# GAutomator使用文档

目录

[GAutomator使用文档 1](#_Toc523928874)

[一、 GAutomator简介 2](#_Toc523928875)

[二、 GAutomator SDK集成 3](#_Toc523928876)

[2.1 基于Unity引擎的游戏 3](#_Toc523928877)

[2.2 基于UE4引擎的游戏 4](#_Toc523928878)

[2.3 基于自研引擎的游戏 4](#_Toc523928879)

[三、 GAutomator测试框架 9](#_Toc523928880)

[3.1 准备工作 9](#_Toc523928881)

[3.2 Getting Started 9](#_Toc523928882)

[3.3 Locating Elements 12](#_Toc523928883)

[3.4 interaction 21](#_Toc523928884)

[3.5 device 25](#_Toc523928885)

[3.6 Reporter 26](#_Toc523928886)

[3.7 实战用例 30](#_Toc523928887)

[3.8 更上一层的封装 35](#_Toc523928888)

1. GAutomator简介

GAutomator是一个针对手游的自动化测试框架。通过使用Python语言编写GAutomator脚本，用户能够获取游戏中的元素信息或者对游戏元素进行控制操作。GAutomator不是一个跨进程的自动化测试框架，被测的游戏需要集成GAutomator SDK（详见第二章），目前GAutomator SDK支持的游戏引擎包括Unity与Unreal Engine 4 ，支持的操作系统为Android。如果游戏基于自研引擎，通过自行实现GAutomator SDK协议，也能够使用GAutomator脚本进行自动化测试。

GAutomator框架如图所示：

Perform Action

Elem Info

Game(UI) Elem Info

Game(UI) Elem Interaction

Native Elem Interaction

Game

SDK

r

ADB

GAutomatorView

Python Scripts

UIAutomator

GAutomator SDK在游戏启动时，会开启一个socket监听程序（27019端口）。PC端的程序通过adb forward建立端口映射，以socket长连接的方式与SDK通信。GAutomatorView是一个游戏元素检索工具，类似于Android SDK工具包中的uiautomatorviewer。

1. GAutomator SDK集成
   1. 基于Unity引擎的游戏

基于Unity引擎的游戏的GAutomator SDK包分为NGUI与UGUI版本。SDK包由libcrashmonitor.so,u3dautomation.jar,U3DAutomation.dll三个文件组成。

下载地址：http://wetest.qq.com/cloud/help/blrooike

用户可按照如下步骤在为Unity游戏集成GAutomator SDK：

**步骤1：导入Unity项目工程：**

a. U3DAutomation.dll中包含一个WeTest.U3DAutomation.U3DAutomationBehaviour组件，放在Assets目录下的任意位置均可。

b. libcrashmonitor.so, u3dautomation.jar需要被打包进游戏,一起放在项目的Assets\Plugins\Android或Builds\Plugin\Android目录下。

**步骤2：初始化WeTest SDK：**

选择第一个Scene,创建一个空的GameObject，然后挂载WeTest.U3DAutomation.U3DAutomationBehaviour组件。

public class WeTestManager : MonoBehaviour {

Application.LogCallback logCall;

void Start () {

#if WETEST\_SDK

this.gameObject.AddComponent<WeTest.U3DAutomation.U3DAutomationBehaviour>();

//this.gameObject.AddComponent<BuglyInitScript>(); //BuglyAgent.RegisterLogCallback(WeTest.U3DAutomation.CrashMonitor.\_OnLogCallbackHandler);

#endif

}

}

备注1：可以通过全局宏定义控制是否集成GAutomator SDK( 由smcs.rsp和gmcs.rsp两个文件控制预编译内容)，如：

def ModifyMacro(workSpacePath, appName, mode):

fsmcs = open(workSpacePath + /Assets/smcs.rsp", 'w')

fgmcs = open(workSpacePath + /Assets/gmcs.rsp", 'w')

if mode == "debug":

fsmcs.write("-define:WETEST\_SDK")

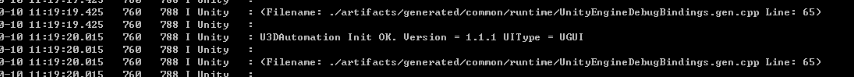
fgmcs.write("-define:WETEST\_SDK")

备注2：如果需要集成第三方或自己实现的异常捕获模块（如Bugly），可以通过将WeTest.U3DAutomation.CrashMonitor.\_OnLogCallbackHandler注册给第三方函数来实现。

**步骤3：检查是否集成成功：**

编译出apk包之后，启动游戏，同时通过adb logcat -vthreadtime -s Unity

查看日志。如果看到U3DAutomation Init OK，则表示SDK已经集成成功。



* 1. 基于UE4引擎的游戏

GAutomator SDK是Unreal 4中的一个项目插件。GAutomator插件目前仅支持Android平台（PIE环境部分功能可用），项目中集成插件之后，可自动启动SDK。SDK支持获取游戏组件、获取位置信息等。

GAutomator UE版本的SDK以源码的形式发布，目前仅支持Android平台，默认运行在Runtime编译状态下，发行版本也会带上。

**注：发布版本关闭GAutomator 插件**  
工程目录下, .uproject只需要把Plugins对应GAutomator设置为不启动即可

"Plugins": [

{

"Name": "GAutomator",

"Enabled": false

}

],

**Step 1：**

将Unreal GAutomator的插件目录拷贝至项目的Plugins目录下。如果游戏项目目录中没有"Plugins"目录，创建"Plugins"目录。插件可拷贝至"Plugins"目录下的任何子目录下。

**Step 2：**

重新生成 c++ 项目文件。插件模块和代码应该在项目文件中被体现出来。

**Step 3：**

正常 编译游戏项目。Unreal Build Tool 会检测到插件的存在，并将它们作为游戏的依赖项进行编译。

**Step 4:**

启动编辑器（或者游戏）。插件一开始是处于禁用的状态，但可以在编辑器界面中启用它。

**Step 5:**

打开插件编辑器（Window->Plugins），找到该插件并选中勾选框。

**Step 6:**

重启编辑器。插件就会自动在启动时被加载。正常编译，即会集成SDK

**集成成功**集成GAutomator插件后，启动的时候会在左上角显示WeTest GAutomator。

GAutomator插件启动成功之后，会在手机上开启27019端口，可通过以下命令查看

adb shell netstat -an|findstr 27019

* 1. 基于自研引擎的游戏

对于基于自研引擎的游戏，我们无法直接提供GAutomator SDK，用户需要自己实现SDK的协议来支持GAutomator自动化测试。

GAutomator SDK基本协议格式如下：

Body

32bit

Json:{“cmd”: 1, “value”: “”}

Head

Head: 32bit的int类型，表示body的length。

Body：json字符串。PC客户端（脚本）向手机端服务发送命令格式为{“cmd”: number, ”value”: “what you want}，手机端服务向PC的回包格式为{“cmd”:number, “status”: code,”data”: “result value”}。status为0表示正确。Status为其他数字，data中的内容表示错误原因。

具体的，游戏端至少需要实现如下几个服务：

1. getSdkVersion:

功能：获取当前SDK版本信息(cmd:100)

请求json：

{

"cmd": 100,

"value": ""

}

回包json：

{

"cmd": 300,

"status": 0,

"data": {

"engine": "Cocos2D",

"sdkVersion": "1.5.0",

"sdkUIType": "cocos",

"engineVersion": "5.1.2f1"

}

}

1. dumpTree

功能：dump当前界面的Node树结构，以xml形式返回

请求json：

{

"cmd": 100,

"value": ""

}

回包json：

{

"cmd": 300,

"status": 0,

"data": "<AbstractRoot engine='cocos'><GameObject name='WetestSDK'/></AbstractRoot>"

}

Xml的具体格式：

<AbstractRoot engine="cocos">

<GameObject name="WetestSDK"/>

</AbstractRoot>

根节点约定为AbstractRoot，属性engine约定为"cocos"。其他Node节点，xml节点名称为GameObject(为了兼容Unity)，name字段为Node的名称。

1. findElements

功能：根据name查找当前界面是否存在该Node（可同时查找多个）。

请求json：

{

"cmd": 101,

"value": [

"Sample",

"m\_Btn"

]

}

回包json：

{

"cmd": 101,

"status": 0,

"data": [

{

"instance": -1,

"name": "Sample"

},

{

"instance": 12312312,

"name": "m\_Btn"

}

]

}

instance：若Node存在可以用地址来表示。若不存在，则可以用-1来表示。

1. getElementBounds

功能：获取Node的屏幕坐标及长宽

请求json：

{

"cmd": 103,

"value": [

"Sample",

"m\_Btn"

]

}

回包json:

{

"cmd": 103,

"status": 0,

"data": [

{

"existed": true,

"width": 0,

"visible": false,

"height": 0,

"path": "Sample",

"y": 0,

"x": 0

},

{

"existed": true,

"width": 250,

"visible": true,

"height": 80,

"path": "m\_btn",

"y": 256,

"x": 175

}

]

}

1. getElementByPos

功能：根据屏幕坐标，找到该位置下对应的Node节点。

请求json:

{

"cmd": 301,

"value": [

100,

499

]

}

其中value为数组，value[0]表示x坐标，value[1]表示y坐标。x,y为屏幕坐标系，左上角为(0,0)坐标。

回包json：

a.若能够找到对应的Node：

{

"cmd": 301,

"status": 0,

"data": {"name":"m\_btn","instance":123123}

}

b.若不能找到对应Node

{

"cmd": 301,

"status": 4,

"data": ""

}

1. GAutomator测试框架

3.1 准备工作

环境：python 2.7/3.4+, adb （已加入PATH环境变量）

建议使用Pycharm作为Python IDE.

GAutomator测试框架(Python工程)。

GAutomatorView：一个基于Java的GameObject检索工具。连接Android手机到PC,启动集成SDK的游戏，点击“同步游戏”获取页面元素信息。

如果您的自动化测试内容仅包括简单的查找与点击元素，可直接使用3.8节的接口。3.3~2.5节的内容可帮助您更好的理解GAutomator测试框架以及使用更多的功能。

3.2 Getting Started

Sample Code: sample/Unity/sample.py

Sample Apk: sample/Unity/wetest\_demo\_unity.apk

本节使用的sample是基于Unity引擎的游戏。UE4的demo在sample/UE4下。

3.2.1 Simple Usage

可以将下面的代码保存到GAutomator框架Python工程中，如sample.py

<a name="lib"></a>

#import lib path,only use in this demo

#import sys,os

#sys.path.append(os.path.abspath(os.path.join(os.getcwd(), "..\\")))

import wpyscripts.manager as manager

def test():

engine=manager.get\_engine()

logger=manager.get\_logger()

version=engine.get\_sdk\_version()

logger.debug("Version Information : {0}".format(version))

scene=engine.get\_scene()

logger.debug("Scene : {0}".format(scene))

sample\_button=engine.find\_element("/Canvas/Panel/Sample")

logger.debug("Button : {0}".format(sample\_button))

engine.click(sample\_button)

test()

3.2.2 实例讲解

wpyscripts.manager模块提供了自动化测试所需的所有功能，提供与引擎、设备、报告、日志相关的内容。

import wpyscripts.manager as manager

下一步，创建Engine和日志实例。

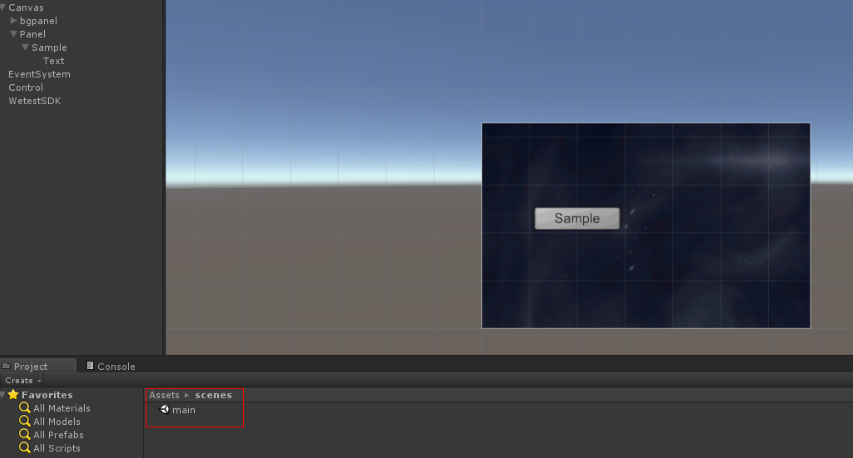
engine=manager.get\_engine()

logger=manager.get\_logger()

engine.get\_sdk\_version()能够获取引擎版本、GAutomator SDK版本信息。如果能够获取该信息，证明脚本已经成功连上游戏中的SDK服务。如果获取失败，则会抛出WeTestNativeEngineDllError异常,抛出该异常可能是手机USB线没有连好或者手机开发者选项未打开。

通过*logger.debug("")*输出对应日志，请使用manager.get\_logger()获取的实例，避免脚本在云端[wetest.qq.com](http://wetest.qq.com)执行时出错。

engine.get\_scene()获取当前游戏界面对应scene名称。wetestdemo游戏的第一个scene名称为main。



engine.find\_element("/Canvas/Panel/Sample")查找当前界面中路径为/Canvas/Panel/Sample的节点，如果存在则返回Element，不存在则返回None。find\_element直接使用Unity [GameObject.Find](http://docs.unity3d.com/ScriptReference/GameObject.Find.html?from=index)查找当前游戏中的game object。

查找到的节点samle\_button（Element），有两个属性object\_name,instance。object\_name是节点的全路径，instance是节点实例的编号（GameObject.GetInstanceID()获取）instance在当前游戏中一定是唯一的。

engine.click(sample\_button)尝试点击samle\_button这个GameObject的中心点。

GAutomator测试框架包含3大接口：

engine=manager.get\_engine()

reporter=manager.get\_reporter()

device=manager.get\_devcie()

engine: 引擎相关内容，主要包括控件获取，游戏操作。Unity与UE4 engine的接口不一致，具体详见文档。GAutomator根据配置，选择对应的实现，Unity引擎实现为UnityEngine类，UE4引擎为UnRealEngine类。自研引擎可根据服务端实现的协议来构建Engine。

reporter: 云端报告相关。功能包括截图、标记操作过程、性能数据打标签。

device: 手机设备相关，如屏幕长宽高、转向，也包括QQ登录等。

3.2.3 WeTest云端兼容性测试

GAutomator编写好的测试脚本，只需要非常简单的修改，就能WeTest云端做兼容测试。云端成千上百台手机，执行脚本对游戏进行兼容性测试。WeTest能够发现兼容问题，同时高度还原执行现场，包括手机日志、崩溃信息、截图、执行过程等。

云端执行脚本时，会执行testcase.runner下的run函数，只需要把自己的业务逻辑加入到这个函数中即可。

import traceback

try:

from sample.sample import \*

except Exception,e:

traceback.print\_exc()

def run():

"""

Begin of the test logic

"""

try:

test()

except Exception,e:

traceback.print\_exc()

然后，运行python build.py。会产生一个压缩包wpyscripts\_upload.zip用于上传WeTest 平台测试。只有企业用户才可以在云端测试，请登录wetest.qq.com，联系工作人员了解详情。

3.2.4 本地运行

注：调试时手动启动游戏，运行到指定界面，运行对应的脚本即可，如调试大厅界面的代码，游戏跑到大厅界面，再运行自动化测试逻辑。不需要从main.py启动。

GAutomator支持一台PC在多台android手机上同时测试。在config.py文件中，可以配置，完成单台手机测试的情况。

测试的游戏包名：

class TestInfo(object):

PACKAGE="com.tencent.wetest.demo" # the packagename of test game

测试账号:

class Account(object):

QQNAME="" #QQ acccount

QQPWD="" #QQ password

WECHATNAME="" #wechat account

WECHATPWD="" #wechat password

引擎选择（默认为Unity):

<a name="Engine"></a>

### Engine Type

Unity="unity"

UE4="ue4"

class Engine(object):

Type=Unity #Type="unity" # unity or ue4

1、测试一台手机，如果PC上USB只连接一台手机，直接启动main.py即可(前提配置好测试包名)

python main.py

2、测试多台手机，如果PC上USB连接超过一台手机，需要通过命令行的方式启动。

adb devices #查看当前手机序列号

saaaweadf device

asdfadfadf device

获取到当前PC连接的手机序列号之后，通过命令行的方式控制脚本在指定的手机上进行测试。

python main.py --qqname=2952020110 --qqpwd=wetestpwd --engineport=50031 --uiport=19000 --serial=saaaweadf

python main.py --qqname=2952020111 --qqpwd=wetestpwd --engineport=50032 --uiport=19001 --serial=asdfadfadf

上面的命令分别代表:在序列号"saaaweadf"手机上测试，测试时使用的QQ号为2952020110,密码为wetestpwd，与引擎建立映射的网络端口号为50031,与UIAutomator服务建立映射的网络端口为19000。第二条命令类似。

命令行参数含义如下：

--qqname:qq账号，每部手机应该都不一样

--qqpwd:qq密码

--wechataccount:微信账号

--wechatpwd:微信密码

--othername:其他任何账号

--otherpwd:其他任何账号的密码

--engineport:与手机端的sdk服务建立网络映射，填入的为本地的网络端口号（如,50031），不同手机之间要确保不同

--uiport:与手机端的UIAutomator服务建立网络映射，填入的为本地的网络端口号（如,19008），不同手机之间要确保不同

--serial:adb devcies能够查看手机的序列号，不同的序列号代表不同的手机

3.3 Locating Elements

Unity的engine模块提供了三种GameObject的查找方式(find\_element、find\_elements\_path、find\_elements\_by\_component)

demo：sample/Unity/find\_elements.py

UE4的engine模块一种查找方式(*find\_element)*，目前只支持UMG控件的查找和操作。

demo：sample/UE4/find\_elements.py

自研引擎可根据自己在服务端实现的协议确定查找方式的种类。

3.3.1 find\_element

Unity版本中，find\_element通过Unity的GameObject.Find()方法查找游戏中的gameobject。find\_element通过GameObject的名称查找对象，名字中可以包含'/'代表GameObject树中的一层。这方法只返回当前激活(active)的gameobject。当界面上有两个相同路径的gameobject时，只返回其中的一个。Unity游戏的demo：

#import sys,os,time

#sys.path.append(os.path.abspath(os.path.join(os.getcwd(), "..\\")))

import wpyscripts.manager as manager

def test\_find\_element():

button=engine.find\_element("/Canvas/Panel/Button")

bound=engine.get\_element\_bound(button)

logger.debug("Button : {0},Bound : {1}".format(button,bound))

engine.click(button)

button=engine.find\_element("Button")

bound=engine.get\_element\_bound(button)

logger.debug("Button : {0},Bound : {1}".format(button,bound))

engine.click(button)

button=engine.find\_element("Panel/Button")

bound=engine.get\_element\_bound(button)

logger.debug("Button : {0},Bound : {1}".format(button,bound))

engine.click(button)

unexited\_gameobj=engine.find\_element("Test")

if unexited\_gameobj is None:

logger.debug("Test GameObject not find")

test\_find\_element()

上面的代码可以保存为find\_elments.py,从wetestdemo点击FindElements，然后运行：

python find\_elments.py



<GameObject name="WetestSDK" components="U3DAutomationBehaviour" id="10064" />

<GameObject name="Control" components="gobal" id="10074" />

<GameObject name="EventSystem" components="EventSystem|StandaloneInputModule|TouchInputModule" id="10196" />

<GameObject name="Canvas" components="Canvas|CanvasScaler|GraphicRaycaster|AudioListener" id="10188">

<GameObject name="bg" components="CanvasRenderer|Image" id="10198" img="Background">

<GameObject name="RawImage" components="CanvasRenderer|RawImage" id="10194" img="find\_bg" />

</GameObject>

<GameObject name="Panel" components="CanvasRenderer|FindElementsControl" id="10192">

<GameObject name="Button" components="CanvasRenderer|Image|Button|EventTriggerListener" id="10182" img="bt\_bg">

<GameObject name="Text" components="CanvasRenderer|Text" id="10186" txt="Button" />

</GameObject>

<GameObject name="Back" components="CanvasRenderer|Image|Back|EventTriggerListener" id="10190" img="back" />

<GameObject name="Button" components="CanvasRenderer|Image|Button" id="10200" img="bt\_bg">

<GameObject name="Text" components="CanvasRenderer|Text" id="10184" txt="Button" />

</GameObject>

</GameObject>

</GameObject>

Unity版本demo运行时符合"/Canvas/Panel/Button"的节点有两个，但是每次都是节点的第一个。如果查找的节点不存在，则返回None。

UE4游戏的demo：

#import sys,os,time

#sys.path.append(os.path.abspath(os.path.join(os.getcwd(), "..","..")))

def test\_find\_element():

button = engine.find\_element("Button\_0")

bound = engine.get\_element\_bound(button)

logger.debug("Button : {0},Bound : {1}".format(button, bound))

engine.click(button)

button = engine.find\_element("Button\_1")

bound = engine.get\_element\_bound(button)

logger.debug("Button : {0},Bound : {1}".format(button, bound))

engine.click(button)

unexited\_gameobj = engine.find\_element("Test")

if unexited\_gameobj is None:

logger.debug("Test GameObject not find")

test\_find\_element()

上面的代码可以保存为find\_elments.py,从wetest\_demo\_ue点击FindElements，然后运行：

python find\_elments.py

UE4的UMG中UI控件的名称一定是唯一的。如果查找的控件存在则返回Element，如果不存在则返回None

3.3.2 find\_element\_path( Unity Only)

*find\_elements\_path*能够一次查找到多个符合的gameobject。但它是一个**非常耗时的操作需要谨慎使用**，对测试时的性能数据有一定影响（主要是fps值）。*find\_elments*通过表达式查找gameobject,查找条件为（查找条件为与，只要出现就一定要满足）：gameobject路径、gameobject在节点中的位置，顺序(第一个节点为0)、gameobject包含图片组件时的图片名称、gameobject包含文字组件时的文字内容。  
***注：不能确保返回结果的顺序***

3.3.2.1 名称查找

def test\_find\_elements\_by\_name():

elements = engine.find\_elements\_path("/Canvas/Panel/VerticalPanel/Item(Clone)")

for element in elements:

bound=engine.get\_element\_bound(element)

logger.debug("Button : {0},Bound : {1}".format(element,bound))

engine.click(bound)

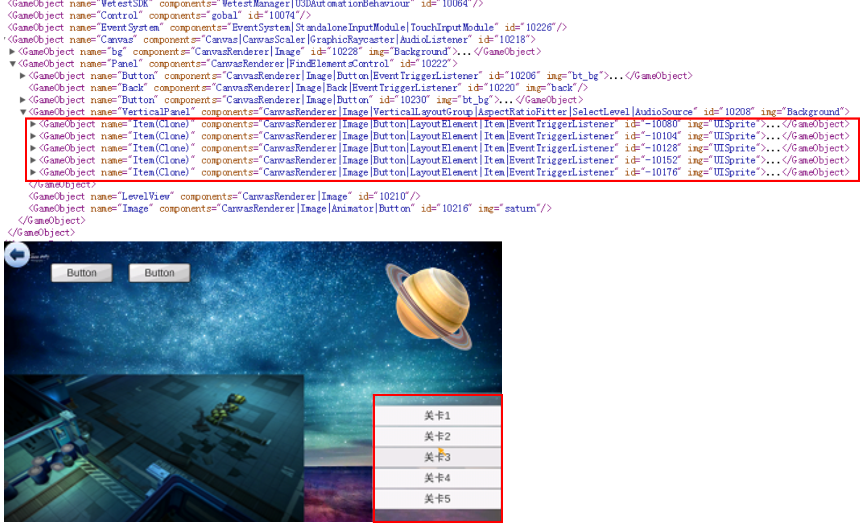
time.sleep(0.5)

test\_find\_elements\_by\_name()

上面的代码可以保存为find\_elments.py,从wetestdemo点击FindElements，然后运行：

python find\_elments.py

*find\_elements\_path*能够返回所有符合的节点。



返回结果：

GameObject /Canvas/Panel/VerticalPanel/Item(Clone) Instance = -10080

GameObject /Canvas/Panel/VerticalPanel/Item(Clone) Instance = -10104

GameObject /Canvas/Panel/VerticalPanel/Item(Clone) Instance = -10128

GameObject /Canvas/Panel/VerticalPanel/Item(Clone) Instance = -10152

GameObject /Canvas/Panel/VerticalPanel/Item(Clone) Instance = -10176

关卡列表，除了/Canvas/Panel/VerticalPanel/Item(Clone)符合之外，也可以是其他表达式。

\_elements = engine.find\_elements\_path("Panel/VerticalPanel/Item(Clone)")

\_elements = engine.find\_elements\_path("VerticalPanel/Item(Clone)")

\_elements = engine.find\_elements\_path("Item(Clone)")

\_elements = engine.find\_elements\_path("/Canvas/Panel/\*/Item(Clone)")

* Panel/VerticalPanel/Item(Clone)，表示查找节点Item(Clone)，且父亲节点为VerticalPanel，祖父节点Panel
* VerticalPanel/Item(Clone)，表示查找节点Item(Clone)，且父亲节点为VerticalPanel
* Item(Clone)，表示查找所有名叫Item(Clone)的节点
* /Canvas/Panel/\*/Item(Clone)，表示查找节点Item(Clone)，任意父亲节点，祖父节点为Panel,曾祖父节点为Canvas且为根节点。其中\*表示任意名称。

***find\_elments\_path***接口非常耗时。

3.3.2.2 子节点序列查找

如果只想选择关卡2，不想返回所有的节点。可以利用*find\_elements\_path*中的序列来进行查找定位。序列的表达式为[num],num为数字从0开始。

def test\_find\_elements\_by\_index():

elements = engine.find\_elements\_path("/Canvas/Panel/VerticalPanel/\*[1]")

for element in elements:

bound=engine.get\_element\_bound(element)

logger.debug("Button : {0},Bound : {1}".format(element,bound))

engine.click(bound)

time.sleep(0.5)

elements=engine.find\_elements\_path("/Canvas/Panel/VerticalPanel/Button[0]")

assert elements == []

test\_find\_elements\_by\_index()

上面的代码可以保存为find\_elments.py,从wetestdemo点击FindElements，然后运行：

python find\_elments.py

*find\_elements\_path*能够返回所有符合的节点,这个脚本中会返回关卡2，并点击。

* /Canvas/Panel/VerticalPanel/\*[1]，表示查找VericalPanel节点中的第二个子节点，VericalPanel的父节点为Panel,Panel的父节点为Cavnvas，且Canvas为根节点
* /Canvas/Panel/VerticalPanel/Button[0]，表示查找VericalPanel节点中第一个子节点，**且名为Button**,VericalPanel的父节点为Panel,Panel的父节点为Cavnvas，且Canvas为根节点  
  **注：序列从0开始，0表示第一个子节点；序列[num]与名字的关系是与，需要都符合，如果是任意名称请用\***。根节点不能使用[]，[]是相对于父亲节点的位置，所以根节点不存在父节点。

3.3.2.3 节点图片名称查找

*find\_elements\_path*能够根据节点的图片名称进行查找。表达式为{img=imageName}，img为图片名称。

def test\_find\_elements\_by\_img():

elements = engine.find\_elements\_path("/Canvas/Panel/Image{img=saturn}")

for element in elements:

bound=engine.get\_element\_bound(element)

logger.debug("Button : {0},Bound : {1}".format(element,bound))

engine.click(bound)

time.sleep(0.5)

engine.click\_position(100,200)

elements = engine.find\_elements\_path("/Canvas/Panel{img=saturn}")

for element in elements:

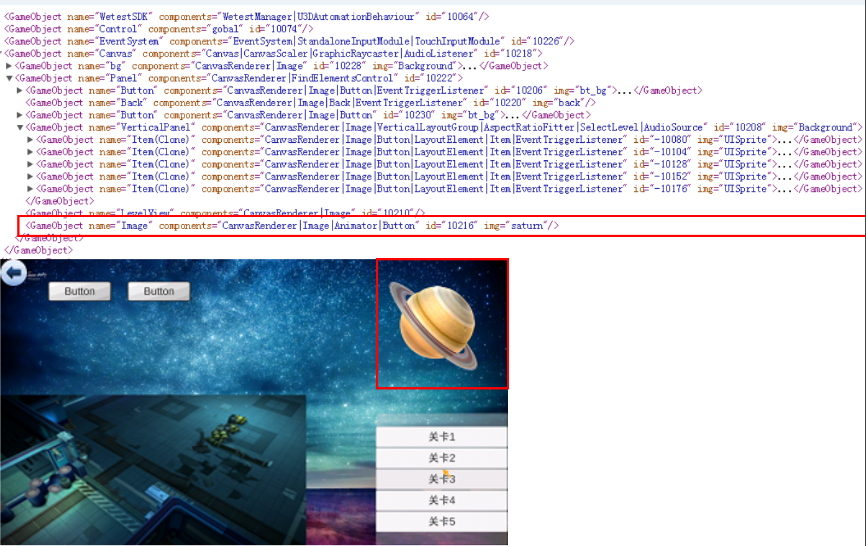
bound=engine.get\_element\_bound(element)

logger.debug("Button : {0},Bound : {1}".format(element,bound))

test\_find\_elements\_by\_img()

上面的代码可以保存为find\_elments.py,从wetestdemo点击FindElements，然后运行：

python find\_elments.py



运行结果如下：

Button : GameObject /Canvas/Panel/Image Instance = 10218,Bound : point(1461.0,81.0) width = 352.0 height = 341.0

Button : GameObject /Canvas/Panel Instance = 10222,Bound : point(0.0,0.0) width = 1920.0 height = 1080.5

/Canvas/Panel/Image{img=saturn},表示查找Image节点（且Image节点或者子节点包含saturn图片），Image父节点为Panel,Panel父节点为Canvas,且Canvas为根节点。

/Canvas/Panel{img=saturn},表示查找Panel节点（且Panel节点或者子节点包含saturn图片），Panel父节点为Canvas,且Canvas为根节点。

img代表的是图片名称，Unity游戏中哪些组件符合这边的名称呢？

对UGUI, wetest sdk会搜索Image、RawImage、SpriteRender中的图片。

对NGUI，wetest sdk会搜索UISprite、UITexture、Renderer中的图片。

为什么要搜索节点及其所有子节点？Unity制作的时候，往往会在可交互节点下面挂载图片文字等。这样做的目的是为了尽可能测试人员方便查找。

3.3.2.4 节点文字查找

*find\_elements\_path*能够根据节点及子节点中文字内容进行查找。表达式为{txt=txtName}，txtName为文字内容

def test\_find\_elements\_by\_txt():

elements=engine.find\_elements\_path("Panel/VerticalPanel/Item(Clone){txt=关卡2}")

for element in elements:

bound=engine.get\_element\_bound(element)

logger.debug("Button : {0},Bound : {1}".format(element,bound))

engine.click(bound)

time.sleep(0.5)

elements=engine.find\_elements\_path("Panel/VerticalPanel/Item(Clone){txt=关卡4}")

if len(elements) > 0:

engine.click(elements[0])

test\_find\_elements\_by\_txt()

上面的代码可以保存为find\_elments.py,从wetestdemo点击FindElements，然后运行：

python find\_elments.py



运行结果如下：

Button : GameObject /Canvas/Panel/VerticalPanel/Item(Clone) Instance = -11784,Bound : point(1420.0,710.5) width = 500.0 height = 80.0

Button : GameObject /Canvas/Panel/VerticalPanel/Item(Clone) Instance = -11832,Bound : point(1420.0,870.5) width = 500.0 height = 80.0

Panel/VerticalPanel/Item(Clone){txt=关卡2},表示查找Item(Clone)节点（且Item(Clone)节点或者其子节点，包含文字"关卡2"），Item(Clone)父节点为Panel,Panel父节点为Canvas,且Canvas为根节点

Panel/VerticalPanel/Item(Clone){txt=关卡4},表示查找Item(Clone)节点（且Item(Clone)节点或者其子节点，包含文字"关卡4"），Item(Clone)父节点为Panel,Panel父节点为Canvas,且Canvas为根节点。

txt代表的是文字内容，寻找匹配时，会从以下节点查找问题内容：

对UGUI，wetest sdk会搜索组件Text、GUIText中的内容。

对NGUI，wetest sdk会搜索组件UILabel、UIInput和GUIText中的内容。

3.3.3 find\_elements\_by\_component（Unity Only）

根据Unity中Component名称查找Gameobject。本质上调用的是Unity中的GameObject.FindObjectsOfType(Type.GetType(name))接口。C#小红花Type.GetType传入的应该是AssemblyQualifiedName。所以下面的例子中传入的是"UnityEngine.UI.Button,UnityEngine.UI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"，如果传入的是Button返回为空。

def test\_find\_elements\_by\_component():

elements=engine.find\_elements\_by\_component("UnityEngine.UI.Button,UnityEngine.UI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null")

for element in elements:

bound = engine.get\_element\_bound(element)

logger.debug("Button : {0},Bound : {1}".format(element, bound))

engine.click(bound)

time.sleep(0.5)

test\_find\_elements\_by\_component()

3.3.4 查找节点位置

3.3.4.1 获取屏幕坐标

engine.get\_element\_bound(element)能够获取节点在屏幕中的位置。GAutomator所有的操作都是通过触屏进行的，因此获取节点在屏幕上的位置是进行交互操作的基石。

def test\_click():

# click the element

element = engine.find\_element("ClickBtn")

bound = engine.get\_element\_bound(element)

logger.debug("Button : {0},Bound : {1}".format(element, bound))

engine.click(bound)

time.sleep(1)

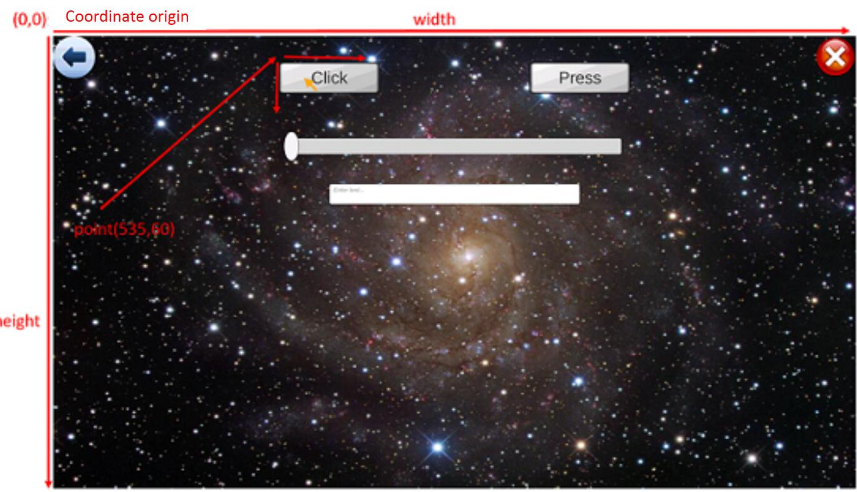
engine.click(element)

time.sleep(2)

engine.click\_position(600.0, 100.0)

test\_click()

engine.get\_element\_bound(Element)获取的是ElementBound，获取Element的左上角在屏幕上的坐标，和Element的长宽高。遵循的是手机的坐标系，以左上角为坐标原点，上下边框的长度为width，左右边框的长度为height。



3.3.4.2 获取世界坐标（Unity Only）

手游越来越重度化，3D手游越来越普及，对于3D手游而言仅仅是屏幕坐标已经无法满足自动化测试的需求。对于王者荣耀、全民超神这类手游，自动化测试过程中需要知道自己、敌方和队友英雄在地图上的位置，才能编写出想要的自动化功能（如移动英雄到某个位置、发现敌方英雄攻击等）。所以在wpyscripts v 1.1.1版本，GAutomator SDK 8版本，推出了get\_element\_world\_bound(elements)，能够获取节点的世界坐标系。

示例：sample/Unity/joystick\_tester.py 中的test\_world\_bounds()

def test\_world\_bounds():

person=engine.find\_element("/3rd Person Controller/Bip001/Bip001 Pelvis")

world\_bound=engine.get\_element\_world\_bound(person)

logger.debug(world\_bound[0])

test\_world\_bounds()

以上代码，需要在wetest\_demo\_unity.apk点击Joystick，进入Joystick界面后运行才有效。

python joystick\_tester.py

结果如下：

center = (5.03773808305e-05,0.1374322474,0.00151373702101) extents =(0.0807622969151,0.09486310184,0.0181320905685)

返回查询的所有Element的对应世界坐标，WorldBound[]。WorldBound是节点在Unity世界坐标系中的各项值，主要包括中心点的x,y,z坐标值，及中心点距物体在x,y,z轴方向上的距离。具体可参考：Unity官网Bounds介绍<http://docs.unity3d.com/ScriptReference/Bounds.html>

3.4 interaction

找到节点后，就需要对其进行操作。demo：sample/Unity/interaction.py与sample/UE4/interaction.py。

3.4.1 click

Engine执行操作后，会立马返回，不会等button按钮响应完成才返回。engine.click(Element)返回为True的话，只保证执行了button中心点的点击事件，不能确保button对应的事件被有效执行（有弹出框，遮住的情况就可能使点击无效）。

*engine.click()*允许传入Element和ElementBound。如果传入的是Element，会先去查找ElementBound,然后再计算出节点的中心位置进行点击。所以，在有ElementBound的情况下，应该首先传入ElementBound。

def test\_click():

#点击节点

element = engine.find\_element("ClickBtn")

bound=engine.get\_element\_bound(element)

logger.debug("Button : {0},Bound : {1}".format(element,bound))

engine.click(bound)

time.sleep(1)

engine.click(element)

time.sleep(2)

engine.click\_position(600.0,100.0)

test\_click()

上面的代码可以保存为interaction.py,从wetestdemo点击Interaction，然后运行：

python interaction.py

程序会连续点击3下Click按钮。

engine.click(bound)会点击Click的中心节点(point.x+withd/2,point.y+height/2)。

engine.click(element)首先回去查找element节点的ElementBound，然后计算出中心点，在进行点击。

engine.click\_position(600.0,100.0)直接点击屏幕坐标为(600.0,100.0)的坐标。手机屏幕尺寸发生变化，点击将无效，不能点击到期望位置。

3.4.2 long press

engine.press()和engine.press\_position与click相似，多一个时间参数，表示长按的时间（单位ms,毫秒）

def test\_press():

element=engine.find\_element("PressBtn")

engine.press(element,5000)

time.sleep(2)

engine.press\_position(1200,100,3000)

test\_press()

上面的代码可以保存为interaction.py,从wetestdemo点击Interaction，然后运行:

python interaction.py

engine.press(element,5000),PressBtn节点连续按住5s

engine.press\_position(1200,100,3000),(1200,100)点，连续按住3s

3.4.3 swipe

engine.swipe(start\_element, end\_element, steps, duration=1000)

engine.swipe\_position(start\_x,start\_y,end\_x,end\_y,steps, duration=1000)

它们可以从一个节点滑动到另外一个节点，通过设置滑动步骤来控制滑动的平滑度和滑动速度。duration以毫秒为单位，为滑动的时长。滑动时长不能不能精确控制，只是尽可能接近。滑动由Touch触屏操作的Down->move->move...->up组合而成，steps指的是move的数量，一般指的是滑动的平滑度。swipe与swipe\_position动作执行完之后返回，由SDK负责执行动作，不能并行的执行动作。如下面的示例中，第一个动作执行完后，才会执行第二个动作。

def test\_swipe():

start\_e=engine.find\_element("/Canvas/Panel/Press")

end\_e=engine.find\_element("/Canvas/Panel/Click")

engine.swipe(start\_e,end\_e,50,2000)

silder=engine.find\_element("/Canvas/Panel/Slider")

if silder:

bound=engine.get\_element\_bound(silder)

engine.swipe\_position(bound.x,bound.y+bound.height/2.0,bound.x+bound.width,bound.y+bound.height/2,100,3000)

test\_swipe()

上面的代码可以保存为interaction.py,从wetestdemo点击Interaction，然后运行:

python interaction.py

从Press的中心点按钮按下，一直Move到Click的中心点，中间经过50步，最后执行Up动作，持续时长大致为2秒.

start\_e=engine.find\_element("/Canvas/Panel/Press")

end\_e=engine.find\_element("/Canvas/Panel/Click")

engine.swipe(start\_e,end\_e,50,2000)

无论swipe的步长设置为多少，都会立刻返回。立刻执行swipe\_position函数，swipe\_position也需要动作执行完之后才会返回，但是游戏中还不会马上执行这个动作。需要swipe执行完成后，才会执行swipe\_position的动作。

silder=engine.find\_element("/Canvas/Panel/Slider")

if silder:

bound=engine.get\_element\_bound(silder)

engine.swipe\_position(bound.x,bound.y+bound.height/2.0,bound.x+bound.width,bound.y+bound.height/2,100,3000)

3.4.4 input(Unity Only)

engine.input(Element,txt)设置Element里面的文字内容

def test\_input():

element=engine.find\_element("/Canvas/Panel/InputField")

engine.input(element,"Run Wpy")

test\_input()

上面的代码可以保存为interaction.py,从wetestdemo点击Interaction，然后运行:

python interaction.py

运行后，文本框里面的内容从*Hello wpyscripts*变成*Run Wpy*。

UGUI的Element必须包含InputField组件；NGU的Element必须包含UILabel组件

3.4.5 get\_touchable\_elements（Unity Only）

所有的交互操作，只能保证屏幕上产生了这些事件。例如下图，点击Click按钮，只确保在屏幕Click按钮的位置按了一下，不确保Click按钮有效果，因为这个时候有弹出框遮住了Click按钮。



游戏运行过程中，因为等级、公告、网络等各种原因可能会出现弹出框，这个时候原本的测试逻辑将无法继续运行。engine.get\_touchable\_elements()可以返回当前可点击的节点。

def test\_get\_touchable\_elements():

e=engine.find\_element("/Canvas/Panel/Close")

engine.click(e)

elements=engine.get\_touchable\_elements()

for e,pos in elements:

logger.debug("Button : {0},Bound : {1}".format(e,pos))

time.sleep(2)

engine.click\_position(elements[0][1]["x"],elements[0][1]["y"])

test\_get\_touchable\_elements()

上面的代码可以保存为interaction.py,从wetestdemo点击Interaction，然后运行：

python interaction.py

运行后，返回当前可点击的有效节点及他们的位置：

Button : GameObject /Canvas/Dialog(Clone)/Sure Instance = 4294957156,Bound : {'y': 733.0, 'x': 660.0}

Button : GameObject /Canvas/Dialog(Clone)/Cancel Instance = 4294957136,Bound : {'y': 733.0, 'x': 1260.0}

例子中返回的就是，Confirm和Canel两个按钮。这两个按钮是有效的可交互的按钮，其他红色圈出来的回退、Click、Press和Close都已经被遮住，在这个位置点击也是无效的。  
engine.get\_touchable\_elements()是一个相对耗时的接口，常见的游戏耗时在80ms以内（在一帧内处理）。接口返回的是一个元组列表，元组中包含可交互的有效节点和位置（一个x,y的字典），所以无需再请求节点位置，可以直接点击。

3.4.6 get\_element\_text

可以获取到游戏中的文字内容。NGUI能够获取到UILable、UIInput、GUIText组件上的文字内容，如果GameObject上不包含以上组件，将抛出异常。UGUI能够获取Text、GUIText组件上的文字信息。示例在interaction.py中，demo游戏需要在interaction界面。

def test\_get\_element\_txt():

e=engine.find\_element("Click/Text")

text=engine.get\_element\_text(e)

logger.debug("Text = {0}".format(text))

3.4.7 get\_element\_image(Unity Only)

可以获取到游戏中的GameObject上面对应的图片名称。NGUI取UITexture、UISprite、SpriteRenderer组件上的图片名称，如果GameObject不包含以上组件，将抛出异常。UGUI能够获取Image、RawImage、SpriteRenderer组件上的图片名称。示例在interaction.py中，wetest\_demo\_unity游戏需要在interaction界面。

def test\_get\_element\_image():

e = engine.find\_element("Back")

image = engine.get\_element\_image(e)

logger.debug("Image = {0}".format(image))

上面的代码在sample/Unity/interaction.py中，运行该函数可以获取图片名称"back"

3.5 device

engine.get\_device()获取的device实例提供与手机相关信息的API，也提供简单的操作。示例：sample/devices\_tester.py

3.5.1 获取屏幕尺寸与方向

def test\_get\_display\_size():

display\_size=device.get\_display\_size()

logger.debug(display\_size)

rotation=device.get\_rotation()

logger.debug("Rotation : {0}".format(rotation))

test\_get\_display\_size()

DisplaySize类包括width、height单位为px。



3.5.2 顶层package与activity

def test\_get\_top\_package\_activity():

top\_activity=device.get\_top\_package\_activity()

logger.debug(top\_activity)

test\_get\_top\_package\_activity()

上面的代码可以保存为devices\_tester.py,再任意界面启动

python devices\_tester.py

*device.get\_top\_package\_activity()*获取android设备当前界面的TopActivity对象，包含顶层app的包名和Activity名称。

package name = com.tencent.wetest.demo,activity = com.unity3d.player.UnityPlayerActivity

3.5.3 回退键

GAutomator本身不提供对标准Android控件的支持，所以当界面上出现标准控件时将无法进行操作。因此，提供了回退（Back）操作，返回到游戏Activity。

def test\_back():

device.back()

test\_back()

上面的代码可以保存为devices\_tester.py,再任意界面启动

python devices\_tester.py

*device.back()*与按Android的回退键效果一致。

3.5.4 多点触控

GAuaumator集成了minitouch框架，能够在设备上模拟按下，移动和松开手指的操作。

device.touchDown(0,100,700,True)

device.touchDown(1,900,700,True)

device.touchMove(0,450,700,True)

device.touchMove(1,500,700,True)

device.touchUp(0)

device.touchUp(1)

上面的代码模拟了两个手指从屏幕两边向中间滑动的操作，可以保存为devices\_tester.py,再任意界面启动

python devices\_tester.py

3.6 Reporter

*engine.get\_reporter()*获取的Reporter类封装了与云端报告相关的内容，本地实现为空，只有在云端运行的时候才会有效果。游戏自动化测试过程中需要保持测试现场，所以在云端运行过程中需要标记测试过程和截图。

import sys, os, time

#sys.path.append(os.path.abspath(os.path.join(os.getcwd(), "..\\")))

import wpyscripts.manager as manager

engine = manager.get\_engine()

logger = manager.get\_logger()

reporter = manager.get\_reporter()

def screen\_shot\_click(element):

logger.debug("screen\_shot\_click")

if element is None:

return

bound = engine.get\_element\_bound(element)

logger.debug(bound)

pos\_x = bound.x + bound.width / 2

pos\_y = bound.y + bound.height / 2

reporter.capture\_and\_mark(pos\_x, pos\_y, locator\_name = element.object\_name)

engine.click\_position(pos\_x, pos\_y)

def enter\_find\_elmeents():

find\_elements\_button = engine.find\_element("/Canvas/Panel/FindElements")

logger.debug(find\_elements\_button)

screen\_shot\_click(find\_elements\_button)

time.sleep(1)

def back\_main():

find\_elements\_button = engine.find\_element("/Canvas/Panel/Back")

logger.debug(find\_elements\_button)

screen\_shot\_click(find\_elements\_button)

time.sleep(1)

def test\_capture\_and\_mark():

elements = engine.find\_elements\_path("/Canvas/Panel/VerticalPanel/Item(Clone)")

for element in elements:

screen\_shot\_click(element)

time.sleep(2)

def test\_reporter():

print("test\_reporter")

enter\_find\_elmeents()

time.sleep(2)

reporter.add\_start\_scene\_tag("Find\_Scene")

test\_capture\_and\_mark()

reporter.add\_end\_scene\_tag("Find\_Scene")

time.sleep(2)

back\_main()

reporter.screenshot()

runner.py里，调用test\_reporter()。

import traceback

try:

from sample.reporter\_tester import \*

except Exception,e:

traceback.print\_exc()

def run():

"""

Begin of test logic

"""

try:

test\_reporter()

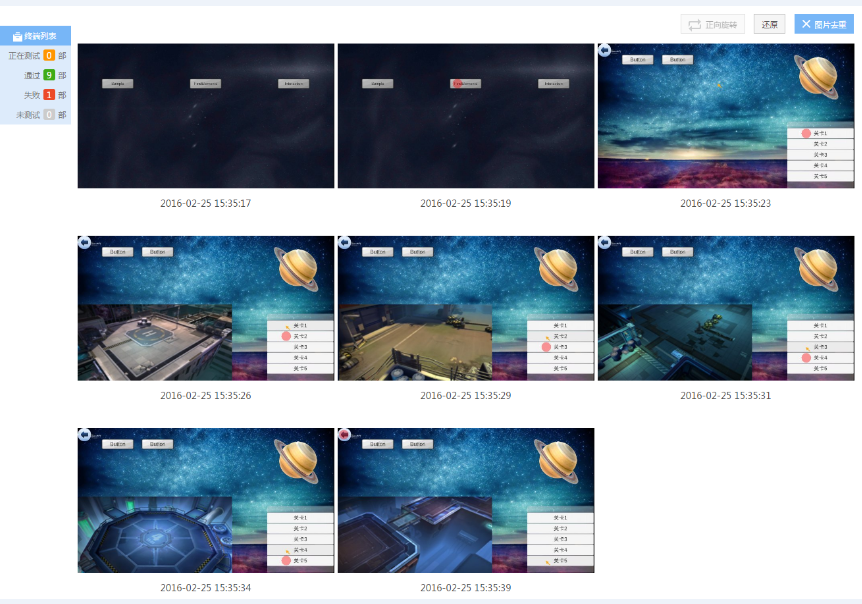
except Exception,e:

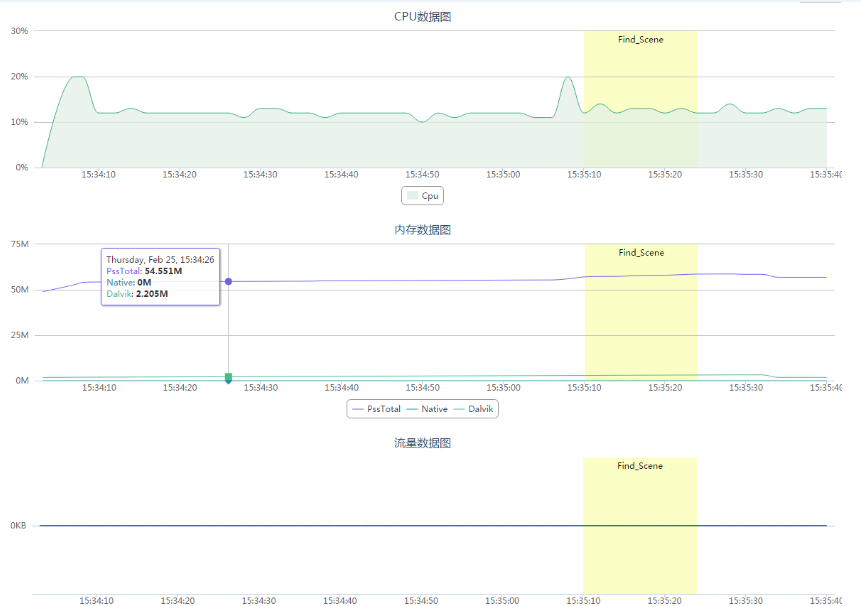
traceback.print\_exc()

stack=traceback.format\_exc()

logger.debug(stack)

上传到平台后的运行结果（同时在几百台手机上运行）,包括运行时截图，性能数据，设备日志，游戏运行错误等信息。





3.6.1截图与操作过程标记

def screen\_shot\_click(element):

logger.debug("screen\_shot\_click")

if element is None:

return

bound = engine.get\_element\_bound(element)

logger.debug(bound)

pos\_x = bound.x + bound.width / 2

pos\_y = bound.y + bound.height / 2

reporter.capture\_and\_mark(pos\_x, pos\_y, locator\_name = element.object\_name)

engine.click\_position(pos\_x, pos\_y)

reporter.capture\_and\_mark(pos\_x, pos\_y, locator\_name = element.object\_name)将会截取当前手机屏幕，并在pos\_x,pos\_y位置标记一个红点。



3.6.2 截图

*reporter.screenshot()*在云端会截图在报告里面体现，在本地运行时会截图并放在运行目录下的screenshot目录下面。

3.6.3 打标签

reporter.add\_start\_scene\_tag("Find\_Scene")

reporter.add\_end\_scene\_tag("Find\_Scene")

reporter.add\_start\_scene\_tag("")和reporter.add\_end\_scene\_tag("")一定是成对出现的，先start然后end，里面的标签内容需要一样。

通过打标签，用户能够在报告中的性能数据中看到每个场景的执行情况。



3.6.4 报告错误

GAutomator并不是使用常见的unittest作为测试的底层框架，但通过reporter接口，也能上报功能用例执行结果。report接口，可用于错误报告，并且在运行目录下生成一份\_wetest\_testcase\_result.txt用户记录报告的内容。该文件的报告格式与unittest的测试报告格式一致，因此在云端测试时可显示具体的信息。

report.report(True,"testName","test first content")

*reporter.* *report(result,test\_case\_name, message*接口调用的过程中，会在日志中输出。脚本运行结束时，runner.run中会调用\_report\_total(),将所有的判断结果输出到\_wetest\_testcase\_result.txt中。除了输出message和test\_case\_name之外GAutomator还会打印调用堆栈。name的名称尽可能不要重复。name与message传入的编码方式需要一致，如果存在中文必须使用UTF-8编码格式。

3.7 定制功能（与unity交互）

GAutomatorView不可能集成所有的功能，部分功能也不方便通过简单的触屏操作完成，或者通过触屏操作完成的复杂度极高。定制功能，可以向Gautomator SDK注册委托，通过python脚本来触发委托的执行，并将结果返回给python脚本。或通过python调用第三方C#脚本完成特定功能,或在Unity中回调python方法.

3.7.1 注册与委托

3.7.1.1 unity端注册函数

Unity游戏开发者可以注册函数供脚本调用，如完成英雄位移等

using UnityEngine;

using System.Collections;

using WeTest.U3DAutomation;

public class CustomTester : MonoBehaviour {

void Start () {

Debug.Log("Register test");

WeTest.U3DAutomation.CustomHandler.RegisterCallBack("test", testReq);

}

string testReq(string args)

{

Debug.Log("Args = " + args);

string result = args + " Response";

return result;

}

void OnDestroy()

{

Debug.Log("UnRegister test");

CustomHandler.UnRegisterCallBack("test");

}

}

WeTest.U3DAutomation.CustomHandler.RegisterCallBack("test", testReq):主要一个注册的函数的名称和一个函数。注册了这个函数之后，脚本执行时如果发送执行test，SDK就会调用testReq（string arg）这个函数，并把脚本发送过来的内容作为string参数传入。函数返回结果会返回给脚本端。CustomHandler.UnRegisterCallBack("test"):将函数从注册表中移除。

3.7.1.2 python获取可用委托

engine.call\_registered\_handler("test", "python call test"):可以调用SDK中注册的委托

def test\_call\_registered\_handler():

result = engine.call\_registered\_handler("test", "python call test")

logger.debug(result)

test\_call\_registered\_handler()

确保wetest\_demo\_unity.apk在拉起来的界面

python self\_define\_fun.py

运行"test"关键词对应的注册的委托，传入参数为"python call test"。获取委托执行后的返回值"python call test Response"

3.7.1.3 python执行委托

engine.call\_registered\_handler("test", "python call test"):可以调用SDK中注册的委托

def test\_call\_registered\_handler():

result = engine.call\_registered\_handler("test", "python call test")

logger.debug(result)

test\_call\_registered\_handler()

确保wetest\_demo\_unity.apk在拉起后的界面

python self\_define\_fun.py

运行"test"关键词对应的注册的委托，传入参数为"python call test"。获取委托执行后的返回值"python call test Response"。

3.7.1.4 python获取组件方法

可以获取到游戏中的GameObject上某个Component上的public方法信息，包括方法名称，方法需要的参数和返回的类型。在wetest\_demo\_unity.apk中，Sample按钮上挂载了ReflectionTest组件，可以使用下面代码返回该组件中的方法。

def test\_get\_component\_methods(self):

element = self.engine.find\_element("Sample")

methods = self.engine.get\_component\_methods(element, "ReflectionTest")

logger.debug(methods)

上面的代码在sample/Unity/interaction.py中，运行该函数可以获得全部的public方法。

3.7.1.5 python调用组件方法

通过反射可以调用GameObject某个组件上的public方法，并获得返回值。调用时需要传入组件名称、方法名称和参数列表。wetest\_demo\_unity.apk中调用Sample按钮ReflectionTest组件上的TestReflection方法（需要两个参数int,string,返回int值），调用代码如下：

def test\_call\_component\_method(self):

element = self.engine.find\_element("Sample")

params = []

params.append(5)

params.append("Hello World")

result = self.engine.call\_component\_method(element, "ReflectionTest", "TestReflection", params)

logger.debug(result)

上面的代码在sample/Unity/interaction.py中，将调用游戏中的TestReflection方法，并返回105（方法的返回值）。

3.7.2 反射获取游戏中属性

在使用自动化测试过程中，定制一些高级功能时，现有的接口获取的数据可能无法满足需求。如，希望根据英雄血量来定制策略。因此，GAutomator提供了一个高级接口，通过反射的方式获取游戏中组件里面的属性值。*get\_component\_field(element,component,attribute)*接口可以获取GameObject上组件对应的属性值。

def test\_get\_component\_field(self):

elements = self.engine.find\_elements\_path("Sample/Text")

self.\_start()

res = self.engine.get\_component\_field(elements[0], "Text", "text")

self.\_end("get\_component\_field")

self.assertEqual(res, "Sample")

e = self.engine.find\_element("Panel")

res = self.engine.get\_component\_field(e, "MainControl", "bollon")

self.assertEqual(res, "Bollon (UnityEngine.RectTransform)")

确保wetest\_demo\_unity.apk在拉起来的界面，运行结果可以看到获取了Sample/Text GameObject上面组件Text中对应text属性的值为Sample。GAutomator获取到组件属性后，会直接调用toString()转换为string字符串。如果属性本身也是Object，toString()之后的值可能意义并不会特别大，如（Bollon (UnityEngine.RectTransform)）。目前暂未提供级联的操作，无法获取对象Bollon上的属性值。

* + 1. 设置最佳渲染camera

Unity里面一个物体可能会被多个Camera渲染，如有一个主摄像机还会有光晕渲染等摄像机。GAutomator SDK中会寻找一个最佳的Camera，但是有时候可能找到的Camera并不准确。具体表现为，渲染的物体大小及位置不准确，可能是远大于实际的长宽。  
GAutomatorView查看物体，发现长宽高不对。可以使用set\_camera设置其他相机，然后再用GAutomatorView尝试看是否恢复正常，恢复正常了则该GameObject为最适合的摄像机。engine中et\_camera(gameobject\_name)：设置Camera所在的gameobject名称。

engine.set\_camera("CharModeCamera")

设置Camera后，如果物体渲染的Camera中包含设置的Camera，则会直接采用设置的Camera。

[UnityCamera资料](https://docs.unity3d.com/ScriptReference/Camera.html)

* + 1. 调用第三方C#脚本

Unity手游自动化过程中，纯UI的自动化面临越来越大的瓶颈，如人物的自动寻路、人物的战斗属性获取等通过UI的方式获取困难。因此，GAutomator一个功能，允许将自己的C#脚本注入到游戏中。能够以C#脚本的方式，调用游戏内的接口，获取游戏内的数据。  
GAutomator python端提供了接口engine.game\_script\_init(),会将gametestlib.dll推送到/data/local/tmp，然后调用gametestlib.dll下的GameTest.Test.init方法，该方法无参数与无返回值。因此，通常会在改接口中调用WeTest.U3DAutomation.CustomHandler.RegisterCallBack，将需要调用的函数注册上去，利用engine.call\_registered\_handler机制完成python调用游戏中C#脚本。

def test\_game\_script\_init(self):

file\_path = os.path.split(os.path.realpath(\_\_file\_\_))[0]

path=os.path.join(file\_path,"gametestlib.dll")

result=self.engine.game\_script\_init(path)

logger.debug("init result : ".format(result))

result=self.engine.get\_registered\_handlers()

logger.debug("registered functions : {0}".format(result))

result=self.engine.call\_registered\_handler("showColider","")

启动wetest\_demo，点击Joystick按钮，进入到摇杆页面。然后运行代码，所有的柱子会变成粉红色。执行了gametestlib.dll中的，showColider方法。

[gametestlib source](https://share.weiyun.com/b27da45dc1013db82b09cd4a31625056)编写的时候，注意事项：  
1、vs编写，工程名称任意。Assembly name为gametestlib  
2、非IL2CPP编译的apk，用压缩软件打开，在assets\bin\Data\Managed下的dll为游戏脚本。可以作为gametestlib编写的依赖。  
3、初始化模板固定

namespace GameTest //固定 namespace为GameTest

{

public class Test //初始化类名固定

{

public static void init() //初始化函数，GA只支持调用该函数，在该函数中注册委托

{

WeTest.U3DAutomation.CustomHandler.RegisterCallBack("TEST", test);

WeTest.U3DAutomation.CustomHandler.RegisterCallBack("RecordPath", record);

WeTest.U3DAutomation.CustomHandler.RegisterCallBack("showColider", addBoxColider);

}

}

}

* + 1. c#回调python方法

SDK 1.5.0版本提供接口，可以调用python注册的函数。该功能类似于RPC调用，C#脚本调用python端的实现，可传递string参数与获取返回值。

SDK接口：

WeTest.U3DAutomation.CustomHandler.InvokeClientMethod(string name, string params,PcCallBack callback)

异步调用，name为python端注册的函数名称，params为传递的参数（以json方式传输），callback为python端执行完毕后的回调函数。

WeTest.U3DAutomation.CustomHandler.InvokeClientMethodReturnValue(string name, string params,int timeout)

同步调用，name为python端注册的函数名称，params为传递的参数（以json方式传输），timeout为超时时长(毫秒)。返回值string,为远程python注册函数的返回值。

WeTest.U3DAutomation.CustomHandler.InvokeClientMethod(string name, string value)

异步调用（无返回值）,name为python端注册的函数名称，params为传递的参数（以json方式传输）。调用后立刻返回。

wetest\_demo首页的CallPc注册了3个点击处理函数,c#代码：

public static void printResult(int status, string message)

{

Debug.Log("call back result = " + message + " status = " + status);

}

public void CallPythonClient(GameObject obj)

{

/\*

\* call python register function "\_print\_fun"

\*

\* def \_print\_fun(v):

\* print "game call"

\* print "value = "+v

\* return "\_print\_fun call return value"

\*

\* engine.

\*

\*

\* \_game\_callback("test", \_print\_fun)

\*

\*/

WeTest.U3DAutomation.CustomHandler.InvokeClientMethod("test", "call python client callback", printResult);

}

public void CallPythonClientReturnValue(GameObject obj)

{

/\*

\* call python register function "\_print\_fun\_returnvalue"

\*

\* def \_print\_fun\_returnvalue(v):

\* print "game call return value"

\* print "value = "+v

\* return "\_print\_fun\_returnvalue call return value"

\*

\* engine.register\_game\_callback("testReturn", \_print\_fun\_returnvalue)

\*

\*/

string result = WeTest.U3DAutomation.CustomHandler.InvokeClientMethodReturnValue("testReturn", "call python client return value", 1000);

Debug.Log("CallPythonClientReturnValue return value = " + result);

}

public void CallPythonClientReturnNone(GameObject obj)

{

/\*

\* call python register function "\_print\_fun\_returnvalue",ignore the return value

\*

\* def \_print\_fun\_returnvalue(v):

\* print "game call return value"

\* print "value = "+v

\* return "\_print\_fun\_returnvalue call return value"

\*

\* engine.register\_game\_callback("testReturn", \_print\_fun\_returnvalue)

\*

\*/

WeTest.U3DAutomation.CustomHandler.InvokeClientMethod("testReturn", "call python client not return value");

}

python端注册处理函数：

def \_print\_fun(v):

print "game call"

print "value = "+v

return "\_print\_fun call return value"

def \_print\_fun\_returnvalue(v):

print "game call return value"

print "value = "+v

return "\_print\_fun\_returnvalue call return value"

result = engine.register\_game\_callback("test", \_print\_fun)

logger.debug("register game callback {0}".format(result))

result = engine.register\_game\_callback("testReturn", \_print\_fun\_returnvalue)

logger.debug("register game callback {0}".format(result))

callPcButton=engine.find\_element("CallPC")

engine.click(callPcButton)

3.8 实战用例

3.7.1 摇杆操作

MOBA游戏越来越流行，摇杆类游戏区别需要滑动和按压连续操作。Wpyscripts提供了专门针对摇杆类动作的封装engine.swipe\_and\_press(start\_x, start\_y, end\_x, end\_y, steps, duration, step\_sleep=5)能够实现，对大部分摇杆的操作。Demo：sample/Unity/joystick\_tester.py

def convert\_pos(x, y):

display\_size = device.get\_display\_size()

return x \* display\_size.width, y \* display\_size.height

def test\_swipe\_and\_press():

time.sleep(2)

start\_time=datetime.datetime.now()

start\_x,start\_y=convert\_pos(0.1197916,0.796296)

end\_x,end\_y=convert\_pos(0.1197916,0.69444)

engine.swipe\_and\_press(start\_x,start\_y,end\_x,end\_y,100,10000)

logger.debug("Use time : {0}".format(datetime.datetime.now()-start\_time))

time.sleep(3)

test\_swipe\_and\_press()

上面的代码可以保存为joystick\_tester.py,点击JoyStick按钮，进入摇杆操作界面：

python joystick\_tester.py

start\_x,start\_y,表示滑动的起始位置

end\_x,end\_y,表示滑动的结束位置

steps，表示滑动中间经过的步骤数，每一步的时间约为5ms。滑动的步骤数控制着滑动的速度和平滑度

step\_sleep:每个步骤的间隔时长，单位毫秒

duration，结束位置按压时间，单位是毫秒ms

3.7.2 随机操作

进入战斗场景后，我们通常可以在里面进行随机操作，直到比赛结束。scripts/testcase/tools.py里面封装了一个random\_click(fun=None, forbid\_elements=(),max\_num=1000,sleep = 2)

def random\_click(fun=None, forbid\_elements=(),max\_num=1000,sleep = 2):

"""

随机点击界面上的可操作控件。

:param fun: 如果fun调用返回True，则随机点击结束

:param forbid\_elements: 禁止点击的组件列表(如退出键)

:param max\_num:最大点击次数

:param sleep:每次点击后的睡眠时间

:return:

"""

logger.debug("Random click")

elements = engine.get\_touchable\_elements()

for i in range(max\_num):

if elements is None or len(elements) <= 0:

time.sleep(1)

elements=engine.get\_touchable\_elements()

continue

if fun and fun(elements):

logger.info("Find need elements")

return

elements = filter(lambda e: e[0].object\_name not in forbid\_elements,elements)

e,pos = find\_less\_click\_element(elements)

if pos is None:

continue

screen\_shot\_click\_pos(pos["x"], pos["y"], e.object\_name)

time.sleep(sleep)

elements = engine.get\_touchable\_elements()



上面王者荣耀这个游戏，进入游戏战斗场景后，就可以随意操作界面里面的按钮（除了暂停），如技能、英雄切换、攻击。如果，比赛胜利，则会弹出"点击屏幕继续"这个element。

def get\_condition\_fun(\*name):

"""

主要用于random\_click，传入一系列的elements，只要可点击的节点里面有符合的就返回

应用场景：需要点击某个节点，但是在该界面可能出现弹出框。弹出框不确定的情况下，可以使用

:param name:

:return:

"""

def find\_need\_element(elements):

for e,pos in elements:

if e.object\_name in name:

logger.info("Find element name {0}".format(e.object\_name))

return True

return False

return find\_need\_element

# 比赛随机点

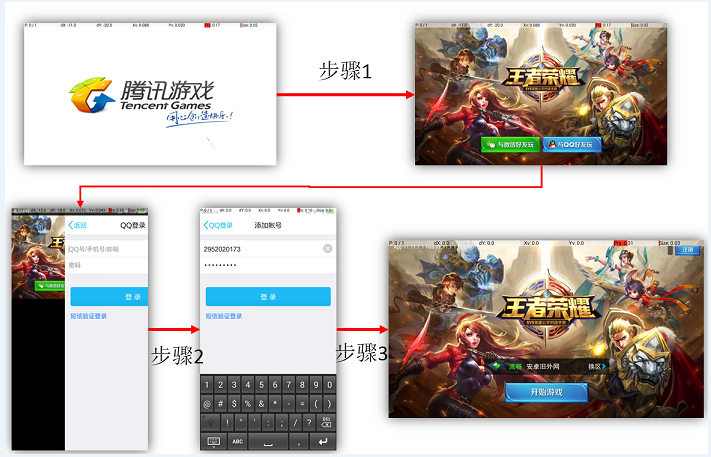
random\_click(get\_condition\_fun("/Root/Panel/ContinueBtn"),

("/Root/Pause/Button"))

可点击的节点中出现"/Root/Panel/ContinueBtn"则random\_click退出。界面中的"/Root/Pause/Button"不需要进行点击。random\_click()会优先点击次数最少的按钮。

3.7.3 QQ/微信登录

QQ或者微信登录，涉及到Activity的切换和Android标准控件的操作,操作过程复杂，但是相对较为固定。在云端运行时，每次拉起游戏之前，都会清理数据，所以每次都需要重新登录。每次登录的过程大致如下所示：



对应的处理代码如下所示：

def login():

# 步骤1，等待到达登录界面

wait\_for\_scene("SceneName")

# 选择QQ登陆

qq\_button = find\_elment\_wait("/BootObj/Panel/btnQQ")

screen\_shot\_click(qq\_button, 6)

#步骤2 ，等待进入QQ登录界面，packagename为com.tencent.mobileqq，如果是微信登录界面package为com.tencent.mm

wait\_for\_package("com.tencent.mobileqq")

device.login\_qq\_wechat\_wait(120)

time.sleep(10)

#步骤3，等待QQ登录界面退出，切换到游戏界面

select\_btn = find\_elment\_wait("/BootObj/Panle/selectBtn")

步骤1：等待进入到登录选择scene，如何获取scene名称，请看[1.4 GAutomatorView](local://base_request.html/#1.4)。wait\_for\_scene("SceneName")，会一直查询，直到进入名称为"SceneName"的场景。进入到"SceneName"的场景后，查询QQ登录按钮直到出现(find\_elment\_wait)，并点击QQ登录按钮。

步骤2：从游戏的Activity切换到QQ或者微信的登录界面需要一定的时间。wait\_for\_package("com.tencent.mobileqq")检查顶层包名，直到QQ的顶层包名(微信包名为com.tencent.mm)。device.login\_qq\_wechat\_wait(120)会根据当前的顶层包名，自动选择QQ或者微信登录，当顶层包名不再是"com.tencent.mm"或"com.tencent.mobileqq"时退出。  
**注：账号由云端自动分配。本地调试时请修改wpyscripts/wetest/device.py下面native\_deivce.\_\_init\_\_(self)中的账号密码**

步骤3：等待进入游戏界面，直到出现某个element为止。

3.7.4 异常处理

对于GAutomator的异常处理是一件非常头痛的事情，在设计框架的过程中也是左右为难，本质原因在于手机的端的不稳定性。不稳定主要包括以下几方面：

1. adb不稳定: windows的adb及其不稳定长期连接过程中不可避免的会出现断开连接的情况。如个人PC上被其他应用程序抢占adb端口。wetest平台重写了adb，并且运行在linux之上稳定性好很多。adb断开连接，不可恢复，脚本退出。

2. 游戏不稳定: SDK部分与UI相关的内容运行在UI主线程，当游戏暂停时可能会出现timeout的情况。如，QQ登录按钮跳转到登录界面，分享按钮，游戏会退出前台主线程暂停。

3. UIAutomator不稳定：UIAutomator并不是一个非常稳定的服务，可能会出现操作无效的情况。

4. 框架本身，只要是框架处理不了的异常，都会抛给调用者。测试编写的过程中，如果出现操作的内容，可能会让游戏退出前台engine相关接口抛异常时尽量catch。对于操作可有可无的，也尽量catch。如点击操作不影响测试流程，如攻击按钮，可以选择catch。

3.9 更上层的封装

GAutomator主要的大三类接口engine,reporter,device，它们属于颗粒度非常细的接口。但是直接使用这些接口进行开发并不是一件非常容易的事情。所以，根据在实际项目中使用的经验，我们封装了一些更加方便的、易于使用、不容易出错的接口，供开发人员快速的开发出稳定有效的测试用例。

3.8.1 screen\_shot\_click

*screen\_shot\_click(element, sleeptime=2, exception=False)*  
element:可以为Element实例，也可以为需要点击的name  
sleeptime:点击完成后sleep的时间  
exception:异常发生时，如果exception为True则抛出异常，如果exception为False则不抛出异常返回False

实际使用时，该接口基本可以替代GameEngine.click。操作流程为截图->点击的位置标记红点->点击->sleep指定的时间。这个操作过程是比较理想的，也是一个最基本的操作。

wetest平台截图的速度非常快，对性能影响也极低，可以对每一个操作步骤均进行截图。

example:

from testcase.tools import \*

button=engine.find\_element("LoginQQ")

screen\_shot\_click(button,sleeptime=5,exception=True)

screen\_shot\_click("Attack",sleeptime=0)

3.8.2 screen\_shot\_click\_pos

*screen\_shot\_click\_pos(pos\_x,pos\_y, sleeptime=2, exception=False)*  
pos\_x:x坐标位置  
pos\_y:y坐标位置  
sleeptime:点击完成后sleep的时间  
exception:异常发生时，如果exception为True则抛出异常，如果exception为False则不抛出异常返回False

screen\_shot\_click\_pos与screen\_shot\_click的区别是，一个点击的是UI控件，一个纯粹是位置信息。操作流程是一致的：截图->点击的位置标记红点->点击->sleep指定的时间。