods.maxtouch module

`class Maxtouch(adb, backend=False, size_info=None, input_event=None)`

基类: *airtest.core.android.touch_methods.base_touch.BaseTouch*

`install()`

Install maxtouch

返回 None

`uninstall()`

Uninstall maxtouch

返回 None

setup_server()

Setup maxtouch server and adb forward

返回 server process

setup_client()

Setup client

返回 None

transform_xy(*x*, *y*)

Normalized coordinates (x, y)

参数

- **x** – coordinate x
- **y** – coordinate y

返回 transformed coordinates (x, y)

airtest.core.android.touch_methods.minitouch module

class Minitouch(*adb*, *backend=False*, *size_info=None*, *input_event=None*)

基类: *airtest.core.android.touch_methods.base_touch.BaseTouch*

install()

Install minitouch

返回 None

uninstall()

Uninstall minitouch

返回 None

setup_server()

Setup minitouch server and adb forward

返回 server process

setup_client()

Setup client in following steps:

```
1. connect to server
2. receive the header
    v <version>
    ^ <max-contacts> <max-x> <max-y> <max-pressure>
    $ <pid>
3. prepare to send
```

返回 None

transform_xy(*x*, *y*)

Transform coordinates (x, y) according to the device display

参数

- **x** – coordinate x
- **y** – coordinate y

返回 transformed coordinates (x, y)

airtest.core.android.touch_methods.touch_proxy module

class TouchProxy(*touch_method*)

基类：object

Perform touch operation according to the specified method

TOUCH_METHODS = {'MAXTOUCH': <class 'airtest.core.android.touch_methods.touch_proxy.MaxtouchImpleme

classmethod check_touch(*touch_impl*)

classmethod auto_setup(*adb*, *default_method=None*, *ori_transformer=None*, *size_info=None*, *input_event=None*)

参数

- **adb** – *airtest.core.android.adb.ADB*
- **default_method** – The default click method, such as "MINITOUCH"
- **ori_transformer** – dev._touch_point_by_orientation
- **size_info** – the result of dev.get_display_info()
- **input_event** – dev.input_event
- ***args** –
- ****kwargs** –

Returns: TouchProxy object

实际案例

```
>>> dev = Android()
>>> touch_proxy = TouchProxy.auto_setup(dev.adb, ori_transformer=dev._touch_
↪point_by_orientation)
>>> touch_proxy.touch((100, 100))
```

register_touch(*cls*)

class AdbTouchImplementation(*base_touch*)

基类：object

METHOD_NAME = 'ADBTOUCH'

touch(*pos*, *duration=0.01*)

swipe(*p1*, *p2*, *duration=0.5*, **args*, ***kwargs*)

class MinitouchImplementation(*minitouch*, *ori_transformer*)

基类：*airtest.core.android.touch_methods.touch_proxy.AdbTouchImplementation*

METHOD_NAME = 'MINITOUCH'

METHOD_CLASS

airtest.core.android.touch_methods.minitouch.Minitouch 的别名

touch(*pos*, *duration=0.01*)

swipe(*p1*, *p2*, *duration=0.5*, *steps=5*, *fingers=1*)

pinch(*center=None*, *percent=0.5*, *duration=0.5*, *steps=5*, *in_or_out='in'*)

swipe_along(*coordinates_list*, *duration=0.8*, *steps=5*)

two_finger_swipe(*tuple_from_xy*, *tuple_to_xy*, *duration=0.8*, *steps=5*, *offset=(0, 50)*)

perform(*motion_events*, *interval=0.01*)

class MaxtouchImplementation(*maxtouch*, *ori_transformer*)

基类：*airtest.core.android.touch_methods.touch_proxy.MinitouchImplementation*

METHOD_NAME = 'MAXTOUCH'

METHOD_CLASS

airtest.core.android.touch_methods.maxtouch.Maxtouch 的别名

perform(*motion_events*, *interval=0.01*)

1.3.1.2 airtest.core.android.cap_methods package

Submodules

airtest.core.android.cap_methods.adbcap module

class AdbCap(*adb*, **args*, ***kwargs*)

基类：*airtest.core.android.cap_methods.base_cap.BaseCap*

get_frame_from_stream()

Get a frame of the current screen from the mobile screen stream

从手机画面流中，获取一张当前屏幕截图

Returns: frame_data

snapshot(*ensure_orientation=True*)

Take a screenshot and convert it into a cv2 image object

获取一张屏幕截图，并转化成 cv2 的图像对象

Returns: numpy.ndarray

airtest.core.android.cap_methods.base_cap module

class BaseCap(*adb*, **args*, ***kwargs*)

基类：object

Base class for all screenshot methods 所有屏幕截图方法的基类

get_frame_from_stream()

Get a frame of the current screen from the mobile screen stream

从手机画面流中，获取一张当前屏幕截图

Returns: frame_data

get_frame()

teardown_stream()

snapshot(*ensure_orientation=True*, **args*, ***kwargs*)

Take a screenshot and convert it into a cv2 image object

获取一张屏幕截图，并转化成 cv2 的图像对象

Returns: numpy.ndarray

airtest.core.android.cap_methods.javacap module

class Javacap(*adb*, **args*, ***kwargs*)

基类：*airtest.core.android.yosemite.Yosemite*, *airtest.core.android.cap_methods.base_cap.BaseCap*

This is another screencap class, it is slower in performance than minicap, but it provides the better compatibility

APP_PKG = 'com.netease.nie.yosemite'

SCREENCAP_SERVICE = 'com.netease.nie.yosemite.Capture'

RECVTIMEOUT = None

get_frames()

Get the screen frames

返回 None

get_frame_from_stream()

Get frame from the stream

返回 frame

teardown_stream()

End stream

返回 None

airtest.core.android.cap_methods.minicap module

retry_when_socket_error(*func*)

class Minicap(*adb*, *projection=None*, *rotation_watcher=None*, *display_id=None*, *ori_function=None*)

基类: *airtest.core.android.cap_methods.base_cap.BaseCap*

super fast android screenshot method from stf minicap.

reference https://github.com/openstf/minicap

VERSION = 5

RECVTIMEOUT = None

CMD = 'LD_LIBRARY_PATH=/data/local/tmp /data/local/tmp/minicap'

install_or_upgrade()

Install or upgrade minicap

返回 None

uninstall()

Uninstall minicap

返回 None

install()

Install minicap

Reference: https://github.com/openstf/minicap/blob/master/run.sh

返回 None

get_frame(*projection=None*)

Get the single frame from minicap -s, this method slower than *get_frames*

1. shell cmd 1. remove log info 1.

Args: projection: screenshot projection, default is None which means using self.projection

Returns: jpg data

`get_stream`(*lazy=True*)

Get stream, it uses ‘adb forward‘and socket communication. Use minicap ‘‘lazy‘‘mode (provided by gzmaruijie) for long connections - returns one latest frame from the server

参数 **lazy** – True or False

Returns:

`get_frame_from_stream`()

Get one frame from minicap stream

返回 frame

`snapshot`(*ensure_orientation=True*, *projection=None*)

参数

- **ensure_orientation** – True or False whether to keep the orientation same as display
- **projection** – the size of the desired projection, (width, height)

Returns:

`update_rotation`(*rotation*)

Update rotation and reset the backend stream generator

参数 **rotation** – rotation input

返回 None

`teardown_stream`()

End the stream

返回 None

airtest.core.android.cap_methods.screen_proxy module

`class ScreenProxy`(*screen_method*)

基类：`object`

Perform screen operation according to the specified method

`SCREEN_METHODS = {'ADBCAP': <class 'airtest.core.android.cap_methods.adbcap.AdbCap'>, 'JAVACAP': <c`

`classmethod register_method`(*name*, *method_class*)

classmethod check_frame(*cap_method*)

Test whether a frame of image can be obtained correctly

测试能否正确获取一帧图像

参数 **cap_method** – *airtest.core.android.cap_methods.base_cap.BaseCap*

Returns:

classmethod auto_setup(*adb*, *default_method=None*, **args*, ***kwargs*)

In order of priority, try to initialize all registered screenshot methods, select an available method to return

按优先顺序，尝试初始化注册过的所有屏幕截图方法，选择一个可用方法返回

Custom method 自定义方法 > MINICAP > JAVACAP > ADBCAP

参数

- **adb** – *airtest.core.android.adb.ADB*
- **default_method** – String such as "MINICAP", or *airtest.core.android.cap_methods.minicap.Minicap* object

Returns: ScreenProxy object

实际案例

```
>>> dev = Android()
>>> screen_proxy = ScreenProxy.auto_setup(dev.adb, rotation_watcher=dev.
↪rotation_watcher)
>>> screen_proxy.get_frame_from_stream()
>>> screen_proxy.teardown_stream()
```

register_screen()

1.3.2 Submodules

1.3.2.1 airtest.core.android.adb module

class ADB(*serialno=None*, *adb_path=None*, *server_addr=None*, *display_id=None*, *input_event=None*)

基类：object

adb client object class

status_device = 'device'

status_offline = 'offline'

`SHELL_ENCODING = 'utf-8'`

`static builtin_adb_path()`

Return built-in adb executable path

返回 adb executable path

`start_server()`

Perform *adb start-server* command to start the adb server

返回 None

`kill_server()`

Perform *adb kill-server* command to kill the adb server

返回 None

`version()`

Perform *adb version* command and return the command output

返回 command output

`start_cmd(`*cmds*, *device=True*`)`

Start a subprocess with adb command(s)

参数

- **cmds** – command(s) to be run
- **device** – if True, the device serial number must be specified by *-s serialno* argument

Raises `RuntimeError` – if *device* is True and serialno is not specified

返回 a subprocess

`cmd(`*cmds*, *device=True*, *ensure_unicode=True*, *timeout=None*`)`

Run the adb command(s) in subprocess and return the standard output

参数

- **cmds** – command(s) to be run
- **device** – if True, the device serial number must be specified by -s serialno argument
- **ensure_unicode** – encode/decode unicode of standard outputs (stdout, stderr)
- **timeout** – timeout in seconds

Raises

- `DeviceConnectionError` – if any error occurs when connecting the device
- `AdbError` – if any other adb error occurs

返回 command(s) standard output (stdout)

`close_proc_pipe`(*proc*)

close stdin/stdout/stderr of subprocess.Popen.

`devices`(*state=None*)

Perform *adb devices* command and return the list of adb devices

参数 **state** – optional parameter to filter devices in specific state

返回 list od adb devices

`connect`(*force=False*)

Perform *adb connect* command, remote devices are preferred to connect first

参数 **force** – force connection, default is False

返回 None

`disconnect`()

Perform *adb disconnect* command

返回 None

`get_status`()

Perform *adb get-state* and return the device status

Raises `AdbError` – if status cannot be obtained from the device

返回 None if status is *not found*, otherwise return the standard output from *adb get-state* command

`wait_for_device`(*timeout=5*)

Perform *adb wait-for-device* command

参数 **timeout** – time interval in seconds to wait for device

Raises `DeviceConnectionError` – if device is not available after timeout

返回 None

`start_shell`(*cmds*)

Handle *adb shell* command(s)

参数 **cmds** – adb shell command(s)

返回 None

`raw_shell`(*cmds*, *ensure_unicode=True*)

Handle *adb shell* command(s) with unicode support

参数

- **cmds** – adb shell command(s)
- **ensure_unicode** – decode/encode unicode True or False, default is True

返回 command(s) output

`shell`(*cmd*)

Run the *adb shell* command on the device

参数 `cmd` – a command to be run

Raises `AdbShellError` – if command return value is non-zero or if any other *AdbError* occurred

返回 command output

`keyevent`(*keyname*)

Perform *adb shell input keyevent* command on the device

参数 `keyname` – key event name

返回 None

`getprop`(*key*, *strip=True*)

Perform *adb shell getprop* on the device

参数

- `key` – key value for property
- `strip` – True or False to strip the return carriage and line break from returned string

返回 propery value

`sdk_version`

`push`(*local*, *remote*)

Perform *adb push* command

参数

- `local` – local file to be copied to the device
- `remote` – destination on the device where the file will be copied

返回 None

`pull`(*remote*, *local*)

Perform *adb pull* command :param remote: remote file to be downloaded from the device :param local: local destination where the file will be downloaded from the device

返回 None

`forward`(*local*, *remote*, *no_rebind=True*)

Perform *adb forward* command

参数

- `local` – local tcp port to be forwarded
- `remote` – tcp port of the device where the local tcp port will be forwarded

- **no_rebind** – True or False

返回 None

get_forwards()

Perform ‘adb forward –list‘command

Yields serial number, local tcp port, remote tcp port

返回 None

classmethod get_available_forward_local()

Generate a pseudo random number between 11111 and 20000 that will be used as local forward port

返回 integer between 11111 and 20000

注解: use *forward –no-rebind* to check if port is available

---

setup_forward(***kwargs*)

remove_forward(*local=None*)

Perform *adb forward –remove* command

参数 **local** – local tcp port

返回 None

install_app(*filepath*, *replace=False*, *install_options=None*)

Perform *adb install* command

参数

- **filepath** – full path to file to be installed on the device
- **replace** – force to replace existing application, default is False

  e.g.[“-t” , # allow test packages ” -l” , # forward lock application, “-s” , # install application on sdcard, “-d” , # allow version code downgrade (debuggable packages only) “-g” , # grant all runtime permissions

  ]

返回 command output

install_multiple_app(*filepath*, *replace=False*, *install_options=None*)

Perform *adb install-multiple* command

参数

- **filepath** – full path to file to be installed on the device
- **replace** – force to replace existing application, default is False

- **install_options** – list of options e.g.["-t" , # allow test packages

  " -l" , # forward lock application, "-s" , # install application on sdcard, "-d" , # allow version code downgrade (debuggable packages only) "-g" , # grant all runtime permissions "-p" , # partial application install (install-multiple only)

  ]

返回 command output

pm_install(*filepath*, *replace=False*)

Perform *adb push* and *adb install* commands

注解: This is more reliable and recommended way of installing *.apk* files

---

参数

- **filepath** – full path to file to be installed on the device
- **replace** – force to replace existing application, default is False

返回 None

uninstall_app(*package*)

Perform *adb uninstall* command :param package: package name to be uninstalled from the device

返回 command output

pm_uninstall(*package*, *keepdata=False*)

Perform *adb uninstall* command and delete all related application data

参数

- **package** – package name to be uninstalled from the device
- **keepdata** – True or False, keep application data after removing the app from the device

返回 command output

snapshot()

Take the screenshot of the device display

返回 command output (stdout)

touch(*tuple_xy*)

Perform user input (touchscreen) on given coordinates

参数 **tuple_xy** – coordinates (x, y)

返回 None

`swipe`(*tuple_x0y0*, *tuple_x1y1*, *duration=500*)

Perform user input (swipe screen) from start point (x,y) to end point (x,y)

参数

- **tuple_x0y0** – start point coordinates (x, y)
- **tuple_x1y1** – end point coordinates (x, y)
- **duration** – time interval for action, default 500

Raises `AirtestError` – if SDK version is not supported

返回 None

`logcat`(*grep_str=''*, *extra_args=''*, *read_timeout=10*)

Perform *adb shell logcat* command and search for given patterns

参数

- **grep_str** – pattern to filter from the logcat output
- **extra_args** – additional logcat arguments
- **read_timeout** – time interval to read the logcat, default is 10

Yields logcat lines containing filtered patterns

返回 None

`exists_file`(*filepath*)

Check if the file exits on the device

参数 **filepath** – path to the file

返回 True or False if file found or not

`file_size`(*filepath*)

Get the file size

参数 **filepath** – path to the file

返回 The file size

Raises AdbShellError if no such file

`line_breaker`

Set carriage return and line break property for various platforms and SDK versions

返回 carriage return and line break string

`display_info`

Set device display properties (orientation, rotation and max values for x and y coordinates)

Notes: if there is a lock screen detected, the function tries to unlock the device first

返回 device screen properties

`get_display_info()`

Get information about device physical display (orientation, rotation and max values for x and y coordinates)

返回

device screen properties e.g {

'width': 1440, 'height': 2960, 'density': 4.0, 'orientation': 3, 'rotation': 270, 'max_x': 4095, 'max_y': 4095

}

`getMaxXY()`

Get device display maximum values for x and y coordinates

返回 max x and max y coordinates

`getRestrictedScreen()`

Get value for mRestrictedScreen (without black border / virtual keyboard)'

返回 screen resolution mRestrictedScreen value as tuple (x, y)

`getPhysicalDisplayInfo()`

Get value for display dimension and density from *mPhysicalDisplayInfo* value obtained from *dumpsys* command.

返回 physical display info for dimension and density

`getDisplayOrientation()`

Another way to get the display orientation, this works well for older devices (SDK version 15)

返回 display orientation information

`update_cur_display(`*display_info*`)`

Some phones support resolution modification, try to get the modified resolution from dumpsys adb shell dumpsys window displays | find "cur="

本方法虽然可以更好地获取到部分修改过分辨率的手机信息但是会因为 cur=(d+)x(d+) 的数值在不同设备上 width 和 height 的顺序可能不同，导致横竖屏识别出现问题 airtest 不再使用本方法作为通用的屏幕尺寸获取方法，但依然可用于部分设备获取当前被修改过的分辨率

实际案例

```
>>> # 部分三星和华为设备，若分辨率没有指定为最高，可能会导致点击偏移，可以用这个方式强制修改:
>>> # For some Samsung and Huawei devices, if the resolution is not specified␣
↪as the highest,
>>> # it may cause click offset, which can be modified in this way:
```

(下页继续)

(续上页)

```
>>> dev = device()
>>> info = dev.display_info
>>> info2 = dev.adb.update_cur_display(info)
>>> dev.display_info.update(info2)
```

参数 **display_info** – the return of self.getPhysicalDisplayInfo()

返回 display_info

get_top_activity()

Perform *adb shell dumpsys activity top* command search for the top activity

Raises **AirtestError** – if top activity cannot be obtained

返回 (package_name, activity_name, pid)

返回类型 top activity as a tuple

is_keyboard_shown()

Perform *adb shell dumpsys input_method* command and search for information if keyboard is shown

返回 True or False whether the keyboard is shown or not

is_screenon()

Perform *adb shell dumpsys window policy* command and search for information if screen is turned on or off

Raises **AirtestError** – if screen state can't be detected

返回 True or False whether the screen is turned on or off

is_locked()

Perform *adb shell dumpsys window policy* command and search for information if screen is locked or not

Raises **AirtestError** – if lock screen can't be detected

返回 True or False whether the screen is locked or not

unlock()

Perform *adb shell input keyevent MENU* and *adb shell input keyevent BACK* commands to attempt to unlock the screen

返回 None

警告: Might not work on all devices

`get_package_version`(*package*)

Perform *adb shell dumpsys package* and search for information about given package version

参数 `package` – package name

返回 None if no info has been found, otherwise package version

`list_app`(*third_only=False*)

Perform *adb shell pm list packages* to print all packages, optionally only those whose package name contains the text in FILTER.

Options -f: see their associated file -d: filter to only show disabled packages -e: filter to only show enabled packages -s: filter to only show system packages -3: filter to only show third party packages -i: see the installer for the packages -u: also include uninstalled packages

参数 `third_only` – print only third party packages

返回 list of packages

`path_app`(*package*)

Perform *adb shell pm path* command to print the path to the package

参数 `package` – package name

Raises

- `AdbShellError` – if any adb error occurs
- `AirtestError` – if package is not found on the device

返回 path to the package

`check_app`(*package*)

Perform *adb shell dumpsys package* command and check if package exists on the device

参数 `package` – package name

Raises `AirtestError` – if package is not found

返回 True if package has been found

`start_app`(*package*, *activity=None*)

Perform *adb shell monkey* commands to start the application, if *activity* argument is *None*, then *adb shell am start* command is used.

参数

- `package` – package name
- `activity` – activity name

返回 None

`start_app_timing`(*package*, *activity*)

Start the application and activity, and measure time

参数

- **package** – package name
- **activity** – activity name

返回 app launch time

`stop_app`(*package*)

Perform *adb shell am force-stop* command to force stop the application

参数 **package** – package name

返回 None

`clear_app`(*package*)

Perform *adb shell pm clear* command to clear all application data

参数 **package** – package name

返回 None

`text`(*content*)

Use adb shell input for text input

参数 **content** – text to input

返回 None

`get_ip_address`()

Perform several set of commands to obtain the IP address.

- *adb shell netcfg | grep wlan0*
- *adb shell ifconfig*
- *adb getprop dhcp.wlan0.ipaddress*

返回 None if no IP address has been found, otherwise return the IP address

`get_gateway_address`()

Perform several set of commands to obtain the gateway address.

- *adb getprop dhcp.wlan0.gateway*
- *adb shell netcfg | grep wlan0*

返回 None if no gateway address has been found, otherwise return the gateway address

`get_memory`()

`get_storage()`

`get_cpuinfo()`

`get_cpufreq()`

`get_cpuabi()`

`get_gpu()`

`get_model()`

`get_manufacturer()`

`get_device_info()`

Get android device information, including: memory/storage/display/cpu/gpu/model/manufacturer ...

返回 Dict of info

`get_display_of_all_screen`(*info*)

Perform *adb shell dumpsys window windows* commands to get window display of application.

参数 `info` – device screen properties

返回 None if adb command failed to run, otherwise return device screen properties(portrait mode) eg. (offset_x, offset_y, screen_width, screen_height)

`cleanup_adb_forward()`

1.3.2.2 airtest.core.android.android module

`class Android`(*serialno=None*, *host=None*, *cap_method='MINICAP'*, *touch_method='MINITOUCH'*, *ime_method='YOSEMITEIME'*, *ori_method='MINICAPORI'*, *display_id=None*, *input_event=None*)

基类: *airtest.core.device.Device*

Android Device Class

`touch_proxy`

根据 self.touch_method 的类型，执行对应的触摸操作

Module: *airtest.core.android.touch_methods.touch_proxy.TouchProxy*

返回 TouchProxy

实际案例

```
>>> dev = Android()
>>> dev.touch_proxy.touch((100, 100))  # If the device uses minitouch, it is␣
↪the same as dev.minitouch.touch
>>> dev.touch_proxy.swipe_along([(0,0), (100, 100)])
```

touch_method

In order to be compatible with the previous *dev.touch_method*

为了兼容以前的 'dev.touch_method'

返回 "MINITOUCH" or "MAXTOUCH"

实际案例

```
>>> dev = Android()
>>> print(dev.touch_method)  # "MINITOUCH"
```

cap_method

In order to be compatible with the previous *dev.cap_method*

为了兼容以前的 'dev.cap_method'

返回 "MINICAP" or "JAVACAP"

实际案例

```
>>> dev = Android()
>>> print(dev.cap_method)  # "MINICAP"
```

screen_proxy

Similar to touch_proxy, it returns a proxy that can automatically initialize an available screenshot method, such as Minicap

Afterwards, you only need to call `self.screen_proxy.get_frame()` to get the screenshot

类似 touch_proxy，返回一个代理，能够自动初始化一个可用的屏幕截图方法，例如 Minicap

后续只需要调用"self.screen_proxy.get_frame()"即可获取到屏幕截图

Returns: ScreenProxy(Minicap())

实际案例

```
>>> dev = Android()
>>> img = dev.screen_proxy.get_frame_from_stream()  # dev.minicap.get_frame_
↪from_stream() is deprecated
```

get_deprecated_var(*old_name*, *new_name*)

Get deprecated class variables

When the airtest version number>=1.1.2, the call device.minicap/device.javacap is removed, and relevant compatibility is made here, and DeprecationWarning is printed

airtest 版本号 >=1.1.2 时，去掉了 device.minicap/device.javacap 这样的调用，在此做了相关的兼容，并打印 DeprecationWarning

Usage: Android.minicap=property(lambda self: self.get_deprecated_var(“minicap” , “screen_proxy”))

参数

- **old_name** – “minicap”
- **new_name** – “screen_proxy”

返回

New implementation of deprecated object, e.g self.minicap -> self.screen_proxy

dev.minicap.get_frame_from_stream() -> dev.screen_proxy.get_frame_from_stream()

实际案例

```
>>> dev = Android()
>>> isinstance(dev.minicap, ScreenProxy)  # True
>>> dev.minicap.get_frame_from_stream()  # --> dev.screen_proxy.get_frame_from_
↪stream()
```

get_default_device()

获取本地默认连接的设备，当没有传入设备序列号时

返回 本地设备序列号 serialno

uuid

Serial number

返回

list_app(*third_only=False*)

返回 packages 列表

参数 **third_only** – 如果为 True，只返回所有第三方应用列表

返回 应用列表

`path_app`(*package*)

打印出 package 的完整路径

参数 **package** – package name

返回 package 的完整路径

`check_app`(*package*)

检查 package 在设备中是否存在

参数 **package** – package name

返回 如果存在，返回 True

Raises `AirtestError` – raised if package is not found

`start_app`(*package*, *activity=None*)

启动应用

参数

- **package** – package name
- **activity** – activity name

返回 None

`start_app_timing`(*package*, *activity*)

启动应用，并且返回启动耗费时间

参数

- **package** – package name
- **activity** – activity name

返回 app 启动时间

`stop_app`(*package*)

停止应用

参数 **package** – package name

返回 None

`clear_app`(*package*)

清理应用数据

参数 **package** – package name

返回 None

`install_app`(*filepath*, *replace=False*, *install_options=None*)

将应用安装到手机上

参数

- **filepath** – *apk* 文件在 PC 上的完整路径
- **replace** – 如果应用已存在，是否替换
- **install_options** – install 命令的额外选项，默认是 []

返回 安装进程的输出内容

install_multiple_app(*filepath*, *replace=False*, *install_options=None*)

Install multiple the application on the device

参数

- **filepath** – *apk* 文件在 PC 上的完整路径
- **replace** – 如果应用已存在，是否替换
- **install_options** – install 命令的额外选项，默认是 []

返回 安装进程的输出内容

uninstall_app(*package*)

从设备中卸载应用

参数 **package** – package name

返回 卸载进程中的输出内容

snapshot(*filename=None*, *ensure_orientation=True*, *quality=10*, *max_size=None*)

截取一张当前手机画面，默认会发送到 stdout

参数

- **filename** – 保存截图的文件名，默认为 None 的话将会发送到 stdout
- **ensure_orientation** – 截图方向是否要与当前显示情况一致，默认为 True
- **quality** – The image quality, integer in range [1, 99]
- **max_size** – the maximum size of the picture, e.g 1200

返回 截图输出

shell(**args*, ***kwargs*)

返回 *adb shell* 解释器

参数

- ***args** – optional shell commands
- ****kwargs** – optional shell commands

返回 None

keyevent(*keyname*, ***kwargs*)

在设备上执行 keyevent

参数

- **keyname** – keyevent name
- ****kwargs** – optional arguments

返回 None

wake()

执行唤醒操作

返回 None

home()

按下 HOME 键

返回 None

text(*text*, *enter=True*, ***kwargs*)

向设备中输入字符串

参数

- **text** – 要输入的字符串
- **enter** – 是否按下 'Enter'键
- **search** – 是否要按下输入法键盘中的 search 键

返回 None

touch(*pos*, *duration=0.01*)

在设备上执行点击

参数

- **pos** – coordinates (x, y)
- **duration** – 点击屏幕的时长

返回 None

double_click(*pos*)

swipe(*p1*, *p2*, *duration=0.5*, *steps=5*, *fingers=1*)

在设备上执行滑动操作

参数

- **p1** – 开始坐标
- **p2** – 结束坐标
- **duration** – 在屏幕上滑动的时长，默认是 0.5
- **steps** – 滑动过程中的步数，默认为 5
- **fingers** – 滑动的手指数量，1 或者 2，默认为 1

返回 None

`pinch`(*center=None*, *percent=0.5*, *duration=0.5*, *steps=5*, *in_or_out='in'*)

在设备上执行 pinch 操作（放大缩小），仅适用于 minitouch 和 maxtouch

参数

- **center** – pinch 操作的中心点
- **percent** – pinch 操作捏合屏幕的距离，默认是 0.5
- **duration** – 滑动过程中的时间间隔，默认是 0.8
- **steps** – 滑动过程中的步数，默认为 5
- **in_or_out** – 向内捏合 in 或者向外捏合 out，默认是’in’

返回 None

Raises `TypeError` – An error occurred when center is not a list/tuple or None

`swipe_along`(*coordinates_list*, *duration=0.8*, *steps=5*)

执行一段连续的滑动操作，仅适用于 minitouch 和 maxtouch

参数

- **coordinates_list** – 一个坐标的列表：[(x1, y1), (x2, y2), (x3, y3)]
- **duration** – 滑动过程中的时间间隔，默认是 0.8
- **steps** – 滑动过程中的步数，默认为 5

返回 None

`two_finger_swipe`(*tuple_from_xy*, *tuple_to_xy*, *duration=0.8*, *steps=5*, *offset=(0, 50)*)

执行两个手指一起滑动的操作，仅适用于 minitouch 和 maxtouch

参数

- **tuple_from_xy** – 开始坐标
- **tuple_to_xy** – 结束坐标
- **duration** – 滑动过程中的时间间隔，默认是 0.8
- **steps** – 滑动过程中的步数，默认为 5
- **offset** – 第二个手指相对于第一个手指的偏移坐标，默认是 (0, 50)

返回 None

`logcat`(**args*, ***kwargs*)

执行 *logcat*

参数

- ***args** – optional arguments

- ****kwargs** – optional arguments

返回 *logcat* 输出

getprop(*key*, *strip=True*)

根据传入的 key 获取 properties 内容

参数

- **key** – key name
- **strip** – 是否对输出内容进行 strip

返回 property value(s)

get_ip_address()

执行以下几种命令行来获取 IP 地址

- *adb shell netcfg | grep wlan0*
- *adb shell ifconfig*
- *adb getprop dhcp.wlan0.ipaddress*

返回 如果获取 IP 失败，返回 None，否则返回 IP 地址

get_top_activity()

Get the top activity

返回 (package, activity, pid)

get_top_activity_name()

Get the top activity name

返回 package/activity

is_keyboard_shown()

如果软键盘正在启用，返回 True，否则 False

Notes

不一定在所有设备上都可用

返回 True or False

is_screenon()

如果屏幕是亮着的，返回 True，否则 False

Notes

不一定在所有设备上都可用

返回 True or False

`is_locked()`
如果是锁定状态返回 True，否则 False

Notes

部分设备上可能不可用

返回 True or False

`unlock()`
解锁设备

Notes

不一定在所有设备上都可用

返回 None

`display_info`
返回显示信息 (width, height, orientation 和 max_x, max_y)

返回 显示信息

`get_display_info()`
返回显示信息 (width, height, orientation 和 max_x, max_y)

返回 显示信息

`get_current_resolution()`
返回旋转后的当前分辨率

返回 宽, 高

`get_render_resolution(`*refresh=False*`)`
返回旋转后的渲染分辨率

参数 **refresh** – 是否强制刷新渲染分辨率

返回 offset_x, offset_y, offset_width and offset_height of the display

`start_recording(`*max_time=1800*, *bit_rate_level=1*, *bit_rate=None*`)`
开始对设备画面进行录制

参数

- **max_time** – maximum screen recording time, default is 1800
- **bit_rate_level** – bit_rate=resolution*level, 0 < level <= 5, default is 1
- **bit_rate** – the higher the bitrate, the clearer the video

返回 None

实际案例

Record 30 seconds of video and export to the current directory test.mp4:

```
>>> from airtest.core.api import connect_device, sleep
>>> dev = connect_device("Android:///")
>>> # Record the screen with the lowest quality
>>> dev.start_recording(bit_rate_level=1)
>>> sleep(30)
>>> dev.stop_recording(output="test.mp4")
```

Or set max_time=30, the screen recording will stop automatically after 30 seconds:

```
>>> dev.start_recording(max_time=30, bit_rate_level=5)
>>> dev.stop_recording(output="test_30s.mp4")
```

The default value of *max_time* is 1800 seconds, so the maximum screen recording time is half an hour. You can modify its value to obtain a longer screen recording:

```
>>> dev.start_recording(max_time=3600, bit_rate_level=5)
>>> dev.stop_recording(output="test_hour.mp4")
```

stop_recording(*output='screen.mp4'*, *is_interrupted=False*)

停止对设备画面的录制。录制出的文件将会放在设备中。

参数

- **output** – default file is *screen.mp4*
- **is_interrupted** – True or False. Stop only, no pulling recorded file from device.

返回 None

javacap

maxtouch

minicap

minitouch

adjust_all_screen()

对全面屏设备进行渲染分辨率的调整。

返回 None

1.3.2.3 airtest.core.android.constant module

class CAP_METHOD
基类：object

MINICAP = 'MINICAP'

ADBCAP = 'ADBCAP'

JAVACAP = 'JAVACAP'

class TOUCH_METHOD
基类：object

MINITOUCH = 'MINITOUCH'

MAXTOUCH = 'MAXTOUCH'

ADBTOUCH = 'ADBTOUCH'

class IME_METHOD
基类：object

ADBIME = 'ADBIME'

YOSEMITEIME = 'YOSEMITEIME'

class ORI_METHOD
基类：object

ADB = 'ADBORI'

MINICAP = 'MINICAPORI'

1.3.2.4 airtest.core.android.ime module

ensure_unicode(*value*)
Decode UTF-8 values

参数 value – value to be decoded

返回 decoded valued

class CustomIme(*adb*, *apk_path*, *service_name*)
基类：object

Input Methods Class Object

start()
Enable input method

返回 None

end()

Disable input method

返回 None

text(*value*)

class YosemiteIme(*adb*)

基类: *airtest.core.android.ime.CustomIme*

Yosemite Input Method Class Object

start()

Enable input method

返回 None

text(*value*)

Input text with Yosemite input method

参数 **value** – text to be inputted

返回 output form *adb shell* command

code(*code*)

Sending editor action

参数 **code** – editor action code, e.g., 2 = IME_ACTION_GO, 3 = IME_ACTION_SEARCH Editor Action Code Ref: http://developer.android.com/reference/android/view/inputmethod/EditorInfo.html

返回 output form *adb shell* command

1.3.2.5 airtest.core.android.javacap module

1.3.2.6 airtest.core.android.minicap module

1.3.2.7 airtest.core.android.recorder module

class Recorder(*adb*)

基类: *airtest.core.android.yosemite.Yosemite*

Screen recorder

start_recording(***kwargs*)

stop_recording(*output='screen.mp4', is_interrupted=False*)

Stop screen recording

参数

- **output** – default file is *screen.mp4*

- **is_interrupted** – True or False. Stop only, no pulling recorded file from device.

Raises AirtestError – if recording was not started before

返回 None

pull_last_recording_file(*output='screen.mp4'*)

Pull the latest recording file from device. Error raises if no recording files on device.

参数 **output** – default file is *screen.mp4*

1.3.2.8 airtest.core.android.rotation module

class RotationWatcher(*adb*, *ori_method='MINICAPORI'*)

基类: object

RotationWatcher class

get_ready()

install()

Install the RotationWatcher package

返回 None

uninstall()

Uninstall the RotationWatcher package

返回 None

setup_server()

Setup rotation wacher server

返回 server process

teardown()

start()

Start the RotationWatcher daemon thread

返回 initial orientation

reg_callback(*ow_callback*)

参数 **ow_callback** –

Returns:

class XYTransformer

基类: object

transform the coordinates (x, y) by orientation (upright <–> original)

static up_2_ori(*tuple_xy*, *tuple_wh*, *orientation*)

Transform the coordinates upright –> original

参数

- **tuple_xy** – coordinates (x, y)
- **tuple_wh** – screen width and height
- **orientation** – orientation

返回 transformed coordinates (x, y)

static ori_2_up(*tuple_xy*, *tuple_wh*, *orientation*)

Transform the coordinates original –> upright

参数

- **tuple_xy** – coordinates (x, y)
- **tuple_wh** – screen width and height
- **orientation** – orientation

返回 transformed coordinates (x, y)

1.3.2.9 airtest.core.android.yosemite module

class Yosemite(*adb*)

基类：object

Wrapper class of Yosemite.apk, used by javacap/recorder/yosemite_ime.

install_or_upgrade()

Install or update the Yosemite.apk file on the device

返回 None

get_ready()

uninstall()

Uninstall *Yosemite.apk* application from the device

返回 None

1.4 airtest.core.ios package

This package provide IOS Device Class.

为 iOS 平台提供的接口，请参考：airtest.core.ios.IOS class

1.4.1 Submodules

1.4.1.1 airtest.core.ios.constant module

class CAP_METHOD
基类：object

MINICAP = 'MINICAP'

WDACAP = 'WDACAP'

MJPEG = 'MJPEG'

class TOUCH_METHOD
基类：object

WDATOUCH = 'WDATOUCH'

class IME_METHOD
基类：object

WDAIME = 'WDAIME'

1.4.1.2 airtest.core.ios.elements_type module

1.4.1.3 airtest.core.ios.fake_minitouch module

1.4.1.4 airtest.core.ios.instruct_cmd module

class InstructHelper(*uuid=None*)
基类：object

ForwardHelper class or help run other Instruction

static builtin_iproxy_path()

usb_device
Whether the current iOS uses the local USB interface, if so, return the wda.usbmux.Device object 当前 iOS 是否使用了本地 USB 接口，如果是，返回 wda.usbmux.Device 对象 Returns: wda.usbmux.Device or None

tear_down()

setup_proxy(***kwargs*)

remove_proxy(*local_port*)

do_proxy(*port*, *device_port*)
Start do proxy of ios device and self device 目前只支持本地 USB 连接的手机进行端口转发，远程手机暂时不支持:returns: None

`do_proxy_usbmux`(*lport*, *rport*)

Mapping ports of local USB devices using python multithreading 使用 python 多线程对本地 USB 设备的端口进行映射（当前仅使用 USB 连接一台 iOS 时才可用）

参数

- **lport** – local port
- **rport** – remote port

Returns:

1.4.1.5 airtest.core.ios.ios module

`decorator_retry_session`(*func*)

When the operation fails due to session failure, try to re-acquire the session, retry at most 3 times

当因为 session 失效而操作失败时，尝试重新获取 session，最多重试 3 次

`decorator_retry_for_class`(*cls*)

Add decorators to all methods in the class

为 class 里的所有 method 添加装饰器 `decorator_retry_session`

`class IOS`(*addr='http://localhost:8100/'*, *cap_method='MJPEG'*, *mjpeg_port=None*)

基类：*airtest.core.device.Device*

ios client

- before this you have to run WebDriverAgent
- `xcodebuild -project path/to/WebDriverAgent.xcodeproj -scheme WebDriverAgentRunner -destination "id=$(idevice_id -l)" test`
- `iproxy $port 8100 $udid`

`ip`

Returns the IP address of the host connected to the iOS phone It is not the IP address of the iOS phone. If you want to get the IP address of the phone, you can access the interface *get_ip_address*

For example: when the device is connected via http://localhost:8100, return localhost If it is a remote device http://192.168.xx.xx:8100, it returns the IP address of 192.168.xx.xx

Returns:

`uuid`

`using_ios_tagent`

当前基础版本：appium/WebDriverAgent 4.1.4 基于上述版本，2022.3.30 之后发布的 iOS-Tagent 版本，在/status 接口中新增了一个 Version 参数，如果能检查到本参数，说明使用了新版本 ios-Tagent 该版本基于 Appium 版的 WDA 做了一些改动，可以更快地进行点击和滑动，并优化了部

分 UI 树的获取逻辑但是所有的坐标都为竖屏坐标，需要 airtest 自行根据方向做转换同时，大于 4.1.4 版本的 appium/WebDriverAgent 不再需要针对 ipad 进行特殊的横屏坐标处理了 Returns:

is_pad

Determine whether it is an ipad(or 6P/7P/8P), if it is, in the case of horizontal screen + desktop, the coordinates need to be switched to vertical screen coordinates to click correctly (WDA bug)

判断是否是 ipad(或 6P/7P/8P)，如果是，在横屏 + 桌面的情况下，坐标需要切换成竖屏坐标才能正确点击（WDA 的 bug）Returns:

device_info

get the device info.

注解: Might not work on all devices

返回

dict for device info, eg. AttrDict({

'timeZone': 'GMT+0800', 'currentLocale': 'zh_CN', 'model': 'iPhone', 'uuid': '90CD6AB7-11C7-4E52-B2D3-61FA31D791EC', 'userInterfaceIdiom': 0, 'userInterfaceStyle': 'light', 'name': 'iPhone', 'isSimulator': False})

window_size()

return window size namedtuple:

Size(width , height)

orientation

return device oritantation status in LANDSACPE POR

get_orientation()

display_info

touch_factor

get_render_resolution()

Return render resolution after rotation

返回 offset_x, offset_y, offset_width and offset_height of the display

get_current_resolution()

home()

snapshot(*filename=None*, *quality=10*, *max_size=None*)

take snapshot

参数

- **filename** – save screenshot to filename
- **quality** – The image quality, integer in range [1, 99]
- **max_size** – the maximum size of the picture, e.g 1200

Returns:

get_frame_from_stream()

touch(*pos*, *duration=0.01*)

参数

- **pos** – coordinates (x, y), can be float(percent) or int
- **duration** (*optional*) – tap_hold duration

Returns: None

实际案例

```
>>> touch((100, 100))
>>> touch((0.5, 0.5), duration=1)
```

double_click(*pos*)

swipe(*fpos*, *tpos*, *duration=0*, **args*, ***kwargs*)

参数

- **fpos** – start point
- **tpos** – end point
- **duration** (*float*) – start coordinate press duration (seconds), default is 0

返回 None

实际案例

```
>>> swipe((1050, 1900), (150, 1900))
>>> swipe((0.2, 0.5), (0.8, 0.5))
```

keyevent(*keyname*, ***kwargs*)

Perform keyevent on the device

参数

- **keyname** – home/volumeUp/volumeDown

- ****kwargs** –

Returns:

press(*keys*)

some keys in [“home” , “volumeUp” , “volumeDown”] can be pressed

text(*text*, *enter=True*)

Input text on the device :param text: text to input :param enter: True if you need to enter a newline at the end

返回 None

实际案例

```
>>> text("test")
>>> text("中文")
```

install_app(*uri*, *package*)

curl -X POST $JSON_HEADER -d “{ “desiredCapabilities” :{ “bundleId” :” com.apple.mobilesafari” , “app” :” [host_path]/magicapp.app” }}” $DEVICE_URL/session https://github.com/facebook/WebDriverAgent/wiki/Queries

start_app(*package*, **args*)

参数 **package** – the app bundle id, e.g `com.apple.mobilesafari`

返回 None

实际案例

```
>>> start_app('com.apple.mobilesafari')
```

stop_app(*package*)

参数 **package** – the app bundle id, e.g `com.apple.mobilesafari`

Returns:

app_state(*package*)

参数 **package** –

返回

{ “value” : 4, “sessionId” : “0363BDC5-4335-47ED-A54E-F7CCB65C6A65”

}

value 1(not running) 2(running in background) 3(running in foreground)? 4(running)

实际案例

```
>>> dev = device()
>>> start_app('com.apple.mobilesafari')
>>> print(dev.app_state('com.apple.mobilesafari')["value"])  # --> output is 4
>>> home()
>>> print(dev.app_state('com.apple.mobilesafari')["value"])  # --> output is 3
>>> stop_app('com.apple.mobilesafari')
>>> print(dev.app_state('com.apple.mobilesafari')["value"])  # --> output is 1
```

app_current()

get the app current

Notes

Might not work on all devices

返回

{ "pid" : 1281, " name" : "" , "bundleId" : "com.netease.cloudmusic" }

返回类型 current app state dict, eg

get_ip_address()

get ip address from webDriverAgent

返回 raise if no IP address has been found, otherwise return the IP address

device_status()

show status return by webDriverAgent Return dicts of infos

is_locked()

Return True or False whether the device is locked or not

Notes

Might not work on some devices

返回 True or False

unlock()

Unlock the device, unlock screen, double press home

Notes

Might not work on all devices

返回 None

`lock()`

lock the device, lock screen

Notes

Might not work on all devices

返回 None

`alert_accept()`

Alert accept-Actually do click first alert button

Notes

Might not work on all devices

返回 None

`alert_dismiss()`

Alert dissmiss-Actually do click second alert button

Notes

Might not work on all devices

返回 None

`alert_wait(time_counter=2)`

if alert apper in time_counter second it will return True,else return False (default 20.0) time_counter default is 2 seconds

Notes

Might not work on all devices

返回 None

`alert_buttons()`

get alert buttons text. .. rubric:: Notes

Might not work on all devices

返回 (“设置”, “好”)

返回类型 # example return

`alert_exists()`

get True for alert exists or False.

Notes

Might not work on all devices

返回 True or False

`alert_click(buttons)`

when Arg type is list, click the first match, raise ValueError if no match

eg. [“设置”, “信任”, “安装”]

Notes

Might not work on all devices

返回 None

`clear_app(package)`

`home_interface()`

get True for the device status is on home interface.

Reason: some devices can Horizontal screen on the home interface

Notes

Might not work on all devices

返回 True or False

`list_app(**kwargs)`

`shell(*args, **kwargs)`

`uninstall_app(package)`

1.4.1.6 airtest.core.ios.minicap module

`class MinicapIOS(udid=None, port=12345)`

基类: `object`

https://github.com/openstf/ios-minicap

`CAPTIMEOUT = None`

`setup()`

get_frames()

rotation is alwary right on iOS

list_devices()

1.4.1.7 airtest.core.ios.rotation module

class RotationWatcher(*iosHandle*)

基类：**object**

RotationWatcher class

get_ready()

teardown()

start()

Start the RotationWatcher daemon thread

返回 None

reg_callback(*ow_callback*)

参数 **ow_callback** –

Returns:

get_rotation()

class XYTransformer

基类：**object**

transform the coordinates (x, y) by orientation (upright <–> original)

static up_2_ori(*tuple_xy*, *tuple_wh*, *orientation*)

Transform the coordinates upright –> original

参数

- **tuple_xy** – coordinates (x, y)
- **tuple_wh** – current screen width and height
- **orientation** – orientation

返回 transformed coordinates (x, y)

static ori_2_up(*tuple_xy*, *tuple_wh*, *orientation*)

Transform the coordinates original –> upright

参数

- **tuple_xy** – coordinates (x, y)
- **tuple_wh** – current screen width and height

- **orientation** – orientation

返回 transformed coordinates (x, y)

1.4.1.8 airtest.core.ios.wda_client module

1.5 airtest.core.win package

为 Windows 平台提供的接口，请参考：airtest.core.win.Windows class

This package provide Windows Client Class.

1.5.1 Submodules

1.5.1.1 airtest.core.win.ctypesinput module

MagicMock is a subclass of Mock with default implementations of most of the magic methods. You can use MagicMock without having to configure the magic methods yourself.

If you use the *spec* or *spec_set* arguments then *only* magic methods that exist in the spec will be created.

Attributes and the return value of a *MagicMock* will also be *MagicMocks*.

1.5.1.2 airtest.core.win.screen module

screenshot(*filename*, *hwnd=None*)

Take the screenshot of Windows app

参数

- **filename** – file name where to store the screenshot
- **hwnd** –

返回 bitmap screenshot file

1.5.1.3 airtest.core.win.win module

require_app(*func*)

class Windows(*handle=None*, *dpifactor=1*, ***kwargs*)

基类：*airtest.core.device.Device*

Windows 客户端。

uuid

`connect`(*handle=None*, ***kwargs*)

连接到一个 windows 窗口并且把窗口置前

参数 **`**kwargs`** – optional arguments

返回 None

`shell`(*cmd*)

在 subprocess 里运行命令行

参数 **`cmd`** – 需要运行的命令行

Raises `subprocess.CalledProcessError` – when command returns non-zero exit status

返回 命令行的输出内容作为 bytes string 返回

`snapshot`(*filename=None*, *quality=10*, *max_size=None*)

截取一张图片并且保存到 ST.LOG_DIR 文件夹中

参数

- **`filename`** – 截图的文件名，默认为 {time}.jpg
- **`quality`** – The image quality, integer in range [1, 99]
- **`max_size`** – the maximum size of the picture, e.g 1200

返回 截图的内容

`keyevent`(*keyname*, ***kwargs*)

执行一个按键响应

References

https://pywinauto.readthedocs.io/en/latest/code/pywinauto.keyboard.html

参数

- **`keyname`** – key event
- **`**kwargs`** – optional arguments

返回 None

`text`(*text*, ***kwargs*)

Input text

参数

- **`text`** – 待输入的字符串
- **`**kwargs`** – optional arguments

返回 None

key_press(*key*)

模拟一个按下按键的事件。

发送键盘扫描码至计算机来告知哪个按键被按下。一些游戏使用 DirectInput 设备，只响应键盘扫描码，而不是虚拟键码。可以用 key_press() 方法来模拟发送键盘扫描码，而不是上述发送虚拟键码的 keyevent() 方法。

参数 **key** – 一个字符串来表示哪个按键将被按下。可用的选择有：{ 'ESCAPE' , '1' , '2' , '3' , '4' , '5' , '6' , '7' , '8' , '9' , '0' , '-' , '=' , 'BACKSPACE' , 'TAB' , 'Q' , 'W' , 'E' , 'R' , 'T' , 'Y' , 'U' , 'I' , 'O' , 'P' , '[' , ']' , 'ENTER' , 'LCTRL' , 'A' , 'S' , 'D' , 'F' , 'G' , 'H' , 'J' , 'K' , 'L' , ';' , '"' , '`' , 'LSHIFT' , 'BACKSLASH' , 'Z' , 'X' , 'C' , 'V' , 'B' , 'N' , 'M' , ',' , '.' , '/' , 'RSHIFT' , '*' , 'LALT' , 'SPACE' , 'CAPS_LOCK' , 'F1' , 'F2' , 'F3' , 'F4' , 'F5' , 'F6' , 'F7' , 'F8' , 'F9' , 'F10' , 'NUM_LOCK' , 'SCROLL_LOCK' , 'NUMPAD_7' , 'NUMPAD_8', 'NUMPAD_9', 'NUMPAD_-', 'NUMPAD_4', 'NUMPAD_5', 'NUMPAD_6', 'NUMPAD_+', 'NUMPAD_1', 'NUMPAD_2', 'NUMPAD_3' , 'NUMPAD_0' , 'NUMPAD_.' , 'F11' , 'F12' , 'PRINT_SCREEN' , 'PAUSE' , 'NUMPAD_ENTER' , 'RCTRL' , 'NUMPAD_/' , 'RALT' , 'HOME' , 'UP' , 'PAGE_UP' , 'LEFT' , 'RIGHT' , 'END' , 'DOWN' , 'PAGE_DOWN' , 'INSERT' , 'DELETE' , 'LWINDOWS' , 'RWINDOWS' , 'MENU' }.

key_release(*key*)

模拟一个释放按键的事件。

发送键盘扫描码至计算机来告知哪个按键被释放。一些游戏使用 DirectInput 设备，只响应键盘扫描码，而不是虚拟键码。可以用 key_release() 方法来模拟发送键盘扫描码。一般情况下 key_release() 方法与所释放按键的 key_press() 方法搭配使用。

参数 **key** – 一个字符串来表示哪个按键将被释放。

touch(*pos*, ***kwargs*)

执行鼠标点击操作

References

https://pywinauto.readthedocs.io/en/latest/code/pywinauto.mouse.html

参数

- **pos** – 点击位置的坐标
- ****kwargs** – optional arguments

返回 None

double_click(*pos*)

swipe(*p1*, *p2*, *duration=0.8*, *steps=5*)

执行拖动操作（鼠标按下并且释放）

参数

- **p1** – 起始点坐标
- **p2** – 终点坐标
- **duration** – 执行滑动操作的时间间隔
- **steps** – 滑动操作的步数

返回 None

mouse_move(*pos*)

模拟一个移动鼠标的事件。

已知的 bug: 因为 pywinauto 库存在的 bug，用户可能在使用此方法时遇到移动后位置与目标位置的 x 和 y 坐标有 1 个像素点偏差的情况。

参数 pos – 一个 (x, y) 的 tuple, 其中 x 和 y 分别表示目标位置在屏幕上的 x 和 y 坐标。

mouse_down(*button='left'*)

模拟一个按下鼠标按键的事件。

参数 button – 一个字符串来表示将按下哪个鼠标按键。有以下的鼠标按键选项：{ 'left' , 'middle' , 'right' }。

mouse_up(*button='left'*)

模拟一个释放鼠标按键的事件。

一般情况下 mouse_up() 方法与所释放鼠标按键的 mouse_down() 方法搭配使用。

参数 button – 一个字符串来表示将释放哪个鼠标按键。

start_app(*path*, ***kwargs*)

启动应用

参数

- **path** – 应用的完整路径
- **kwargs** – reference: https://pywinauto.readthedocs.io/en/latest/code/pywinauto.application.html#pywinauto.application.Application.start

返回 None

stop_app(*pid*)

关闭应用

参数 pid – 需要被关闭应用的 process ID

返回 None

`set_foreground()`
将窗口切换到最前

返回 None

`get_rect()`
获取窗口的边界矩形的尺寸

返回 win32structures.RECT

`get_title()`
获取窗口标题

返回 窗口标题

`get_pos()`
获取窗口位置的坐标

返回 窗口左上角坐标 (left, top)

`move(pos)`
移动窗口到指定坐标

参数 `pos` – 要移动到的目标位置坐标 (x, y)

返回 None

`kill()`
杀死应用进程

返回 None

`focus_rect`

`get_current_resolution()`

`get_ip_address()`
获取应用的外部 IP 地址。

返回 ip address

返回类型 :py：obj：’ str’

1.6 airtest

1.6.1 airtest package

1.6.1.1 Subpackages

airtest.aircv package

Submodules

airtest.aircv.aircv module

imread(*filename*, *flatten=False*)
根据图片路径，将图片读取为 cv2 的图片处理格式.

imwrite(*filename*, *img*, *quality=10*, *max_size=None*)
写出图片到本地路径，压缩

show(*img*, *title='show_img'*, *test_flag=False*)
在可缩放窗口里显示图片.

show_origin_size(*img*, *title='image'*, *test_flag=False*)
原始尺寸窗口中显示图片.

rotate(*img*, *angle=90*, *clockwise=True*)
函数使图片可顺时针或逆时针旋转 90、180、270 度. 默认 clockwise=True：顺时针旋转

crop_image(*img*, *rect*)
区域截图，同时返回截取结果和截取偏移; Crop image , rect = [x_min, y_min, x_max ,y_max]. (airtest 中有用到)

mark_point(*img*, *point*, *circle=False*, *color=100*, *radius=20*)
调试用的: 标记一个点

mask_image(*img*, *mask*, *color=(255, 255, 255)*, *linewidth=-1*)
将 screen 的 mask 矩形区域刷成白色 gbr(255, 255, 255). 其中 mask 区域为: [x_min, y_min, x_max, y_max]. color: 顺序分别的 blue-green-red 通道. linewidth: 为-1 时则完全填充填充，为正整数时为线框宽度.

get_resolution(*img*)

airtest.aircv.cal_confidence module

These functions calculate the similarity of two images of the same size.

cal_ccoeff_confidence(*im_source*, *im_search*)
求取两张图片的可信度，使用 TM_CCOEFF_NORMED 方法.

cal_rgb_confidence(*img_src_rgb*, *img_sch_rgb*)
同大小彩图计算相似度.

airtest.aircv.error module

Declaration: Define all BaseError Classes used in aircv.

exception BaseError(*message=''*)

基类：Exception

Base class for exceptions in this module.

exception FileNotExistError(*message=''*)

基类：*airtest.aircv.error.BaseError*

Image does not exist.

exception TemplateInputError(*message=''*)

基类：*airtest.aircv.error.BaseError*

Resolution input is not right.

exception NoSIFTModuleError(*message=''*)

基类：*airtest.aircv.error.BaseError*

Resolution input is not right.

exception NoSiftMatchPointError(*message=''*)

基类：*airtest.aircv.error.BaseError*

Exception raised for errors 0 sift points found in the input images.

exception SiftResultCheckError(*message=''*)

基类：*airtest.aircv.error.BaseError*

Exception raised for errors 0 sift points found in the input images.

exception HomographyError(*message=''*)

基类：*airtest.aircv.error.BaseError*

In homography, find no mask, should kill points which is duplicate.

exception NoModuleError(*message=''*)

基类：*airtest.aircv.error.BaseError*

Resolution input is not right.

exception NoMatchPointError(*message=''*)

基类：*airtest.aircv.error.BaseError*

Exception raised for errors 0 keypoint found in the input images.

exception MatchResultCheckError(*message=''*)

基类：*airtest.aircv.error.BaseError*

Exception raised for errors 0 keypoint found in the input images.

airtest.aircv.keypoint_base module

Detect keypoints with KAZE.

class KeypointMatching(*im_search*, *im_source*, *threshold=0.8*, *rgb=True*)

基类: object

基于特征点的识别基类: KAZE.

METHOD_NAME = 'KAZE'

FILTER_RATIO = 0.59

ONE_POINT_CONFI = 0.5

mask_kaze()

基于 kaze 查找多个目标区域的方法.

find_all_results()

基于 kaze 查找多个目标区域的方法.

find_best_result(**args*, ***kwargs*)

show_match_image()

Show how the keypoints matches.

init_detector()

Init keypoint detector object.

get_keypoints_and_descriptors(*image*)

获取图像特征点和描述符.

match_keypoints(*des_sch*, *des_src*)

Match descriptors (特征值匹配).

airtest.aircv.keypoint_matching module

Detect keypoints with KAZE/AKAZE/BRISK/ORB. No need for opencv-contrib module.

class KAZEMatching(*im_search*, *im_source*, *threshold=0.8*, *rgb=True*)

基类: *airtest.aircv.keypoint_base.KeypointMatching*

KAZE Matching.

class BRISKMatching(*im_search*, *im_source*, *threshold=0.8*, *rgb=True*)

基类: *airtest.aircv.keypoint_base.KeypointMatching*

BRISK Matching.

METHOD_NAME = 'BRISK'

init_detector()

Init keypoint detector object.

class AKAZEMatching(*im_search*, *im_source*, *threshold=0.8*, *rgb=True*)

基类: *airtest.aircv.keypoint_base.KeypointMatching*

AKAZE Matching.

`METHOD_NAME = 'AKAZE'`

`init_detector()`
Init keypoint detector object.

`class ORBMatching(im_search, im_source, threshold=0.8, rgb=True)`
基类: *airtest.aircv.keypoint_base.KeypointMatching*

ORB Matching.

`METHOD_NAME = 'ORB'`

`init_detector()`
Init keypoint detector object.

airtest.aircv.keypoint_matching_contrib module

Detect keypoints with BRIEF/SIFT/SURF. Need opencv-contrib module.

`check_cv_version_is_new()`
opencv 版本是 3.0 或 4.0 以上, API 接口与 2.0 的不同.

`class BRIEFMatching(im_search, im_source, threshold=0.8, rgb=True)`
基类: *airtest.aircv.keypoint_base.KeypointMatching*

FastFeature Matching.

`METHOD_NAME = 'BRIEF'`

`init_detector()`
Init keypoint detector object.

`get_keypoints_and_descriptors(image)`
获取图像特征点和描述符.

`match_keypoints(des_sch, des_src)`
Match descriptors (特征值匹配).

`class SIFTMatching(im_search, im_source, threshold=0.8, rgb=True)`
基类: *airtest.aircv.keypoint_base.KeypointMatching*

SIFT Matching.

`METHOD_NAME = 'SIFT'`

`FLANN_INDEX_KDTREE = 0`

`init_detector()`
Init keypoint detector object.

get_keypoints_and_descriptors(*image*)
获取图像特征点和描述符.

match_keypoints(*des_sch*, *des_src*)
Match descriptors (特征值匹配).

class SURFMatching(*im_search*, *im_source*, *threshold=0.8*, *rgb=True*)
基类：*airtest.aircv.keypoint_base.KeypointMatching*

SURF Matching.

METHOD_NAME = 'SURF'

UPRIGHT = 0

HESSIAN_THRESHOLD = 400

FLANN_INDEX_KDTREE = 0

init_detector()
Init keypoint detector object.

get_keypoints_and_descriptors(*image*)
获取图像特征点和描述符.

match_keypoints(*des_sch*, *des_src*)
Match descriptors (特征值匹配).

airtest.aircv.sift module

find_sift(*im_source*, *im_search*, *threshold=0.8*, *rgb=True*, *good_ratio=0.59*)
基于 sift 进行图像识别，只筛选出最优区域.

mask_sift(*im_source*, *im_search*, *threshold=0.8*, *rgb=True*, *good_ratio=0.59*)
基于 sift 查找多个目标区域的方法.

find_all_sift(*im_source*, *im_search*, *threshold=0.8*, *rgb=True*, *good_ratio=0.59*)
基于 sift 查找多个目标区域的方法.

airtest.aircv.template module

模板匹配.

对用户提供的调节参数:

1. threshod: 筛选阈值，默认为 0.8
2. rgb: 彩色三通道, 进行彩色权识别.

find_template(*im_source*, *im_search*, *threshold=0.8*, *rgb=False*)
函数功能：找到最优结果.

find_all_template(*im_source*, *im_search*, *threshold=0.8*, *rgb=False*, *max_count=10*)
根据输入图片和参数设置, 返回所有的图像识别结果.

airtest.aircv.template_matching module

模板匹配.

对用户提供的调节参数:

1. threshod: 筛选阈值，默认为 0.8
2. rgb: 彩色三通道, 进行彩色权识别.

class TemplateMatching(*im_search*, *im_source*, *threshold=0.8*, *rgb=True*)
基类：`object`

模板匹配.

`METHOD_NAME = 'Template'`

`MAX_RESULT_COUNT = 10`

find_all_results(**args*, ***kwargs*)

find_best_result(**args*, ***kwargs*)

airtest.aircv.utils module

print_run_time(*func*)

generate_result(*middle_point*, *pypts*, *confi*)
Format the result: 定义图像识别结果格式.

check_image_valid(*im_source*, *im_search*)
Check if the input images valid or not.

check_source_larger_than_search(*im_source*, *im_search*)
检查图像识别的输入.

img_mat_rgb_2_gray(*img_mat*)
Turn img_mat into gray_scale, so that template match can figure the img data. "print(type(im_search[0][0])") can check the pixel type.

img_2_string(*img*)

string_2_img(*pngstr*)

pil_2_cv2(*pil_image*)

cv2_2_pil(*cv2_image*)

compress_image(*pil_img*, *path*, *quality*, *max_size=None*)

Save the picture and compress

参数

- **pil_img** – PIL image
- **path** – save path
- **quality** – the image quality, integer in range [1, 99]
- **max_size** – the maximum size of the picture, e.g 1200

返回

airtest.cli package

Submodules

airtest.cli.info module

get_script_info(*script_path*)

extract info from script, like basename, ___author___, ___title___ and ___desc___.

get_author_title_desc(*text*)

Get author title desc.

process_desc(*desc*)

strip_str(*string*)

Strip string.

airtest.cli.parser module

get_parser()

runner_parser(*ap=None*)

cli_setup(*args=None*)

future api for setup env by cli

airtest.cli.runner module

class AirtestCase(*methodName='runTest'*)

基类: unittest.case.TestCase

PROJECT_ROOT = '.'

SCRIPTEXT = '.air'

TPLEXT = '.png'

classmethod setUpClass()
Hook method for setting up class fixture before running tests in the class.

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

runTest()

classmethod exec_other_script(*scriptpath*)
run other script in test script

setup_by_args(*args*)

run_script(*parsed_args*, *testcase_cls=<class 'airtest.cli.runner.AirtestCase'>*)

airtest.core package

Subpackages

airtest.core.linux package

This package provide Windows Client Class.

Submodules

airtest.core.linux.linux module

class Linux(*pid=None*, ***kwargs*)
基类：*airtest.core.device.Device*

Linux desktop.

shell(*cmd*)
Run shell command in subprocess

参数 **cmd** – command to be run

Raises subprocess.CalledProcessError – when command returns non-zero exit status

返回 command output as a byte string

snapshot(*filename='tmp.png'*, *quality=None*)
Take a screenshot and save it to *tmp.png* filename by default

参数

- **filename** – name of file where to store the screenshot
- **quality** – ignored

返回 display the screenshot

keyevent(*keyname*, ***kwargs*)

Perform a key event

References

https://pywinauto.readthedocs.io/en/latest/code/pywinauto.keyboard.html

参数

- **keyname** – key event
- ****kwargs** – optional arguments

返回 None

text(*text*, ***kwargs*)

Input text

参数

- **text** – text to input
- ****kwargs** – optional arguments

返回 None

touch(*pos*, ***kwargs*)

Perform mouse click action

References

https://pywinauto.readthedocs.io/en/latest/code/pywinauto.mouse.html

参数

- **pos** – coordinates where to click
- ****kwargs** – optional arguments

返回 None

double_click(*pos*)

swipe(*p1*, *p2*, *duration=0.8*, *steps=5*)

Perform swipe (mouse press and mouse release) :param p1: start point :param p2: end point :param duration: time interval to perform the swipe action :param steps: size of the swipe step

返回 None

`start_app`(*path*)

Start the application

参数 **path** – full path to the application

返回 None

`stop_app`(*pid*)

Stop the application

参数 **pid** – process ID of the application to be stopped

返回 None

`get_current_resolution`()

`get_ip_address`()

Return default external ip address of the linux os.

返回 ip address

返回类型 str

Submodules

airtest.core.cv module

"Airtest 图像识别专用.

`loop_find`(*query*, *timeout=20*, *threshold=None*, *interval=0.5*, *intervalfunc=None*)

Search for image template in the screen until timeout

参数

- **query** – image template to be found in screenshot
- **timeout** – time interval how long to look for the image template
- **threshold** – default is None
- **interval** – sleep interval before next attempt to find the image template
- **intervalfunc** – function that is executed after unsuccessful attempt to find the image template

Raises TargetNotFoundError – when image template is not found in screenshot

返回 TargetNotFoundError if image template not found, otherwise returns the position where the image template has been found in screenshot

`try_log_screen`(*screen=None*, *quality=None*, *max_size=None*)

Save screenshot to file

参数

- **screen** – screenshot to be saved
- **quality** – The image quality, default is ST.SNAPSHOT_QUALITY
- **max_size** – the maximum size of the picture, e.g 1200

返回 filename, "resolution" : aircv.get_resolution(screen)}

返回类型 { "screen"

class Template(*filename*, *threshold=None*, *target_pos=5*, *record_pos=None*, *resolution=()*, *rgb=False*, *scale_max=800*, *scale_step=0.005*)

基类: object

picture as touch/swipe/wait/exists target and extra info for cv match filename: pic filename target_pos: ret which pos in the pic record_pos: pos in screen when recording resolution: screen resolution when recording rgb: 识别结果是否使用 rgb 三通道进行校验. scale_max: 多尺度模板匹配最大范围. scale_step: 多尺度模板匹配搜索步长.

filepath

match_in(*screen*)

match_all_in(*screen*)

class Predictor

基类: object

this class predicts the press_point and the area to search im_search.

DEVIATION = 100

static count_record_pos(*pos*, *resolution*)

计算坐标对应的中点偏移值相对于分辨率的百分比.

classmethod get_predict_point(*record_pos*, *screen_resolution*)

预测缩放后的点击位置点.

classmethod get_predict_area(*record_pos*, *image_wh*, *image_resolution=()*, *screen_resolution=()*)

Get predicted area in screen.

airtest.core.device module

class MetaDevice

基类: type

REGISTRY = {'Android': <class 'airtest.core.android.android.Android'>, 'Device': <class 'airtest.c

class Device

基类: object

base class for test device

uuid

shell(**args*, ***kwargs*)

snapshot(**args*, ***kwargs*)

touch(*target*, ***kwargs*)

double_click(*target*)

swipe(*t1*, *t2*, ***kwargs*)

keyevent(*key*, ***kwargs*)

text(*text*, *enter=True*)

start_app(*package*, ***kwargs*)

stop_app(*package*)

clear_app(*package*)

list_app(***kwargs*)

install_app(*uri*, ***kwargs*)

uninstall_app(*package*)

get_current_resolution()

get_render_resolution()

get_ip_address()

airtest.core.error module

error classes

exception BaseError(*value*)

基类: Exception

exception AirtestError(*value*)

基类: *airtest.core.error.BaseError*

This is Airtest BaseError

exception InvalidMatchingMethodError(*value*)

基类: *airtest.core.error.BaseError*

This is InvalidMatchingMethodError BaseError When an invalid matching method is used in settings.

exception TargetNotFoundError(*value*)

基类: *airtest.core.error.AirtestError*

This is TargetNotFoundError BaseError When something is not found

exception ScriptParamError(*value*)

基类: *airtest.core.error.AirtestError*

This is ScriptParamError BaseError When something goes wrong

exception AdbError(*stdout*, *stderr*)

基类: Exception

This is AdbError BaseError When ADB have something wrong

exception AdbShellError(*stdout*, *stderr*)

基类: *airtest.core.error.AdbError*

adb shell error

exception DeviceConnectionError(*value*)

基类: *airtest.core.error.BaseError*

device connection error

```
DEVICE_CONNECTION_ERROR = "error:\\s*((device \\'\\S+\\' not found)|(cannot connect to daemon at [\
```

exception ICmdError(*stdout*, *stderr*)

基类: Exception

This is ICmdError BaseError When ICmd have something wrong

exception ScreenError(*value*)

基类: *airtest.core.error.BaseError*

When the screen capture method(Minicap/Javacap/ScreenProxy) has something wrong

exception MinicapError(*value*)

基类: *airtest.core.error.ScreenError*

This is MinicapError BaseError When Minicap have something wrong

exception MinitouchError(*value*)

基类: *airtest.core.error.BaseError*

This is MinitouchError BaseError When Minicap have something wrong

exception PerformanceError(*value*)

基类: *airtest.core.error.BaseError*

airtest.core.helper module

class G

基类: object

Represent the globals variables

BASEDIR = []

LOGGER = <airtest.utils.logwraper.AirtestLogger object>

LOGGING = <Logger airtest.core.api (DEBUG)>

SCREEN = None

DEVICE = None

DEVICE_LIST = []

RECENT_CAPTURE = None

RECENT_CAPTURE_PATH = None

CUSTOM_DEVICES = {}

classmethod add_device(*dev*)

Add device instance in G and set as current device.

实际案例

G.add_device(Android())

参数 **dev** – device to init

返回 None

classmethod register_custom_device(*device_cls*)

set_logdir(*dirpath*)

set log dir for logfile and screenshots.

参数 **dirpath** – directory to save logfile and screenshots

Returns:

log(*arg*, *timestamp=None*, *desc=''*, *snapshot=False*)

Insert user log, will be displayed in Html report.

参数

- **arg** – log message or Exception object
- **timestamp** – the timestamp of the log, default is time.time()
- **desc** – description of log, default is arg.class.__name__
- **snapshot** – whether to take a screenshot, default is False

返回 None

实际案例

```
>>> log("hello world", snapshot=True)
>>> log({"key": "value"}, timestamp=time.time(), desc="log dict")
>>> try:
        1/0
    except Exception as e:
        log(e)
```

logwrap(*f*)

device_platform(*device=None*)

using(*path*)

import_device_cls(*platform*)

lazy import device class

delay_after_operation()

airtest.core.settings module

class Settings

基类: object

DEBUG = False

LOG_DIR = None

LOG_FILE = 'log.txt'

static RESIZE_METHOD(*w*, *h*, *sch_resolution*, *src_resolution*, *design_resolution=(960, 640)*)

图像缩放规则: COCOS 中的 MIN 策略.

CVSTRATEGY = ['mstpl', 'tpl', 'surf', 'brisk']

KEYPOINT_MATCHING_PREDICTION = True

THRESHOLD = 0.7

THRESHOLD_STRICT = None

OPDELAY = 0.1

FIND_TIMEOUT = 20

FIND_TIMEOUT_TMP = 3

PROJECT_ROOT = ''

SNAPSHOT_QUALITY = 10

IMAGE_MAXSIZE = None

SAVE_IMAGE = True

airtest.report package

Submodules

airtest.report.report module

nl2br(*eval_ctx*, *value*)

timefmt(*timestamp*)

Formatting of timestamp in Jinja2 templates :param timestamp: timestamp of steps :return: "%Y-%m-%d %H:%M:%S"

class LogToHtml(*script_root*, *log_root=''*, *static_root=''*, *export_dir=None*, *script_name=''*, *log_file=None*, *lang='en'*, *plugins=None*)

基类：object

Convert log to html display

scale = 0.5

static init_plugin_modules(*plugins*)

classmethod get_thumbnail(*path*)

compress screenshot

classmethod get_small_name(*filename*)

static div_rect(*r*)

count rect for js use

is_pos(*v*)

copy_tree(*src*, *dst*, *ignore=None*)

get_relative_log(*output_file*)

Try to get the relative path of log.txt :param output_file: output file: log.html :return: ./log.txt or ''

get_console(*output_file*)

readFile(*filename*, *code='utf-8'*)

report_data(*output_file=None*, *record_list=None*)

Generate data for the report page

参数

- **output_file** – The file name or full path of the output file, default HTML_FILE

- **record_list** – List of screen recording files

返回

report(*template_name='log_template.html'*, *output_file='log.html'*, *record_list=None*)

Generate the report page, you can add custom data and overload it if needed

参数

- **template_name** – default is HTML_TPL
- **output_file** – The file name or full path of the output file, default HTML_FILE
- **record_list** – List of screen recording files

返回

simple_report(*filepath*, *logpath=True*, *logfile=None*, *output='log.html'*)

get_parger(*ap*)

main(*args*)

1.7 Device Connection

1.7.1 Supported Platforms 各平台支持情况

1.7.1.1 Overview

1.7.1.2 Android

Currently we are compatible with **most Android phones** (2.3 <= Android <= 11) on the market, and a few special Android tablets and devices.

- If you encounter problems during the connection, please refer to the device settings of this document according to the different mobile phone manufacturers: Some manufacturer's device special problems
- For MIUI 11 or above of Xiaomi mobile phone, please use `cap_method=JAVACAP` mode to connect to the phone

目前我们兼容市面上**绝大多数的 Android 手机**（2.3 <= Android <= 11），和少数特殊 Android 平板和设备。

- 如果在连接中遇到问题，请根据手机厂商的不同，查看此文档的设备设置：部分厂商设备特殊问题
- 小米手机的 MIUI 11 以上版本，请使用 `cap_method=JAVACAP` 模式连接手机

1.7.1.3 Android Emulator 模拟器

The following emulators have been verified, Android Emulator Connection Guidelines

下列模拟器都经过验证，Android 模拟器连接指引

- 夜神 Nox
- 网易 Mumu
- 逍遥 Memuplay
- iTools
- 腾讯手游助手
- BlueStacks
- AVD

附录：我们对常见模拟器版本的适配情况测试（2020.06，随着版本更新可能失效）

1.7.1.4 iOS

According to iOS-Tagent document, the current support situation:

以iOS-Tagent文档为准，目前的支持情况：

1.7.2 Android device connection methods and FAQs

中文版

1.7.2.1 Android phone connection

If use AirtestIDE to mobile phone connection, please refer to the documents

If you’ re not going to use AirtestIDE, have a look at this statement:

- Opens the **developer options** on the phone, and **allows USB debugging**
- After using USB to connect the phone, you can see the device using `adb devices` command and refer to *Use ADB to see if the phone is successfully connected*
- On the code and command line, connect the phone with the phone‘s serial number, see *Use the phone in your code*

Use ADB to see if the phone is successfully connected

`adb` is the official Android command line tool for Google, which allows us to communicate with devices.(if you are interested, please refer to: adb.)

We have stored `adb` executables for each platform under `airtest\airtest\core\Android\static\adb` directory, you can use it without downloading.

Take Windows as an example, you can first use the terminal to enter the directory where `adb.exe` is located (in `airtest\airtest\core\Android\static\adb\Windows`, `shift+right click` to open the command line terminal), and then execute the command line of `adb devices` :

```
E:\airtest\airtest\core\android\static\adb\windows>adb devices

List of devices attached
c2b1c2a7        device
eba17551        device
127.0.0.1:10033 device
```

In MAC, you can visit `airtest/core/android/static/adb/mac` directory and run the `./adb devices`, if the adb no executable permissions, can run `chmod + x adb` add executable permissions for it.

- In the above example, you can see that the 3 Android devices currently connected, whose state is `device`, are normally online
- If the device status is `UNAUTHORIZED`, click OK in the `ALLOW USB Debugging` menu that pops up over the phone
- If you can't see the device name, you may need to install the phone's official driver on your PC

If your phone has a connection problem

Due to different mobile phone manufacturers and the corresponding model, may encounter all sorts of problems in the process of connecting, please refer to the link common problems Android

Use the phone in your code

After confirming that the phone can be successfully connected, we can see the device serial number of the phone in the command line of `adb devices` :

```
> adb devices

List of devices attached
c2b1c2a7        device
```

The `c2B1c2A7` above is the device serial number of the mobile phone. We define a mobile phone with the following format string:

```
Android://<adbhost>:<adbport>/<serialno>
```

Among them:

- `adbhost` is the IP of the host where adb Server is located. By default, this is `localhost`or `127.0.0.1`

- `adb port` defaults to 5037
- `serialno` is the serial number of the Android phone, such as `c2B1c2a7` just now

Here are some examples:

```
# Will default to the first phone in the current connection if you fill in nothing
Android:///
# A phone with c2B1C2a7 connected to the default port of the native
Android://127.0.0.1:5037/c2b1c2a7
# Connect a remote device through ADB connect with the native ADB. Note that 10.254.60.
↪1:5555 is actually Serialno
Android://127.0.0.1:5037/10.254.60.1:5555
```

Connect the phone according to the `Android:///` string

When we run a script from the command line, we can use `--device Android:///` to specify the Android device on which the script will run, for example:

```
>airtest run untitled.air --device Android:/// phone serial number --log log/
```

In addition, we can use the `connect_device` interface when we want to connect the phone in our code:

```
from airtest.core.api import *
connect_device("Android:///Phone Serial Number")
```

These two methods only need to choose one of them, basically can meet our needs to connect devices.

Some special parameters

Some special devices may appear black screen when connected, such as some emulators, we can add an extra parameter `cap_method=JAVACAP` to force the screen capture mode to be `JAVACAP` :

```
# Connect the emulator and check the `Use Javacap` mode
Android://127.0.0.1:5037/127.0.0.1:7555?cap_method=JAVACAP
```

In addition, we have two other parameters, `ori_method=ADBORI`, which specifies the rotation mode of the device screen, and `touch_method=ADBTOUCH`, which specifies the click mode of the screen as ADB instruction.

For the most part, we don' t need to specify these parameters, and we only need to add additional parameters if some special Android devices (such as some special models of tablets) can't connect with the default parameters:

```
# Check all the options to connect the device and use && to connect multiple parameter␣
↪strings
Android://127.0.0.1:5037/79d03fa?cap_method=JAVACAP&&ori_method=ADBORI&&touch_
↪method=ADBTOUCH
```

Note: if any of the characters ^<>|&appear on the command line, they may need to be escaped to take effect.

Therefore, if you need to write && in the connection string, you need to rewrite it as ^&^& in Windows, add a ^ symbol for escape, and add \ for escape under MAC:

```
# -- device Android://127.0.0.1:5037/79d03fa?cap_method=JAVACAP&&ori_method=ADBORI is␣
↪not available under Windows
--device Android://127.0.0.1:5037/79d03fa?cap_method=JAVACAP^&^&ori_method=ADBORI #␣
↪Windows command line add ^ escape effect
--device Android://127.0.0.1:5037/79d03fa?cap_method=JAVACAP\&\&ori_method=ADBORI # MAC␣
↪command line add \ escape
```

1.7.2.2 Android interface calls

All interfaces defined in `airtest.core.api` can be used on the Android platform and can be called directly in the script:

```
from airtest.core.api import *
touch((100, 200))
# Start an application
start_app("org.cocos2d.blackjack")
# Pass in a key response
keyevent("BACK")
```

Can refer to airtest.core.api for the API list.

Android device interface

In addition to the cross-platform interface provided in `airtest.core.api`, Android device objects have many built-in interfaces that can be called,We can airtest core. Android. Android module in this document refer to the android device object has a method, and then call something like this:

```
dev = device() # gets the Android object to the current device
print(dev.get_display_info()) # to view the display information for the current device
print(dev.list_app()) # prints out the list of currently installed apps
```

The ADB instruction call

Using the Android device interface, we can call adb directives like this:

```
# Execute the instruction ADB shell LS on the current device
print(shell("ls"))

# Execute the ADB instruction for a specific device
dev = connect_device("Android:///device1")
dev.shell("ls")

# Switch to a device and execute adb instruction
set_current(0)
shell("ls")
```

1.7.2.3 Frequently asked Questions about Android

Android emulator connection

The simulator is connected in a similar way to the real machine. The following steps are required:

- Open developer options on the emulator and check to allow USB debugging. Some emulators may need to find `Settings - about the phone` multiple times before opening the developer options
- Use ADB to connect the corresponding port number, for example, enter `adb connect 127.0.0.1:62001`, where 7555 is the port number corresponding to the simulator, and each brand simulator is different
- you can use the code `Android://127.0.0.1:5037/127.0.0.1:62001?cap_method=JAVACAP` connects to the corresponding emulator

Key points to note:

- Most emulators cannot connect with default parameters and must specify `cap_method=JAVACAP`
- each brand simulator port can be refer to Android emulator

Slide continuously

We provide some sliding interfaces to facilitate more complex operations:

```
dev = device()  # gets the current device
dev.pinch()  # Two fingers pinch or separate
```

(下页继续)

(续上页)

```
dev.swipe_along([(100, 300), (300, 300), (100, 500), (300, 600)]) # continuously slides␣
↪over a series of coordinates
dev.two_finger_swipe((100, 100), (200, 200))  # both fingers slip together
```

Among them, `swipe_along` can continuously streak through a series of coordinate points, which is the most commonly used interface.

Custom slide

In `airtest.core.android.touch_methods.base_touch`, defines four action events:

- `DownEvent(Coordinates, contact=0, pressure=50)` click
- `UpEvent(contact=0)` finger up
- `MoveEvent(coordinates, contact=0, pressure=50)` slide to a coordinate
- `SleepEvent` wait (seconds)

In the above four actions, the `contact` parameter defaults to 0, representing the first finger. If 1 is passed in, the action of the second finger can be defined, so that the complex operation of the double-finger can be achieved.

`pressure=50` defines the pressure when pressed and defaults to 50.

The `touch` interface, for example, is actually made up of `[DownEvent, SleepEvent, UpEvent]` actions, which in theory can be combined to allow you to customize very complex click-and-slide operations.

For example, here's an example of a two-fingered tap on a screen:

```
from airtest.core.android.touch_methods.base_touch import *
# tap with two fingers
multitouch_event = [
     DownEvent((100, 100), 0),
     DownEvent((200, 200), 1),  # second finger
     SleepEvent(1),
     UpEvent(0), UpEvent(1)]

device().touch_proxy.perform(multitouch_event)
```

In the example code above, press the first finger at the coordinates of (100, 100), press the second finger at (200, 200), and wait for a second before lifting each finger.

Also, `MoveEvent` can be added to achieve more diversified operations, such as an ordinary `swipe` :

```
Swipe_event = [DownEvent((500, 500)), SleepEvent(0.1)]

for i in range(5):
     swipe_event.append(MoveEvent((500 + 100*i, 500 + 100*i)))
     Swipe_event. Append (SleepEvent (0.2))

swipe_event.append(UpEvent())

dev.touch_proxy.perform(swipe_event)
```

Based on this improvement, more complex operations can be achieved, such as long press 2 seconds - slide to a position:

```
from airtest.core.android.touch_methods.base_touch import *
dev = device()

# Long press delete application
longtouch_event = [
     DownEvent([908, 892]),  # coordinates of the application to be deleted
     SleepEvent(2),
     MoveEvent([165,285]),  # delete the application's garbage can coordinates
     UpEvent(0)]

dev.touch_proxy.perform(longtouch_event)
```

More examples, please refer to the airtest/playground/android_motionevents.py.

Debug tips

You can switch on `settings-developer options-show input position` on your phone to debug simulated inputs.

Record the screen while running the script

Android phones support recording the screen while running the script. Add the `--recording` parameter to the command line of running the script:

```
airtest run "D:\test\Airtest_example.air"  --device android:/// --log logs/ --recording
```

After running, you can find the mp4 file recorded in the specified log directory.

- If only the `--recording` parameter has been passed, by default `recording_serialnumber.mp4` will be used to name the recording screen file
- If the file name `--recording test.mp4` is specified and there is more than one phone, name it `serialnumber.mp4`
- If you specify the filename `--recording test.mp4` and have only one phone, call it `test.mp4`
- **Note that the file name passed in must end with mp4**
- The default screen recording file is up to 1800 seconds. If you need to record for a longer time, you need to manually call the screen recording interface in the code

If you call the screen recording interface in the code, you can control the clarity and duration of the screen recording. For the document, see Android.start_recording.

For example, to record a 30-second video with the lowest definition and export it to `test.mp4` in the current directory:

```
from airtest.core.api import connect_device, sleep
dev = connect_device("Android:///")
# Record the screen with the lowest quality
dev.start_recording(bit_rate_level=1)
sleep(30)
dev.stop_recording(output="test.mp4")
```

`bit_rate_level` is used to control the resolution of screen recording. The value range is 1-5. `bit_rate_level=5` has the highest resolution, but it will take up more hard disk space.

Or set the parameter `max_time=30`, the screen recording will automatically stop after 30 seconds:

```
dev = device()
dev.start_recording(max_time=30, bit_rate_level=5)
dev.stop_recording(output="test_30s.mp4")
```

The default value of `max_time` is 1800 seconds, so the maximum screen recording time is half an hour, you can modify its value to get a longer screen recording:

```
dev = device()
dev.start_recording(max_time=3600, bit_rate_level=5)
dev.stop_recording(output="test_hour.mp4")
```

1.7.2.4 Refer to the tutorial and documentation for more

- Automated Testing on Android Phones-part 1
- Android connection FAQ

1.7.3 Android 设备连接方法与常见代码示例

English version

1.7.3.1 Android 手机连接

若使用 AirtestIDE 进行手机连接，请参考文档

若不打算使用 AirtestIDE，可以参考以下步骤：

- 打开手机中的 **开发者选项**，以及 **允许 USB 调试**
- 使用 USB 连上手机后，能够使用 `adb devices` 命令看到设备，参考使用 *ADB* 查看手机是否成功连接
- 在代码和命令行中，使用手机序列号连接手机，参见在代码中使用手机

使用 ADB 查看手机是否成功连接

`adb` 是谷歌官方推出的 Android 命令行工具，可以让我们跟设备进行通信。(感兴趣的话，请参考：官方地址。)

我们已经在 `airtest\airtest\core\android\static\adb` 目录下，存放了各平台的 `adb` 可执行文件，大家无需下载也可以使用。

以 windows 为例，可以先使用终端进入到 `adb.exe` 所在的目录下（在 `airtest\airtest\core\android\static\adb\windows` 目录下 shift+ 右键打开命令行终端），然后执行 `adb devices` 命令行：

```
E:\airtest\airtest\core\android\static\adb\windows>adb devices

List of devices attached
c2b1c2a7        device
eba17551        device
127.0.0.1:10033 device
```

在 mac 中，可以访问 `airtest/core/android/static/adb/mac` 目录下，运行 `./adb devices`，若提示 adb 没有可执行权限，可以运行 `chmod +x adb` 为它添加可执行权限。

- 在上面这个例子中，可以看到当前连接的 3 台 Android 设备，状态为 `device` 就是正常在线
- 如果设备状态为 `unauthorized`，请在手机上弹出的允许 `USB` 调试菜单中点击同意
- 如果看不到设备名称，可能需要在 PC 上安装手机对应的官方驱动

手机连接遇到问题

由于手机对应的厂商和型号各不相同，可能在连接过程中会遇到各种各样的问题，请参考Android 连接常见问题

在代码中使用手机

确认手机能够成功连接后，我们能够在 `adb devices` 命令行的结果中看到手机的设备序列号：

```
> adb devices

List of devices attached
c2b1c2a7        device
```

上面的 `c2b1c2a7` 就是手机的设备序列号，我们用以下格式的字符串来定义一台手机：

```
Android://<adbhost>:<adbport>/<serialno>
```

其中：

- `adbhost` 是 adb server 所在主机的 ip，默认是本机，也就是 `localhost` 或 `127.0.0.1`
- `adb port` 默认是 5037
- `serialno` 是 android 手机的序列号，例如刚才的 `c2b1c2a7`

以下是一些示例：

```
# 什么都不填写，会默认取当前连接中的第一台手机
Android:///
# 连接本机默认端口连的一台设备号为 c2b1c2a7 的手机
Android://127.0.0.1:5037/c2b1c2a7
# 用本机的 adb 连接一台 adb connect 过的远程设备，注意 10.254.60.1:5555 其实是 serialno
Android://127.0.0.1:5037/10.254.60.1:5555
```

根据 Android:///字符串连接手机

当我们使用命令行运行脚本时，可以使用`--device Android:///`来为它指定脚本运行的 Android 设备，例如：

```
>airtest run untitled.air --device Android:///手机序列号 --log log/
```

除此之外，当我们想在代码里连接手机时，可以使用 `connect_device` 接口：

```
from airtest.core.api import *
connect_device("Android:///手机序列号")
```

这两种方式只需要选择其中一种，基本上都能满足我们连接设备的需求。

一些特殊参数

部分特殊设备在连接时可能会出现黑屏的情况，例如一些模拟器，我们可以额外添加 `cap_method=JAVACAP` 的参数来强制指定屏幕截图方式为 `JAVACAP`:

```
# 连接了模拟器，勾选了 `Use javacap`模式
Android://127.0.0.1:5037/127.0.0.1:7555?cap_method=JAVACAP
```

除此之外，我们还有另外两个参数，分别是用于指定设备画面旋转模式的 `ori_method=ADBORI`，以及指定点击画面方式为 ADB 指令点击的 `touch_method=ADBTOUCH`。

大部分情况下，我们无需指定这些参数，只有在一些特殊的 Android 设备（例如部分特殊型号的平板）上，使用默认参数无法连接时，才需要加入额外的参数:

```
# 所有的选项都勾选上之后连接的设备，用&& 来连接多个参数字符串
Android://127.0.0.1:5037/79d03fa?cap_method=JAVACAP&&ori_method=ADBORI&&touch_
↪method=ADBTOUCH
```

注意：命令行中如果有出现 `^ < > | &` 这些字符，可能都需要转义才能生效。

因此如果连接字符串中需要写 `&&` 时，在 windows 下需要改写成 `^&^&` ，添加一个 `^` 符号进行转义，在 mac 下则需要添加\进行转义:

```
# --device Android://127.0.0.1:5037/79d03fa?cap_method=JAVACAP&&ori_method=ADBORI 在
windows 下不可用
--device Android://127.0.0.1:5037/79d03fa?cap_method=JAVACAP^&^&ori_method=ADBORI  #␣
↪windows 命令行添加^ 转义后效果
--device Android://127.0.0.1:5037/79d03fa?cap_method=JAVACAP\&\&ori_method=ADBORI  # mac
命令行添加\转义
```

1.7.3.2 Android 接口调用

所有在 `airtest.core.api` 中定义的接口，都可以在 Android 平台上使用，直接在脚本中调用即可:

```
from airtest.core.api import *
touch((100, 200))
# 启动某个应用
```

(下页继续)

(续上页)

```
start_app("org.cocos2d.blackjack")
# 传入某个按键响应
keyevent("BACK")
```

可以查阅airtest.core.api文档获得 API 列表。

Android 设备接口

除了在 airtest.core.api 中提供的跨平台接口之外，Android 设备对象还有很多内置的接口可以调用，我们可以在airtest.core.android.android module这个文档中查阅到 Android 设备对象拥有的方法，然后像这样调用：

```
dev = device()  # 获取到当前设备的 Android 对象
print(dev.get_display_info())  # 查看当前设备的显示信息
print(dev.list_app())  # 打印出当前安装的 app 列表
```

ADB 指令调用

利用 Android 设备接口，我们可以这样调用 adb 指令：

```
# 对当前设备执行指令 adb shell ls
print(shell("ls"))

# 对特定设备执行 adb 指令
dev = connect_device("Android:///device1")
dev.shell("ls")

# 切换到某台设备，执行 adb 指令
set_current(0)
shell("ls")
```

1.7.3.3 Android 常见问题与代码示例

Android 模拟器连接

模拟器与真机的连接方式类似，需要进行以下步骤：

- 打开模拟器上的开发者选项，并勾选允许 USB 调试。部分模拟器可能需要找到 设置-关于手机点击多次后才能打开开发者选项
- 使用 adb 连上对应的端口号，例如输入 `adb connect 127.0.0.1:62001`，其中 7555 是模拟器对应的端口号，每个品牌模拟器不相同

- 可以使用代码 `Android://127.0.0.1:5037/127.0.0.1:62001?cap_method=JAVACAP` 连上对应的模拟器

注意要点：

- 大部分模拟器无法使用默认参数连接，必须要指定 `cap_method=JAVACAP`
- 各品牌模拟器的端口可以查阅Android 模拟器连接

连续滑动

我们提供了一些滑动方面的接口，方便大家进行更复杂的操作：

```
dev = device()  # 获取当前设备
dev.pinch()  # 双指捏合或分开
dev.swipe_along([(100, 300), (300, 300), (100, 500), (300, 600)])  # 连续滑过一系列坐标
dev.two_finger_swipe( (100, 100), (200, 200) )  # 两个手指一起滑动
```

其中，`swipe_along` 可以连续不断地划过一系列坐标点，是最常用的一个接口。

自定义滑动

在 `airtest.core.android.touch_methods.base_touch` 中，定义了 4 个动作事件：

- `DownEvent(coordinates, contact=0, pressure=50)` 手指按下
- `UpEvent(contact=0)` 手指抬起
- `MoveEvent(coordinates, contact=0, pressure=50)` 滑动到某个坐标
- `SleepEvent(seconds)` 等待

上述 4 个动作中，`contact` 参数默认为 0，代表了第一根手指，如果传入 1，就可以定义第二根手指的动作，这样就能实现双指的复杂操作了。

`pressure=50` 定义了按下时的压力，默认为 50。

例如 `touch` 接口，实际上是由 `[DownEvent, SleepEvent, UpEvent]` 三个动作组成的，理论上组合这些动作，能够自定义非常复杂的点击滑动操作。

例如这是一个双指轻点屏幕的例子：

```
from airtest.core.android.touch_methods.base_touch import *
# tap with two fingers
multitouch_event = [
    DownEvent((100, 100), 0),
    DownEvent((200, 200), 1),  # second finger
    SleepEvent(1),
```

(下页继续)

(续上页)

```
    UpEvent(0), UpEvent(1)]

device().touch_proxy.perform(multitouch_event)
```

在上面的示例代码中，先在 (100, 100) 的坐标按下第一个手指，在 (200, 200) 按下第二个手指，等待一秒后再分别抬起两个手指。

还可以加入 MoveEvent 来实现更丰富的操作，例如一个普通的 swipe 是这样实现的：

```
swipe_event = [DownEvent((500, 500)), SleepEvent(0.1)]

for i in range(5):
    swipe_event.append(MoveEvent((500 + 100*i, 500 + 100*i)))
    swipe_event.append(SleepEvent(0.2))

swipe_event.append(UpEvent())

dev.touch_proxy.perform(swipe_event)
```

在此基础上进行改进，可以实现更多复杂操作，例如长按 2 秒-滑动到某个位置：

```
from airtest.core.android.touch_methods.base_touch import *
dev = device()

# 长按删除应用
longtouch_event = [
    DownEvent([908, 892]),  # 待删除应用的坐标
    SleepEvent(2),
    MoveEvent([165,285]),  # 删除应用的垃圾桶坐标
    UpEvent(0)]

dev.touch_proxy.perform(longtouch_event)
```

更多示例，请参考airtest/playground/android_motionevents.py。

Debug tips

你可以打开手机设置-开发者选项-显示触摸位置来调试模拟输入的操作，这样做能看到每次点击的位置。

在运行脚本过程中录屏

Android 手机支持在运行脚本过程中对屏幕进行录制，在运行脚本的命令行中加入--recording 参数即可：

```
airtest run "D:\test\Airtest_example.air"  --device android:/// --log logs/ --recording
```

运行完毕后，可以在指定的 log 目录中找到录制完毕的 mp4 文件。

- 如果只传了--recording 参数，默认将会使用 recording_ 手机序列号.mp4 来命名录屏文件
- 如果指定了文件名--recording test.mp4，且超过一台手机，就命名为 手机序列号 _test.mp4
- 如果指定了文件名--recording test.mp4，且只有一台手机，就命名为 test.mp4
- **注意传入的文件名必须以 mp4 作为结尾**
- 默认录屏文件最长为 1800 秒，如果需要录制更长时间，需要手动在代码中调用录屏接口

如果在代码中调用录屏接口，能够控制录屏时的清晰度和时长，文档参见Android.start_recording。

例如，以最低清晰度录制一段 30 秒的视频，并导出到当前目录下的 test.mp4：

```
from airtest.core.api import connect_device, sleep
dev = connect_device("Android:///")
# Record the screen with the lowest quality
dev.start_recording(bit_rate_level=1)
sleep(30)
dev.stop_recording(output="test.mp4")
```

bit_rate_level 用于控制录屏的清晰度，取值范围是 1-5，bit_rate_level=5 清晰度最高，但是占用的硬盘空间也会更大。

或者设置参数 max_time=30，30 秒后将自动停止录屏：

```
dev = device()
dev.start_recording(max_time=30, bit_rate_level=5)
dev.stop_recording(output="test_30s.mp4")
```

max_time 默认值为 1800 秒，所以录屏最大时长是半小时，可以修改它的值以获得更长时间的录屏：

```
dev = device()
dev.start_recording(max_time=3600, bit_rate_level=5)
dev.stop_recording(output="test_hour.mp4")
```

1.7.3.4 更多参考教程和文档

- 如何在 Android 手机上进行自动化测试
- Android 连接常见问题

1.8 Code Example

1.8.1 Common problems and code examples

中文版

1.8.1.1 How to initialize the script

air script: auto_setup()

Automatically configure the interface of the running environment, you can configure the path where the current script is located, the device used, the storage path of the log content, the project root directory, and the screenshot compression accuracy:

```
auto_setup(basedir=None, devices=None, logdir=None, project_root=None, compress=None)
```

Interface example:

```
auto_setup(__file__)

auto_setup(__file__, devices=["android://127.0.0.1:5037/emulator-5554?cap_method=JAVACAP&
↪&ori_method=MINICAPORI&&touch_method=MINITOUCH"],
           logdir=True, project_root=r"D:\test", compress=90)
```

1.8.1.2 How to connect/use the device

Please note: For more device related information, please refer to Document

Connect the device: connect_device(URI)

The interface to connect to the device requires the URI string used to initialize the device. Example:

```
# Connect Android device
connect_device("Android://127.0.0.1:5037/SJE5T17B17")

# Connect iOS device
connect_device("iOS:///127.0.0.1:8100")

# Connect Windows window
connect_device("Windows:///123456")
```

(下页继续)

(续上页)

```
# Connect the simulator
connect_device("Android://127.0.0.1:5037/127.0.0.1:62001?cap_method=JAVACAP&&ori_
↪method=ADBORI")
```

Connect the device: init_device()

To initialize the interface of the device, you need to pass in the device platform, the uuid and optional parameters of the device, where uuid is the serial number of Android, the window handle of Windows, or the uuid of iOS:

```
init_device(platform='Android', uuid=None, **kwargs)
```

Example of interface usage:

```
# Connect Android device
init_device(platform="Android",uuid="SJE5T17B17",cap_method="JAVACAP")

# Connect Windows window
init_device(platform="Windows",uuid="123456")
```

Get the current device: device()

Return the device instance currently in use, the usage example is as follows:

```
dev = device()
dev.swipe_along([[959, 418],[1157, 564],[1044, 824],[751, 638],[945, 415]])
```

Set the current device: set_current()

Set the current device used, which can be used to switch between multiple devices. Examples are as follows:

```
# The first type: Incoming numbers 0, 1, 2, etc., switch the currently operating mobile␣
↪phone to the first and second mobile phone connected to Airtest
set_current(0)
set_current(1)

# Second: Switch the current mobile phone to the serial number serialno1, serialno2
```

(下页继续)

(续上页)

```
set_current("serialno1")
set_current("serialno2")
```

1.8.1.3 How to perform coordinate click/coordinate sliding

Coordinate click

When you click on the device, you can pass in parameters such as the click location and the number of clicks. The optional parameters under different platforms are slightly different. Examples are as follows:

```
# Pass in absolute coordinates as the click position
touch([100,100])

# Pass in the template picture instance, click the center of the screenshot
touch(Template(r"tpl1606730579419.png", target_pos=5, record_pos=(-0.119, -0.042),␣
↪resolution=(1080, 1920)))

# Click 2 times
touch([100,100],times=2)

# Under Android and Windows platforms, you can set the click duration
touch([100,100],duration=2)
```

Coordinate sliding

For sliding operations on the device, there are two ways to pass parameters, one is to pass in the starting point and end of the sliding, and the other is to pass in the starting point and sliding direction vector. Examples are as follows:

```
# Pass in absolute coordinates as the start and end of the sliding
swipe([378, 1460],[408, 892])

# Incoming image as a starting point, slide along a certain direction
swipe(Template(r"tpl1606814865574.png", record_pos=(-0.322, 0.412), resolution=(1080,␣
↪1920)), vector=[-0.0316, -0.3311])

# Commonly, you can also set the duration of sliding
swipe([378, 1460],[408, 892],duration=1)
```

1.8.1.4 How to install/start/uninstall apps

Start the application: start_app()

To start the target application on the device, you need to pass in the package name of the application, which supports Android and iOS platforms. Examples:

```
start_app("com.netease.cloudmusic")
```

Terminate application operation: stop_app()

To terminate the operation of the target application on the device, the package name of the application needs to be passed in. It supports Android and iOS platforms. Examples:

```
stop_app("com.netease.cloudmusic")
```

Clear application data: clear_app()

To clean up the target application data on the device, the package name of the application needs to be passed in. **Only supports the Android platform**, example:

```
clear_app("com.netease.cloudmusic")
```

Install the application: install()

To install the application on the device, you need to pass in the complete apk installation path, **only supports the Android platform**, example:

```
install(r"D:\demo\tutorial-blackjack-release-signed.apk")
```

Uninstall the application: uninstall()

To uninstall the application on the device, you need to pass in the package name of the uninstalled application. **Only supports the Android platform**, example:

```
uninstall("com.netease.cloudmusic")
```

1.8.1.5 Key event: keyevent

Android keyevent

It is equivalent to executing `adb shell input keyevent KEYNAME`, an example is as follows:

```
keyevent("HOME")
# The constant corresponding to the home key is 3
keyevent("3") # same as keyevent("HOME")
keyevent("BACK")
keyevent("KEYCODE_DEL")
```

Windows keyevent

Unlike Android, the Windows platform uses the `pywinauto.keyboard module` module for key input. Example:

```
keyevent("{DEL}")

# Use Alt+F4 to close the Windows window
keyevent("%{F4}")
```

iOS keyevent

Currently only supports HOME key:

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

1.8.1.6 How to enter text

To enter text on the device, the text box needs to be activated (that is, click the text box first, and then use the `text()` interface to enter). Examples are as follows:

```
touch (Template instance of text box)
text("input text")

# By default, text is with carriage return, you can pass False if you don' t need it
text("123",enter=False)
```

(下页继续)

(续上页)

```
# Under the Android platform, you can also click the search button on the soft keyboard␣
↪after typing
text("123",enter=False,search=True)
```

1.8.1.7 How to delete text

Delete a single character through the `keyevent` interface:

```
keyevent("KEYCODE_DEL")
keyevent("67") # 67 is the delete key, please note that the input is a string
```

Simulate clearing the input box operation:

```
for i in range(10):
    keyevent("67")
```

Delete poco (leave the input box blank):

```
poco("xxx").set_text("").
```

1.8.1.8 Log

How to log into the report

The `log()` interface is convenient to insert some user-defined log information, which will be displayed in the Airtest report. In Airtest version 1.1.6, the log interface supports 4 parameters:

- `args`, which can be a string, a non-string or a `traceback` object;
- `timestamp`, used to customize the timestamp of the current log;
- `desc`, used to customize the title of the log;
- `snapshot`, indicating whether it is necessary to take a screenshot of the current screen image and display it in the report:

Examples are as follows:

```
# Incoming string
log("123",desc="this is title 01")


# Pass in non-string
data = {"test": 123, "time": 123456}
log(data,desc="this is title 02")
```

(下页继续)

(续上页)

```
# Incoming traceback
try:
    1/0
except Exception as e:
    log(e, desc="This is title 03")

# Record timestamp and take a screenshot of the current screen
log("123", timestamp=time.time(), desc="This is title 04", snapshot=True)
```

How to set the log save path

Airtest provides some global settings, in which `LOGFILE` is used to customize the name of the txt file that records the log content; `LOGDIR` is used to customize the save path of the log content, examples are as follows:

```
from airtest.core.settings import Settings as ST
from airtest.core.helper import set_logdir

ST.LOG_FILE = "log123.txt"
set_logdir(r'D:\test\1234.air\logs')

auto_setup(__file__)
```

How to filter unnecessary log information

Add **settings for log information level** at the beginning of the script code:

```
__author__ = "Airtest"

import logging
logger = logging.getLogger("airtest")
logger.setLevel(logging.ERROR)
```

After changing the level of output log information to `[ERROR]`, only a small amount of initialization information is output during the entire script running process, which makes it easier to view error messages.

1.8.1.9 Report

Report generation: simple_report()

Simple interface for generating reports:

```
simple_report(filepath, logpath=True, logfile='log.txt', output='log.html')
```

The 4 parameters that can be passed in represent:

- `filepath`, the path of the script file, you can directly pass in the variable '**file**''
- `logpath`, the path where the log content is located, if it is `True`, it will default to the path where the current script is located to find the log content
- `logfile`, the file path of log.txt
- `output`, the path to the report, must end with `.html`

Examples are as follows:

```
from airtest.report.report import simple_report
auto_setup(__file__, logdir=True)

# N use case scripts are omitted here

simple_report(__file__,logpath=True,logfile=r"D:\test\1234.air\log\log.txt",output=r
↪"D:\test\1234.air\log\log1234.html")
```

Report generation: LogToHtml()

Base class of report:

```
class LogToHtml(script_root, log_root=``, static_root='', export_dir=None, script_name='
↪', logfile='log.txt', lang='en', plugins=None)
```

The `logtohtml` class can pass in many parameters:

- `script_root`, script path
- `log_root`, the path of the log file
- `static_root`, the server path where static resources are deployed
- `export_dir`, the storage path of the export report
- `script_name`, the script name
- `logfile`, the path of the log file log.txt
- `lang`, the language of the report (Chinese: zh; English: en)

- `plugins`, plug-ins, will be used if poco or airtest-selenium is used

When using `logtohtml` to generate a test report, we generally first instantiate a `logtohtml` object, and then use this object to call the class method `report()` to generate the report. An example is as follows:

```
from airtest.report.report import LogToHtml

# N use case scripts are omitted here

h1 = LogToHtml(script_root=r'D:\test\1234.air', log_root=r"D:\test\1234.air\log", export_
↪dir=r"D:\test\1234.air" ,logfile= r'D:\test\1234.air\log\log.txt', lang='en', plugins=[
↪"poco.utils.airtest.report"])
h1.report()
```

1.8.1.10 Screenshot

How to take a screenshot with script

Take a screenshot of the target device and save it to a file. You can pass in the file name of the screenshot, a short description of the screenshot, the compression accuracy of the screenshot, and the maximum size of the screenshot. Examples are as follows:

```
snapshot(filename="123.jpg",msg="Homepage screenshot",quality=90,max_size=800)
```

How to take a partial screenshot

Partial screenshots or screenshots by coordinates are a frequently asked question. Airtest provides the `crop_image(img, rect)` method to help us achieve partial screenshots:

```
# -*- encoding=utf8 -*-
__author__ = "AirtestProject"

from airtest.core.api import *
# crop_image() method is in airtest.aircv and needs to be introduced
from airtest.aircv import *

auto_setup(__file__)
screen = G.DEVICE.snapshot()

# Partial screenshot
screen = aircv.crop_image(screen,(0,160,1067,551))
```

(下页继续)

(续上页)

```
# Save partial screenshots to the log folder
try_log_screen(screen)
```

How to do partial image recognition

Steps to find a partial picture:

- Take a partial screenshot
- Define the target screenshot object to find
- Use the `match_in` method to find the specified screenshot object in the partial screenshot

```
from airtest.core.api import *
from airtest.aircv import *
auto_setup(__file__)

screen = G.DEVICE.snapshot()
# Partial screenshot
local_screen = aircv.crop_image(screen,(0,949,1067,1500))

# Set our target screenshot as a Template object
tempalte = Template(r"png_code/settings.png")
# Find the specified image object in the partial screenshot
pos = tempalte.match_in(local_screen)

# Return the coordinates of the image object found (the coordinates are relative to the␣
↪coordinates of the local screenshot)
print(pos)

# To return the coordinates of the target in the entire screen, both x and y need to be␣
↪added with the minimum x and y set during the partial screenshot
print(pos[0]+0,pos[1]+949)
```

How to set the report screenshot accuracy

`SNAPSHOT_QUALITY` is used to set the global screenshot compression accuracy, the default value is 10, and the value range is [1,100]. Examples are as follows:

```
from airtest.core.settings import Settings as ST

```

(下页继续)

(续上页)

```
# Set the global screenshot accuracy to 90
ST.SNAPSHOT_QUALITY = 90
```

Define the compression accuracy of a single screenshot, example:

```
# Set the compression accuracy of a single screenshot to 90, and the remaining unset␣
↪will be based on the global compression accuracy
snapshot(quality=90)
```

How to set the maximum size of report screenshots

In Airtest1.1.6, a new setting for specifying the maximum size of screenshots is added: `ST.IMAGE_MAXSIZE`. If it is set to 1200, the length and width of the last saved screenshot will not exceed 1200. Example:

```
from airtest.core.settings import Settings as ST

# Set the global screenshot size not to exceed 600*600, if not set, the default is the␣
↪original image size
ST.IMAGE_MAXSIZE = 600

# In the case of not setting separately, the value of the global variable in ST is used␣
↪by default, that is, 600*600
snapshot(msg="test12")
# Set the maximum size of a single screenshot not to exceed 1200*1200
snapshot(filename="test2.png", msg="test02", quality=90, max_size=1200)
```

How to specify the path and name for saving screenshots

Take a screenshot of the screen of the current device and save the screenshot in a custom path. This can be achieved in the following ways:

```
screen = G.DEVICE.snapshot()
pil_img = cv2_2_pil(screen)
pil_img.save(r"D:/test/首页.png", quality=99, optimize=True)
```

1.8.1.11 How to make assertions

Assert the existence: assert_exists()

There is an assertion target on the device screen, you need to pass in 1 assertion target (screenshot) and the assertion step information displayed on the report, example:

```
assert_exists(Template(r"tpl1607324047907.png", record_pos=(-0.382, 0.359),␣
↪resolution=(1080, 1920)), "Find the Tmall entrance on the homepage")
```

Assert that does not exist: assert_not_exists()

There is no assertion target on the device screen. Like `assert_exists()`, you need to pass in an assertion target (screenshot) and the assertion step information displayed on the report, for example:

```
assert_not_exists(Template(r"tpl1607325103087.png", record_pos=(-0.005, 0.356),␣
↪resolution=(1080, 1920)), "The icon of Tmall Global does not exist on the current page
↪")
```

Assert equal: assert_equal()

To assert that two values are equal, you need to pass in the values of the two assertions, and a short description of the assertion that will be recorded in the report:

```
assert_equal("actual value", "predicted value", "please fill in a short description of␣
↪the assertion")
```

It is often used to make an assertion together with the script that poco gets the attribute. Examples are as follows:

```
assert_equal(poco("com.taobao.taobao:id/dx_root").get_text(), "Tmall new product", "The␣
↪text attribute value of the control is Tmall new product")

assert_equal(str(poco(text="Tmall new product").attr("enabled")), "True", "The enabled␣
↪attribute value of the control is True")
```

Assert that is not equal: assert_not_equal()

Assert that two values are not equal, like `assert_equal()`, you need to pass in the values of 2 assertions, and a short description of the assertion that will be recorded in the report:

```
assert_not_equal("actual value", "predicted value", "please fill in a short description␣
↪of the assertion")

assert_not_equal("1", "2", "Assert that 1 and 2 are not equal")
```

1.8.1.12 How to switch between absolute and relative coordinates

Use code to switch between absolute coordinates and relative coordinates:

```
# Get device screen resolution (vertical screen)
height = G.DEVICE.display_info['height']
width = G.DEVICE.display_info['width']

# Known absolute coordinates [311,1065], converted to relative coordinates
x1 = 311/width
y1 = 1065/height
poco.click([x1,y1])

# Known relative coordinates [0.3,0.55], convert to absolute coordinates
x2 = 0.3*width
y2 = 0.55*height
touch([x2,y2])

# If it is a horizontal screen device, the resolution is as follows
height = G.DEVICE.display_info['width']
width = G.DEVICE.display_info['height']
```

Determine whether the current screen is horizontal or vertical, and get the resolution of the current screen:

```
if G.DEVICE.display_info['orientation'] in [1,3]:
    height = G.DEVICE.display_info['width']
    width = G.DEVICE.display_info['height']
else:
    height = G.DEVICE.display_info['height']
    width = G.DEVICE.display_info['width']
```

1.8.1.13 How to call other .air scripts

If you want to call a public function encapsulated in another `.air` script in a `.air` script, you can do this:

```
from airtest.core.api import using
# Relative path or absolute path, make sure the code can be found
using("common.air")

from common import common_function
common_function()
```

If the paths of the sub-scripts that need to be referenced are all placed in a certain directory, you can set a default project root directory `PROJECT_ROOT`, so that when using the `using` interface, you can find other sub-scripts in the current root directory without filling in The full path makes it easier to call each other between scripts.

For example, if we create a script named `test1.air`, the actual path is `/User/test/project/test1.air`:

```
from airtest.core.api import *

def test():
    touch("tmp.png")
```

Another `main.air` script in the same directory can refer to the `test` in it like this:

```
from airtest.core.api import *

ST.PROJECT_ROOT = "/User/test/project"
using("test1.air")
from test1 import test
```

1.8.1.14 How to record screen during script running

Currently only supports screen recording on Android devices, please refer to Record the screen while running the script

1.8.2 常见问题与代码示例

English version

1.8.2.1 如何进行脚本初始化

air 脚本：auto_setup()

自动配置运行环境的接口，可以配置当前脚本所在路径、使用的设备、log 内容的保存路径、项目根目录和截图压缩精度：

```
auto_setup(basedir=None, devices=None, logdir=None, project_root=None, compress=None)
```

接口示例：

```
auto_setup(__file__)

auto_setup(__file__, devices=["android://127.0.0.1:5037/emulator-5554?cap_method=JAVACAP&
↪&ori_method=MINICAPORI&&touch_method=MINITOUCH"], logdir=True, project_root=r"D\test",␣
↪compress=90)
```

1.8.2.2 如何连接/使用设备

请注意：更多设备相关的信息，请参考文档

连接设备：connect_device(URI)

连接设备的接口，需要传入用于初始化设备的 URI 字符串，示例：

```
# 连接安卓设备
connect_device("Android://127.0.0.1:5037/SJE5T17B17")

# 连接 iOS 设备
connect_device("iOS:///127.0.0.1:8100")

# 连接 Windows 窗口
connect_device("Windows:///123456")

# 连接模拟器
connect_device("Android://127.0.0.1:5037/127.0.0.1:62001?cap_method=JAVACAP&&ori_
↪method=ADBORI")
```

连接设备：init_device()

初始化设备的接口，需要传入设备平台、设备的 uuid 和可选参数等，其中 uuid 为，Android 的序列号，Windows 的窗口句柄，或 iOS 的 uuid：

```
init_device(platform='Android', uuid=None, **kwargs)
```

接口使用示例：

```
# 连接安卓设备
init_device(platform="Android",uuid="SJE5T17B17",cap_method="JAVACAP")

# 连接 Windows 窗口
init_device(platform="Windows",uuid="123456")
```

获取当前设备：device()

返回当前正在使用中的设备实例，用法示例如下：

```
dev = device()
dev.swipe_along([[959, 418],[1157, 564],[1044, 824],[751, 638],[945, 415]])
```

设置当前设备：set_current()

设置当前的使用设备，可以用于在多设备之间切换使用，示例如下：

```
# 第一种：传入数字 0、1、2 等，切换当前操作的手机到 Airtest 连接的第 1 台、第 2 台手机
set_current(0)
set_current(1)

# 第二种：切换当前操作的手机到序列号为 serialno1、serialno2 的手机
set_current("serialno1")
set_current("serialno2")
```

1.8.2.3 如何进行坐标点击/坐标滑动

坐标点击

在设备上进行点击操作，可以传入点击位置、点击次数等参数，不同平台下的可选参数稍有不同，示例如下：

```
# 传入绝对坐标作为点击位置
touch([100,100])

# 传入 Template 图片实例，点击截图中心位置
touch(Template(r"tpl1606730579419.png", target_pos=5, record_pos=(-0.119, -0.042),␣
↪resolution=(1080, 1920)))

# 点击 2 次
```

(下页继续)

(续上页)

```
touch([100,100],times=2)

# Android 和 Windows 平台下，可以设置点击时长
touch([100,100],duration=2)
```

坐标滑动

在设备上进行滑动操作，有 2 种传参方式，一种是传入滑动的起点和终点，一种是传入滑动的起点和滑动方向 vector，示例如下：

```
# 传入绝对坐标作为滑动的起点和终点
swipe([378, 1460],[408, 892])

# 传入图像作为起点，沿着某个方向滑动
swipe(Template(r"tpl1606814865574.png", record_pos=(-0.322, 0.412), resolution=(1080,␣
↪1920)), vector=[-0.0316, -0.3311])

# 常见的还可以设置滑动的持续时长
swipe([378, 1460],[408, 892],duration=1)
```

1.8.2.4 如何安装/启动/卸载应用

启动应用：start_app()

在设备上启动目标应用，需传入应用的包名，支持 Android 和 iOS 平台，示例：

```
start_app("com.netease.cloudmusic")
```

终止应用运行：stop_app()

在设备上终止目标应用的运行，需传入应用的包名，支持 Android 和 iOS 平台，示例：

```
stop_app("com.netease.cloudmusic")
```

清除应用数据：clear_app()

清理设备上的目标应用数据，需传入应用的包名，**仅支持 Android 平台**，示例：

```
clear_app("com.netease.cloudmusic")
```

安装应用：install()

安装应用到设备上，需传入完整的 apk 的安装路径，**仅支持 Android 平台**，示例：

```
install(r"D:\demo\tutorial-blackjack-release-signed.apk")
```

卸载应用：uninstall()

卸载设备上的应用，需传入被卸载应用的包名，**仅支持 Android 平台**，示例：

```
uninstall("com.netease.cloudmusic")
```

1.8.2.5 按键事件：keyevent

Android 的 keyevent

等同于执行 `adb shell input keyevent KEYNAME` ，示例如下：

```
keyevent("HOME")
# The constant corresponding to the home key is 3
keyevent("3")  # same as keyevent("HOME")
keyevent("BACK")
keyevent("KEYCODE_DEL")
```

Windows 的 keyevent

与安卓不同，Windows 平台使用 `pywinauto.keyboard module` 模块来进行按键输入，示例：

```
keyevent("{DEL}")

# 使用 Alt+F4 关闭 Windows 窗口
keyevent("%{F4}")
```

iOS 的 keyevent

目前仅支持 HOME 键：

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

1.8.2.6 如何输入文本

在设备上输入文本，文本框需要处于激活状态（即先点击文本框，再使用 text() 接口进行输入）。示例如下：

```
touch(文本框的 Template 实例)
text("输入的文本")

# 默认情况下，text 是带回车的，不需要可以传入 False
text("123",enter=False)

# 安卓平台下，还支持输入后点击软键盘的搜索按钮
text("123",enter=False,search=True)
```

1.8.2.7 如何删除文本

通过 keyevent 接口删除单个字符：

```
keyevent("KEYCODE_DEL")
keyevent("67")  # 67 即为删除键，请注意传入的是字符串
```

模拟清空输入框操作：

```
for i in range(10):
    keyevent("67")
```

poco 的删除（输入框置空）：

poco("xxx").set_text("") 。

1.8.2.8 log 相关

如何记录 log 到报告中

log() 接口方便插入用户自定义的一些 log 信息，将会被显示在 Airtest 报告中。在 1.1.6 版本的 Airtest 中，log 接口支持传入 4 个参数：

- args ，可以是字符串、非字符串或者 traceback 对象
- timestamp ，用于自定义当前 log 的时间戳
- desc ，用于自定义 log 的标题

- snapshot ，表示是否需要截取一张当前的屏幕图像并显示到报告中

示例如下：

```
# 传入字符串
log("123",desc="这是标题 01")

# 传入非字符串
data = {"test": 123, "time": 123456}
log(data,desc="这是标题 02")

# 传入 traceback
try:
    1/0
except Exception as e:
    log(e, desc="这是标题 03")

# 记录 timestamp，并且对当前画面截图
log("123", timestamp=time.time(), desc="这是标题 04", snapshot=True)
```

如何设置 log 的保存路径

Airtest 提供了一些全局的设置，其中 LOGFILE 用于自定义记录 log 内容的 txt 文档的名称；LOGDIR 用于自定义 log 内容的保存路径，示例如下：

```
from airtest.core.settings import Settings as ST
from airtest.core.helper import set_logdir

ST.LOG_FILE = "log123.txt"
set_logdir(r'D:\test\1234.air\logs')

auto_setup(__file__)
```

如何过滤不必要的 log 信息

在脚本代码开头加上 **对日志信息等级的设定**：

```
__author__ = "Airtest"

import logging
```

(下页继续)

(续上页)

```
logger = logging.getLogger("airtest")
logger.setLevel(logging.ERROR)
```

把输出日志信息的级别改成 [ERROR] 以后，整个脚本运行过程中只有少量初始化信息输出，更方便查看报错信息。

1.8.2.9 报告

报告生成：simple_report()

生成报告的简单接口：

```
simple_report(filepath, logpath=True, logfile='log.txt', output='log.html')
```

其中可传入的 4 个参数分别表示：

- filepath，脚本文件的路径，可以直接传入变量 __file__
- logpath ， log 内容所在路径，如为 True ，则默认去当前脚本所在路径找 log 内容
- logfile ， log.txt 的文件路径
- output ， 报告的到处路径，必须以 .html 结尾

示例如下：

```
from airtest.report.report import simple_report
auto_setup(__file__, logdir=True)

# 此处省略 N 条用例脚本

simple_report(__file__,logpath=True,logfile=r"D:\test\1234.air\log\log.txt",output=r
↪"D:\test\1234.air\log\log1234.html")
```

报告生成：LogToHtml()

报告的基类：

```
class LogToHtml(script_root, log_root='', static_root='', export_dir=None, script_name='
↪', logfile='log.txt', lang='en', plugins=None)
```

logtohtml 类可以传入的参数非常多：

- script_root ， 脚本路径

- log_root ， log 文件的路径
- static_root ， 部署静态资源的服务器路径
- export_dir ， 导出报告的存放路径
- script_name ， 脚本名称
- logfile ， log 文件 log.txt 的路径
- lang ， 报告的语言（中文：zh；英文：en）
- plugins ， 插件，使用了 poco 或者 airtest-selenium 会用到

使用 logtohtml 生成测试报告时，我们一般先实例化一个 logtohtml 对象，然后用这个对象调用类方法 report() 生成报告，示例如下：

```
from airtest.report.report import LogToHtml

# 此处省略 N 条用例脚本

h1 = LogToHtml(script_root=r'D:\test\1234.air', log_root=r"D:\test\1234.air\log", export_
↪dir=r"D:\test\1234.air" ,logfile=r'D:\test\1234.air\log\log.txt', lang='en', plugins=[
↪"poco.utils.airtest.report"])
h1.report()
```

1.8.2.10 截图相关

如何用脚本截图

对目标设备进行一次截图，并且保存到文件中，可以传入截图文件名、截图的简短描述、截图压缩精度和截图最大尺寸，示例如下：

```
snapshot(filename="123.jpg",msg="首页截图",quality=90,max_size=800)
```

如何进行局部截图

局部截图或者说按坐标截图是大家经常会问到的问题，Airtest 提供了 crop_image(img, rect) 方法可以帮助我们实现局部截图：

```
# -*- encoding=utf8 -*-
__author__ = "AirtestProject"

from airtest.core.api import *
# crop_image() 方法在 airtest.aircv 中，需要引入
```

(下页继续)

(续上页)

```
from airtest.aircv import *

auto_setup(__file__)
screen = G.DEVICE.snapshot()

# 局部截图
screen = aircv.crop_image(screen,(0,160,1067,551))
# 保存局部截图到 log 文件夹中
try_log_screen(screen)
```

如何做局部识图

局部找图的步骤:

- 进行局部截图
- 定义要查找的目标截图对象
- 利用 match_in 方法，在局部截图中查找指定的截图对象

```
from airtest.core.api import *
from airtest.aircv import *
auto_setup(__file__)

screen = G.DEVICE.snapshot()
# 局部截图
local_screen = aircv.crop_image(screen,(0,949,1067,1500))

# 将我们的目标截图设置为一个 Template 对象
tempalte = Template(r"png_code/设置.png")
# 在局部截图里面查找指定的图片对象
pos = tempalte.match_in(local_screen)

# 返回找到的图片对象的坐标（该坐标是相对于局部截图的坐标）
print(pos)

# 若要返回目标在整个屏幕中的坐标，则 x，y 都需要加上局部截图时设置的最小 x、y
print(pos[0]+0,pos[1]+949)
```

如何设置报告的截图精度

SNAPSHOT_QUALITY 用于设置全局的截图压缩精度，默认值为 10，取值范围 [1,100]。示例如下：

```
from airtest.core.settings import Settings as ST

# 设置全局的截图精度为 90
ST.SNAPSHOT_QUALITY = 90
```

定义单张截图的压缩精度，示例：

```
# 设置单张截图的压缩精度为 90，其余未设置的将按照全局压缩精度来
snapshot(quality=90)
```

如何设置报告截图的最大尺寸

在 Airtest1.1.6 中，新增了一个用于指定截图最大尺寸的设置：ST.IMAGE_MAXSIZE 。假如设置为 1200，则最后保存的截图长宽都不会超过 1200，示例：

```
from airtest.core.settings import Settings as ST

# 设置全局截图尺寸不超过 600*600，如果不设置，默认为原图尺寸
ST.IMAGE_MAXSIZE = 600

# 不单独设置的情况下，默认采用 ST 中的全局变量的数值，即 600*600
snapshot(msg="test12")
# 设置单张截图的最大尺寸不超过 1200*1200
snapshot(filename="test2.png", msg="test02", quality=90, max_size=1200)
```

如何指定截图保存的路径和名称

对当前设备的屏幕进行截图，并将截图保存在自定义路径下，可以用下述方式实现：

```
screen = G.DEVICE.snapshot()
pil_img = cv2_2_pil(screen)
pil_img.save(r"D:/test/首页.png", quality=99, optimize=True)
```

1.8.2.11 如何做断言

断言存在：assert_exists()

设备屏幕上存在断言目标，需要传入 1 个断言目标（截图）和在报告上显示的断言步骤信息，示例：

```
assert_exists(Template(r"tpl1607324047907.png", record_pos=(-0.382, 0.359),␣
↪resolution=(1080, 1920)), "找到首页的天猫入口")
```

断言不存在：assert_not_exists()

设备屏幕上不存在断言目标，与 assert_exists() 一样，需要传入 1 个断言目标（截图）和在报告上显示的断言步骤信息，示例：

```
assert_not_exists(Template(r"tpl1607325103087.png", record_pos=(-0.005, 0.356),␣
↪resolution=(1080, 1920)), "当前页不存在天猫国际的 icon")
```

断言相等：assert_equal()

断言两个值相等，需要传入 2 个断言的值，还有将被记录在报告中的断言的简短描述：

```
assert_equal("实际值", "预测值", "请填写断言的简短描述")
```

常与 poco 获取属性的脚本一起做断言，示例如下：

```
assert_equal(poco("com.taobao.taobao:id/dx_root").get_text(), "天猫新品", "控件的 text 属
性值为天猫新品")

assert_equal(str(poco(text="天猫新品").attr("enabled")), "True", "控件的 enabled 属性值为
True")
```

断言不相等：assert_not_equal()

断言两个值不相等，与 assert_equal() 一样，需要传入 2 个断言的值，还有将被记录在报告中的断言的简短描述：

```
assert_not_equal("实际值", "预测值", "请填写断言的简短描述")

assert_not_equal("1", "2", "断言 1 和 2 不相等")
```

1.8.2.12 如何切换绝对坐标和相对坐标

用代码实现绝对坐标和相对坐标之间的切换：

```
# 获取设备屏幕分辨率（竖屏）
height = G.DEVICE.display_info['height']
width = G.DEVICE.display_info['width']

# 已知绝对坐标 [311,1065]，转换成相对坐标
x1 = 311/width
y1 = 1065/height
poco.click([x1,y1])

# 已知相对坐标 [0.3,0.55]，转换成绝对坐标
x2 = 0.3*width
y2 = 0.55*height
touch([x2,y2])

# 如果是横屏设备的话，则分辨率如下
height = G.DEVICE.display_info['width']
width = G.DEVICE.display_info['height']
```

判断当前屏幕为横屏还是竖屏，并获取当前屏幕的分辨率：

```
if G.DEVICE.display_info['orientation'] in [1,3]:
    height = G.DEVICE.display_info['width']
    width = G.DEVICE.display_info['height']
else:
    height = G.DEVICE.display_info['height']
    width = G.DEVICE.display_info['width']
```

1.8.2.13 如何调用别的.air 脚本

如果想要在一个.air 脚本中，调用另外一个 .air 脚本里封装的公用函数，可以这样做：

```
from airtest.core.api import using
# 相对路径或绝对路径，确保代码能够找得到即可
using("common.air")

from common import common_function
common_function()
```

如果需要引用的子脚本路径统一都放在某个目录下，可以通过设定一个默认项目根目录 PROJECT_ROOT ，让使用 using 接口时能够在当前根目录下寻找别的子脚本，无需填写完整路径，让脚本之间相互调用使用更加方便。

例如，我们建立一个名为 test1.air 的脚本，实际路径为 /User/test/project/test1.air :

```
from airtest.core.api import *

def test():
    touch("tmp.png")
```

在同一目录下另外一个 main.air 脚本可以这样引用到它里面的 test :

```
from airtest.core.api import *

ST.PROJECT_ROOT = "/User/test/project"
using("test1.air")
from test1 import test
```

1.8.2.14 如何在脚本运行过程中录制屏幕

目前仅支持 Android 设备的录屏，请参见在运行脚本过程中录屏

Python 模块索引

a

索引

A

B

C

D

E

F

G

H

I

N

O

P

R

S

T

U

V

W

X

Y