Android 自动化测试初探（一）：捕获 Activity 上的 Element

第一部分：前言

Android 系统下应用程序的测试现在应该还算是个新的领域，网上关于这方面的资料很多都是基于白盒测试的，一般都是基于 JUnit 框架和 Android SDK 中 android.test 等命名空间下的内容进行，但是有一个前提，那就是必须要有应用程序的源代码以提供测试接入点，但是这在很多软件公司中是不现实的。很多测试工程师做的工作是完全黑盒，基本接触不到源代码，白盒测试大部分也是由开发自己完成。

回顾一下 Windows 下的黑盒测试自动化，先前使用的是微软提供的基于.net framework 的 UI Automation 自动化测试框架（要求版本在.net framework 3.0 以上，即 VS.NET 2008 开发环境），对与擅长 C#语言的人来说，使用起来确认比较好用。本人也写了基于 UI Automation 的轻量级的自动化框架，将在以后的博文中引出。

那在 Android 操作系统中能不能做类似于 UI Automation 的事情呢？不幸的是，Android 的权限控制分的非常清楚，不同程序之间的数据访问只能通过 Intent，content provider 类似的功能实现。也就是说你开发的运行在 Android 中的自动化程序想要捕获当前运行的 AUT (Application under Test) 界面上的控件等 Element（该术语引自 UI Automation，觉得翻译成元素有点生硬，故不作翻译）基本不可能，你也拿不到当前 active activity 的引用（截止本文发帖为止，个人暂时没有找到办法获得此引用）。

无路可走了？模拟器里面不能走，外面能不能走？或许可以。

第二部分：捕获 Activity 上的 Element

在 Android 的 SDK 中自带了一个对自动化测试比较有用的工具：hierarchyviewer（位于 SDK 的 tools 目录下）。在模拟器运行的情况下，使用该工具可以将当前的 Activity 上的 Element 以对象树的形式展现出来，每个 Element 所含的属性也能一一尽显。这有点像 Windows 上运行的 UI SPY，唯一遗憾的是不支持事件的触发。不过没有关系，可以想办法绕，当务之急是能在自行编写的自动化测试代码里找到 Activity 上的 Element。

第一个想到的办法就是看 hierarchyviewer 源码，不巧，网上搜了一下，没有资源。或许 Google 的官网上有，但是上不去。看来只能反编译了，找来 XJad，暴力之。虽然反编译出来的代码很多地方提示缺少 import，但代码基本上是正确的。看了一下，确实也知道了许多。后来在编写代码的过程中，确实也证明了如果想引用 hierarchyviewer.jar 这个包并调试，还是需要知道里面的一些设置的。

创建基于 hierarchyviewer.jar 这个包的调用，需要将它和另外两个包，ddmlib.jar（在 hierarchyviewer.jar 同级目录中有）和 org-netbeans-api-visual.jar（需要下载并安装 netbeans，在其安装目录中有）一并导入到工程项目中，因为 hierarchyviewer 的实现依附于这两个包。

想在代码中获取 Activity 上的 Element 需要进行如下几个步骤（如果使用过 hierarchyviewer 这个工具后会发现，自动化代码所要写的就是该工具上的使用步骤）：

1. **Ensure adb running**
2. **Set adb location**（因为 hierarchyviewer 和模拟器的沟通完全是依靠 adb 做的，所以设定正确的 adb 程序的位置至关重要，本人就曾在这个问题上栽了半天多）

3. **Get Active Device** (这个等同动作发生在启动 hierarchyviewer 工具时)
4. **Start View Server** (等同于工具上 Start Server 菜单触发事件)
5. **Load Scene** (等同于工具上 Load View Hierarchy 菜单触发事件)
6. **Get Root View Node** (获得对象树的根节点，这个动作在工具上点击 Load View Hierarchy 菜单后会自动加载)
7. **Get Sub View Node** (获得想要查找的 View Node，这个动作在工具上点击 Load View Hierarchy 菜单后会自动加载)

说明：上述步骤中一些名称实际上就是 hierarchyviewer 中所提供的可访问方法名称，如 startViewServer、loadScene、rootNode 等。另外 View Node 实际上 hierarchyviewer 中的一个类，表示的对象树上的一个 Element。

现将部分核心代码粘贴如下：

1. Set adb location

```
System.setProperty("hierarchyviewer.adb","E:\\ \\Android\\android-sdk-windows\\tools");
```

其中"hierarchyviewer.adb" 这个 key 是 hierarchyviewer.jar 中指定的，后面的 value 是存放 Android SDK 的路径。这个目录必须是当前运行的模拟器所对应的 adb 的目录，不能自行使用其他目录下 adb，否则会发生 adb 进程异常退出的错误。

2. Get Active Device

```
IDevice[] devices = null;
DeviceBridge.terminate();
while(null==devices || 0==devices.length){
    DeviceBridge.initDebugBridge() ;
    //it must wait for some time, otherwise will throw exception
    try {
          Thread.sleep(1000);
    } catch (InterruptedException e) {
           e.printStackTrace();
    }
    devices = DeviceBridge.getDevices() ;
}
return devices;
```

以上方法返回的是所有当前运行的 Device 列表

3. Start View Server

```
DeviceBridge.startViewServer(device);
```

4. Load Scene

```
ViewHierarchyLoader.loadScene(device,Window.FOCUSED_WINDOW) ;
```

5. Get Root View Node

vhs.getRoot() ;

其中 vhs 是 ViewHierarchyScene 的实例对象

6. Get Sub View Node

```
public ViewNode findFirstChildrenElement(IDevice device, ViewNode
entryViewNode, String elementID){
  ViewNode node=null;
  if(0!=entryViewNode.children.size()){
        for(int i=0;i<entryViewNode.children.size();i++){
              node=entryViewNode.children.get(i);
              if(node.id==elementID)
                    return node;
              else
                    continue;
        }
  }
  return node;}
```

虽然上述步骤所涉及的代码量不多，但是花了一天多的时间才终究研究出来，写下此文希望对正在研究 Android 自动化测试的同学们有些帮助。

到目前为止，Element 已经得到了，接下去就是实现怎么去触发这些 Element，如 click button，enter text 等等，尚需再慢慢研究，等有结果再贴出来分享！

Android 自动化测试初探（二）：Hierarchyviewer 捕获 Element 的实现原理 收藏

Android SDK tools 下的工具 hierarchyviewer 可以展现 Device 上的 Element 的层次分布和自身属性，其核心函数之一就是 LoadScene，研究后发现其实现方法是向 Device 的 4939 端口通过 socket 的方式发送了一个 DUMP 的命令，Device 会自动处理该命令并将所有 Screen 上的 Element 层次结构和属性一并发回，实现代码如下：

```
public static void listElement(IDevice device){
       Socket socket;
      BufferedReader in;
      BufferedWriter out;
      socket = null;
      in = null;
      out = null;

      do{
        DeviceBridge.setupDeviceForward(device);
      }while(4939!=DeviceBridge.getDeviceLocalPort(device));

      socket = new Socket();
      try {
```

```
            socket.connect(new InetSocketAddress("127.0.0.1",
            DeviceBridge.getDeviceLocalPort(device)));

            out
            = new BufferedWriter(new OutputStreamWriter(socket.getOutputStr
            eam()));

            in
            = new BufferedReader(new InputStreamReader(socket.getInputStrea
            m()));

            System.out.println("==> DUMP");
            out.write((new StringBuilder()).append("DUMP -1").toString());

            out.newLine();
          out.flush();
          do
          {
                String line;
                  if ((line = in.readLine())
                  == null || "DONE.".equalsIgnoreCase(line))
                      break;
                line = line.trim();
                System.out.println(line);
          } while (true);
      } catch (IOException e) {
                e.printStackTrace();
      }
}
```

运行后的结果摘录其中一部分（button5），列举如下。注：当前 device 中运行的是 2.1SDK 中自带的 Calculator 程序：

```
com.android.calculator2.ColorButton@43b8bee8 mText=1,5 getEllipsize()=4,null
mMinWidth=1,0 mMinHeight=1,0 mMeasuredWidth=2,79 mPaddingBottom=1,0
mPaddingLeft=1,0 mPaddingRight=1,0 mPaddingTop=1,0 mMeasuredHeight=2,78
mLeft=2,81 mPrivateFlags_DRAWING_CACHE_INVALID=3,0x0
mPrivateFlags_DRAWN=4,0x20 mPrivateFlags=8,16779312 mID=9,id/digit5
mRight=3,160 mScrollX=1,0 mScrollY=1,0 mTop=1,0 mBottom=2,78
mUserPaddingBottom=1,0 mUserPaddingRight=1,0 mViewFlags=9,402669569
getBaseline()=2,54 getHeight()=2,78 layout_gravity=4,NONE layout_weight=3,1.0
layout_bottomMargin=1,0 layout_leftMargin=1,1 layout_rightMargin=1,0
layout_topMargin=1,0 layout_height=11,FILL_PARENT layout_width=11,FILL_PARENT
getTag()=4,null getVisibility()=7,VISIBLE getWidth()=2,79 hasFocus()=5,false
isClickable()=4,true isDrawingCacheEnabled()=5,false isEnabled()=4,true
```

```
isFocusable()=4,true isFocusableInTouchMode()=5,false isFocused()=5,false
isHapticFeedbackEnabled()=4,true isInTouchMode()=4,true isOpaque()=5,false
isSelected()=5,false isSoundEffectsEnabled()=4,true
willNotCacheDrawing()=5,false willNotDraw()=5,false
```

另外还支持如下命令：

```
- LIST will show the list of windows:
LIST
43514758 com.android.launcher/com.android.launcher.Launcher
4359e4d0 TrackingView
435b00a0 StatusBarExpanded
43463710 StatusBar
43484c58 Keyguard
DONE.
```

Android 自动化测试初探（三）：架构实现 收藏

前两节讲了用 Android SDK 自带的 tool-hierarchyviewer 来捕获 Activity 上 Element，并分析了其中的原理。对于要实现 GUI 自动化，还有哪些工作没有完成呢？

- Invoke 界面上的 Element，如点击按钮，在文本框中输入内容等
- Press 手机自身所有的按键，如 HOME 键，Menu 键，左右上下方向键，通话键，挂机键等
- 判断测试结果

前面说过，直接从 Emulator 内部获取当前 Activity 上的 Element 这条路已经断了，同理，探索像 UI Automation 上一样 Invoke Element 的操作估计是行不通了，因为你拿不到 Element 的对象实例，所以实例所支持的方法当然也没有办法拿到。

怎么办？实在不行，基于坐标来对 Element 进行触发总可以吧。在 Windows 中发送基于坐标发送键盘和鼠标事件一般是在无法识别 Element 的情况下，想的最后一招，这使我想起起了 Android 中的 monkey 测试，对着屏幕就是一通乱点，压根就不管点的是什么。所幸的是，当前 Android 系统中我们得到了 Element 的属性信息，其中就包括坐标信息，而且这种信息是具有弹性的，也就是说即使 Element 的坐标随着开发的改变而有所变化，也不用担心，因为当前的坐标是实时获得的。

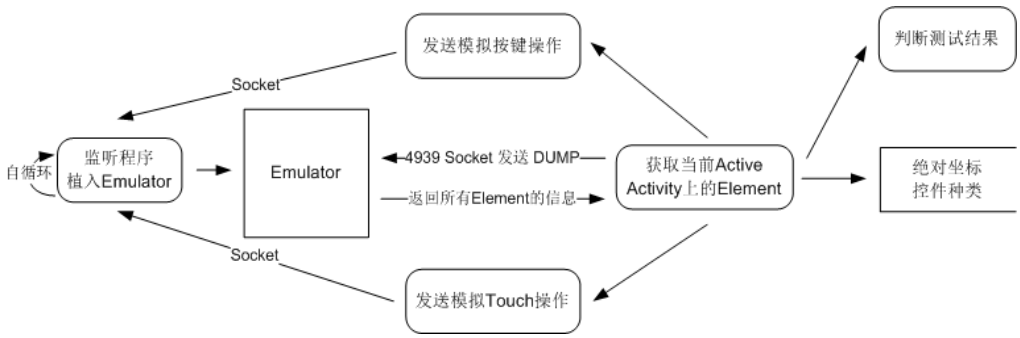
那么怎样才能给 Element 发送模拟按键等操作呢？总不能用 Windows 当前的键盘和鼠标事件吧，那样一旦模拟器的位置改变或失去焦点，啥都白搭，风险太大了。看来给 Emulator 内部发送模拟按键等操作比较靠谱。查了一下 SDK，其中确实有这样的方法存在，但是我们当前的测试基础架构程序位于 Emulator 外部，怎么办？突然想起了 hierarchyviewer 的实现机制，通过 Socket 来发送信息。Hierarchyviewer 有系统自带的进程给予答复响应（具体是哪个进程进行的响应不清楚，没有研究过）。那么我们也来模拟做一个 Listener 总可以吧。

其实对于模拟按键发送，网上的帖子很多，但大部分是基于一种方式实现的，IWindowManager 接口。不巧的是，SDK 并没有将该接口提供为 public，所以需要用 android 源码替代 android.jar 包进行编译的方式进行绕行，感觉方法有点复杂。在后面另一篇系列文章中我会列出我在网上看到的另一种基于 Instrumentation 和 MessageQueue 的机制实现方法。

最后就剩下判断测试结果了。判断测试结果一般分为如下两种：外部条件是否满足，如文件是否产生，数据是否生成等；内部条件是否满足，如对应的 Element 是否出现或消失，Element 上内容如字符串是否有变化。对于前一种本文不予讨论，后一种情况下，Element 出现或消失可以通过 hierarchyviewer 来获取。仔细研究过 hierarchyviewer 会发现，它并没有提供 Element 界面上内容（Text）的属性。这

可有点晕了，好像又要回到实现捕获 Activity 实例的老路上来了。考虑图像识别？这好像不靠谱。突然想到，4939 端口上发送 DUMP 命令后的返回结果中会不会有此类 hierarchyviewer 没有显示出来的信息呢，万幸，还真有。在我上一篇博文（Hierarchyviewer 捕获 Element 的实现原理）中查询 mText 字段，会发现 mText=1,5 这样的信息，其实就是代表了计算器 Button5 上显示的内容 5，逗号前的 1 表示后跟一位信息。

至此，问题似乎都解决掉了。画个基础架构图做个总结：



Android 自动化测试初探（四）：模拟键盘鼠标事件（Socket+Instrumentation 实现） 收藏

通过 Socket + Instrumentation 实现模拟键盘鼠标事件主要通过以下三个部分组成：

- Socket 编程：实现 PC 和 Emulator 通讯，并进行循环监听
- Service 服务：将 Socket 的监听程序放在 Service 中，从而达到后台运行的目的。这里要说明的是启动服务有两种方式，bindService 和 startService，两者的区别是，前者会使启动的 Service 随着启动 Service 的 Activity 的消亡而消亡，而 startService 则不会这样，除非显式调用 stopService，否则一直会在后台运行因为 Service 需要通过一个 Activity 来进行启动，所以采用 startService 更适合当前的情形
- Instrumentation 发送键盘鼠标事件：Instrumentation 提供了丰富的以 send 开头的函数接口来实现模拟键盘鼠标，如下所述：

  sendCharacterSync(int keyCode)　　//用于发送指定 KeyCode 的按键
  sendKeyDownUpSync(int key)　　//用于发送指定 KeyCode 的按键
  sendPointerSync(MotionEvent event)　　//用于模拟 Touch
  sendStringSync(String text)　　//用于发送字符串

  注意：以上函数必须通过 Message 的形式抛到 Message 队列中。如果直接进行调用加会导致程序崩溃。

对于 Socket 编程和 Service 网上有很多成功的范例，此文不再累述，下面着重介绍一下发送键盘鼠标模拟事件的代码：

1. 发送键盘 KeyCode：

步骤 1. 声明类 handler 变量

```
private static Handler handler;
```

步骤 2. 循环处理 Message

```
//在 Activity 的 onCreate 方法中对下列函数进行调用
private void createMessageHandleThread(){
//need start a thread to raise looper, otherwise it will be blocked
```

```
 Thread t = new Thread() {
    public void run() {
        Log.i( TAG,"Creating handler ..." );
        Looper.prepare();
        handler = new Handler(){
            public void handleMessage(Message msg) {
                //process incoming messages here
            }
        };
        Looper.loop();
        Log.i( TAG, "Looper thread ends" );
    }
 };
 t.start();
}
```

步骤 3. 在接收到 Socket 中的传递信息后抛出 Message

```
handler.post( new Runnable() {
    public void run() {
        Instrumentation inst=new Instrumentation();
        inst.sendKeyDownUpSync(keyCode);
    }
} );
```

2. Touch 指定坐标，如下例子即 touch point（240,400）

```
Instrumentation inst=new Instrumentation();
inst.sendPointerSync(MotionEvent.obtain(SystemClock.uptimeMillis(),SystemClock.uptimeMillis(), MotionEvent.ACTION_DOWN, 240, 400, 0));
inst.sendPointerSync(MotionEvent.obtain(SystemClock.uptimeMillis(),SystemClock.uptimeMillis(), MotionEvent.ACTION_UP, 240, 400, 0));
```

3. 模拟滑动轨迹

将上述方法中间添加 MotionEvent.*ACTION_MOVE*

Android 自动化测试初探（五）：再述模拟键盘鼠标事件（adb shell 实现） 收藏

上一篇博文中讲述了通过 Socket 编程从外部向 Emulator 发送键盘鼠标模拟事件，貌似实现细节有点复杂。其实 Android 还有一种更简单的模拟键盘鼠标事件的方法，那就是通过使用 adb shell 命令。

1. 发送键盘事件：

命令格式 1：adb shell input keyevent “value”

其中 value 以及对应的 key code 如下表所列：

KeyEvent Value	KEYCODE	Comment
0	KEYCODE_UNKNOWN	

1	KEYCODE_MENU	在 SDK2.1 的模拟器中命令失效，sendevent 命令可行
2	KEYCODE_SOFT_RIGHT	
3	KEYCODE_HOME	
4	KEYCODE_BACK	
5	KEYCODE_CALL	
6	KEYCODE_ENDCALL	
7	KEYCODE_0	
8	KEYCODE_1	
9	KEYCODE_2	
10	KEYCODE_3	
11	KEYCODE_4	
12	KEYCODE_5	
13	KEYCODE_6	
14	KEYCODE_7	
15	KEYCODE_8	
16	KEYCODE_9	
17	KEYCODE_STAR	
18	KEYCODE_POUND	
19	KEYCODE_DPAD_UP	
20	KEYCODE_DPAD_DOWN	
21	KEYCODE_DPAD_LEFT	
22	KEYCODE_DPAD_RIGHT	
23	KEYCODE_DPAD_CENTER	
24	KEYCODE_VOLUME_UP	
25	KEYCODE_VOLUME_DOWN	
26	KEYCODE_POWER	
27	KEYCODE_CAMERA	
28	KEYCODE_CLEAR	
29	KEYCODE_A	
30	KEYCODE_B	
31	KEYCODE_C	
32	KEYCODE_D	
33	KEYCODE_E	
34	KEYCODE_F	
35	KEYCODE_G	
36	KEYCODE_H	
37	KEYCODE_I	
38	KEYCODE_J	
39	KEYCODE_K	
40	KEYCODE_L	
41	KEYCODE_M	

42	KEYCODE_N	
43	KEYCODE_O	
44	KEYCODE_P	
45	KEYCODE_Q	
46	KEYCODE_R	
47	KEYCODE_S	
48	KEYCODE_T	
49	KEYCODE_U	
50	KEYCODE_V	
51	KEYCODE_W	
52	KEYCODE_X	
53	KEYCODE_Y	
54	KEYCODE_Z	
55	KEYCODE_COMMA	
56	KEYCODE_PERIOD	
57	KEYCODE_ALT_LEFT	
58	KEYCODE_ALT_RIGHT	
59	KEYCODE_SHIFT_LEFT	
60	KEYCODE_SHIFT_RIGHT	
61	KEYCODE_TAB	
62	KEYCODE_SPACE	
63	KEYCODE_SYM	
64	KEYCODE_EXPLORER	
65	KEYCODE_ENVELOPE	
66	KEYCODE_ENTER	
67	KEYCODE_DEL	
68	KEYCODE_GRAVE	
69	KEYCODE_MINUS	
70	KEYCODE_EQUALS	
71	KEYCODE_LEFT_BRACKET	
72	KEYCODE_RIGHT_BRACKET	
73	KEYCODE_BACKSLASH	
74	KEYCODE_SEMICOLON	
75	KEYCODE_APOSTROPHE	
76	KEYCODE_SLASH	
77	KEYCODE_AT	
78	KEYCODE_NUM	
79	KEYCODE_HEADSETHOOK	
80	KEYCODE_FOCUS	
81	KEYCODE_PLUS	
82	KEYCODE_MENU	
83	KEYCODE_NOTIFICATION	

84	KEYCODE_SEARCH	
85	TAG_LAST_KEYCODE	

命令格式 2：adb shell sendevent [device] [type] [code] [value]

如： adb shell sendevent /dev/input/event0 1 229 1 代表按下按下 menu 键

adb shell sendevent /dev/input/event0 1 229 0 代表按下松开 menu 键

说明：上述的命令需组合使用

另外所知道的命令如下：

Key Name	**CODE**
MENU	229
HOME	102
BACK (back button)	158
CALL (call button)	231
END (end call button)	107

2. 发送鼠标事件(Touch)：

命令格式：adb shell sendevent [device] [type] [code] [value]

情况 1：在某坐标点上 touch

如在屏幕的 x 坐标为 40，y 坐标为 210 的点上 touch 一下，命令如下

```
adb shell sendevent /dev/input/event0 3 0 40
adb shell sendevent /dev/input/event0 3 1 210

adb shell sendevent /dev/input/event0 1 330 1 //touch
adb shell sendevent /dev/input/event0 0 0 0       //it must have

adb shell sendevent /dev/input/event0 1 330 0 //untouch
adb shell sendevent /dev/input/event0 0 0 0 //it must have
```

注：以上六组命令必须配合使用，缺一不可

情况 2:模拟滑动轨迹（可下载并采用 aPaint 软件进行试验）

如下例是在 aPaint 软件上画出一条开始于（100,200），止于（108,200）的水平直线

```
adb shell sendevent /dev/input/event0 3 0 100 //start from point (100,200)
adb shell sendevent /dev/input/event0 3 1 200

adb shell sendevent /dev/input/event0 1 330 1 //touch
adb shell sendevent /dev/input/event0 0 0 0

adb shell sendevent /dev/input/event0 3 0 101 //step to point (101,200)
adb shell sendevent /dev/input/event0 0 0 0
……                                          //must list each step, here just skip
adb shell sendevent /dev/input/event0 3 0 108 //end point(108,200)
adb shell sendevent /dev/input/event0 0 0 0
```

```
adb shell sendevent /dev/input/event0 1 330 0 //untouch
adb shell sendevent /dev/input/event0 0 0 0
```