自动化测试过程中弹出框监控

手机产品测试中心

版本号	制订人	制订时间	内容
V1.0	曾梦良	2017/12/09	制定初稿

目录

1、背景

在自动化测试过程中，异常弹框对于测试流程的正常进行有非常大的影响。影响整个测试过程正常进行，目前在基本功能以及以及部分含有自动化脚本专项测试自动化测试过程中由于测试时间长，脚本比较多，对于测试过程中的一些异常弹出框不及时处理会导致整个测试 Fail.查看测试结果的时候如果发现是弹框影响又需要重新测试，浪费人力物力，因此在自动化测试过程中需要引入行的弹出框处理机制。

2、弹窗监控原理

要解决异常弹窗影响，我们必须要知道啥时候有弹出框弹出，方便及时高效的处理。因此我们找开发协商在系统中加入弹框广播，在有弹框出现的时候通知自动化程序去处理。其具体实现步骤如下：

（1）在WindowManagerService.java中的addView（）方法中添加弹窗出现时发送广播代码。该方法实现了当当前的界面发生变化时，如果判断当前的界面类型是有dialog出现则发送该广播。通知所有该广播接受者当前有弹出框出现。

```
//#ifdef VENDOR_EDIT
//liangkun@Swdp, 2016.9.14, add for popup window notifier
if(NOTIFY_POPUP && isWinVisible) {
    PopupInfo popupInfo = new PopupInfo(ACTION_POP_ADD, win.getOwningPackage(),
        win.getOwningUid(), type);
    mHandlerFloatWindow.sendMessage(mHandlerFloatWindow.obtainMessage(
        HandlerFloatWindow.POPUP_NOTIFY, popupInfo));
}
//#endif /* VENDOR_EDIT */
```

图1 WindowManagerService中addView（）添加代码

(2)在WindowManagerService.java类中的removeWindowInnerLocked添加当前窗口被移除时发送移除广播。该方法实现了dialog类型的窗口被移除掉（如点击取消或者自然消失）时，发送窗口移除广播。告知所有该广播接收者当前弹出框被移除。

```
if (windows != null) {
    windows.remove(win);
    //#ifdef VENDOR_EDIT
    //liangkun@Swdp, 2016.9.14, add for popup window notifier
    if(NOTIFY_POPUP && win.mAppOpVisibility) {
        PopupInfo popupInfo = new PopupInfo(ACTION_POP_REMOVE, win.getOwningPackage(),
            win.getOwningUid(), win.mAttrs.type);
        mHandlerFloatWindow.sendMessage(mHandlerFloatWindow.obtainMessage(
            HandlerFloatWindow.POPUP_NOTIFY, popupInfo));
    }
```

图2 removeWindowInnerLocked（）添加代码

图2 windowManagerService中的removeWindowInnerLocked（）添加代码

（3）具体的广播发送代码。

```
final static boolean NOTIFY_POPUP = SystemProperties.getBoolean("persist.sys.popup_notifier", false);
private final static int ACTION_POP_ADD = 1;
private final static int ACTION_POP_REMOVE = 0;
private class PopupInfo {
    int action;
    String pkg;
    int uid;
    int type;
    public PopupInfo(int action, String pkg, int uid, int type) {
        this.action = action;
        this.pkg = pkg;
        this.uid = uid;
        this.type = type;
    }

    public String toString() {
        return ("action=" + action + " pkg=" + pkg + " uid=" + uid + " type=" + type);
    }
}
private void sendPopupWinBroadcast(PopupInfo popupInfo) {
    if(mContext == null) return;
    int type = popupInfo.type;
    //only check specified types
    if(type != 2 && type != 2002 && type != 2003 && type != 2005 &&
        type != 2008 && type != 2010) {
        return;
    }
    Intent intent = new Intent("action.oppo.popup.notify");
    intent.putExtra("action", popupInfo.action);
    intent.putExtra("pkg", popupInfo.pkg);
    intent.putExtra("uid", popupInfo.uid);
    intent.putExtra("type", type);
    mContext.sendBroadcast(intent, "android.permission.RETRIEVE_WINDOW_CONTENT");
    Slog.d("popnotify", popupInfo.toString());
}
```

图3 广播发送

至此，系统层面的任务已经完成。合入上述代码后，只要当前有弹出框出现，就会发送广播事件。在自动化测试程序中只要注册了该广播接收器，接收系统发送的action.oppo.popup.notify广播就能够实现对于弹框的监控，实时处理弹出框。效率高，处理及时。

3、Phenix 中弹框监控处理实现

在 Phenix 系统中，对于异常弹窗的处理主要分为 2 大步骤：

（1）配置异常关键字和处理关键字。

这里在云端配置：http://172.17.172.255/Phenix/index.php；

其配置规则为：弹出框识别关键字_处理步骤或者 Res_弹出框关键字_处理步骤的 resourceId. 目前测试开发配置的关键字如下图所示：

测试开发已配置异常字典

如我们看到上图中有“检测到新版本_暂不更新”，代表的意思就是检测到当前有弹出框之后判断当前界面是否包含“检测到行版本”，如果有则点击关键字“暂不更新”。

（2）手机端处理步骤

安装 PhenixTestServer.apk 之后，会有一下四个步骤：

1）开启系统的弹窗监控广播发送开关

```
/**
 * 打开弹窗广播发送开关，有弹出框则发送广播
 * @param flag
 */
public void openPopupNotifier(String flag) {
        SystemProperties.set("persist.sys.popup_notifier", flag);
}
```

开启弹框广播发送代码

2）下载网页上配置的异常关键字到手机存储中。

3）注册广播接收器，接收"action.oppo.popup.notify" 广播。

```
public class PopupReceiver extends BroadcastReceiver {
    private static final int START_FILTER = 0;
    private Context mContext;
    @SuppressWarnings("deprecation")
    @Override
    public void onReceive(final Context context, Intent intent) {
        if (intent != null && intent.getAction() != null) {
            if ("action.oppo.popup.notify".equals(intent.getAction())) {
                if (!intent.getExtra("pkg").equals("com.oppo.PhenixTestServer")) {// Server本身不处理
                    if ((intent.getExtra("type") + "").equals("2") // type== 2
                            && (intent.getExtra("action") + "").equals("1")) { // action ==1 是弹框比
                        mContext = context;
                        mHandler.sendEmptyMessage(START_FILTER);
                    }
                }
            }
        }
    }

    private Handler mHandler = new Handler(new Handler.Callback() {
        @Override
        public boolean handleMessage(Message msg) {
            switch (msg.what) {

                case START_FILTER:
                    new Thread(new Runnable() {
                        @Override
                        public void run() {
                            try {
                                // 根据异常字典处理弹框
                                new UIOperates(mContext).startDialogFilterNewStyle();
                            } catch (JSONException e) {
                                e.printStackTrace();
                            } catch (DocumentException e) {
                                e.printStackTrace();
                            }
                        }
```

手机端接收弹窗广播

4）接收到广播后，通过 AccessbilityService 抓取当前界面的控件，并遍历获取每一个控件的 Text 值，和步骤 2 中的异常关键字做对比，包含则点击配置好的操作步骤。

该方案的优点在于处理速度快，能在几十毫秒就能处理完弹出框，对于配置好的弹出框处理精准，但是该方案也有一些缺点。首先在关键字的配置上要求一定要独特性，关键字配置的太简单或者太常见会导致匹配出错的问题，导致就算匹配到了关键字到了处理步骤仍然出错。其次要经常去维护异常字典库，根据实际情况实时去更新。

4、Phenix 中弹窗是否生效分析

目前小组有伙伴反馈说弹窗监控功能不起作用。弹窗处理失败的原因有好几个，小组伙伴不知道如何分析弹窗处理失败的具体原因是啥。通过多次的实际问题分析，如果发现弹框监控不起作用了，可以通过以下几个步骤来分析：

1、确认需要处理的关键字和步骤是否已经配置。

需要到 Phenix 测试网站上的异常关键字典网页查看是否已配置。

2、确认是否已经将网页上的异常关键字典下载到手机存储中。

在启动 PhenixTestServer.apk 的时候，会将 Phenix 网站上的异常关键字典下载到手机存储中，路径是：\sdcard0\Phenix\PhenixTestServer\keyword.txt；该文件的下载和连接的 wifi 有关，目前只有内网的 wifi 能够下载该文件，外销的 vpn 无法下载该文件。如果该文件不存

在或者大小小于 1kb，则代表异常字典配置文件下载失败。此时即便是在网页上配置了相关的异常关键字和处理步骤也不会起作用。

3、确认系统弹框广播发送开关是否开启或者是否已经集成了该功能。

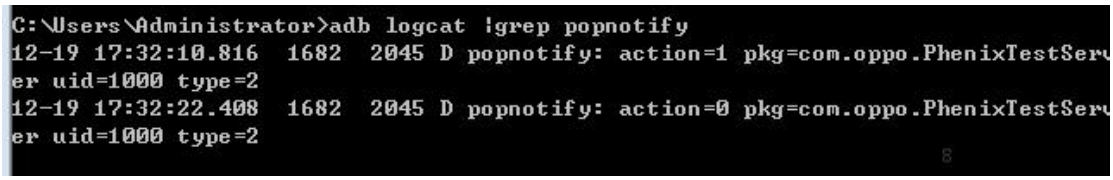
启动 PhenixTestServer.apk 时 如果判断当前系统没有打开弹框广播，则会开启弹框广播发送开关，需要注意这里一定要重启，不重启不起作用。 可以通过 adb 检验方式如下：

是否开启弹窗广播开关校验

当检验广播开关为 true 状态，而弹窗还是处理失败则需要确认当前的平台是否有合入弹框监控代码。这里可以通过 log 来检测。简单的检测步骤如下：

在 cmd 下输入 adb logcat |grep popnotify

在启动 PhenixTestServer.apk。点击主界面的，左上或者右上角的按钮，有弹出框弹出，如果 cmd 下有如下图所示，则代表代码已经合入。



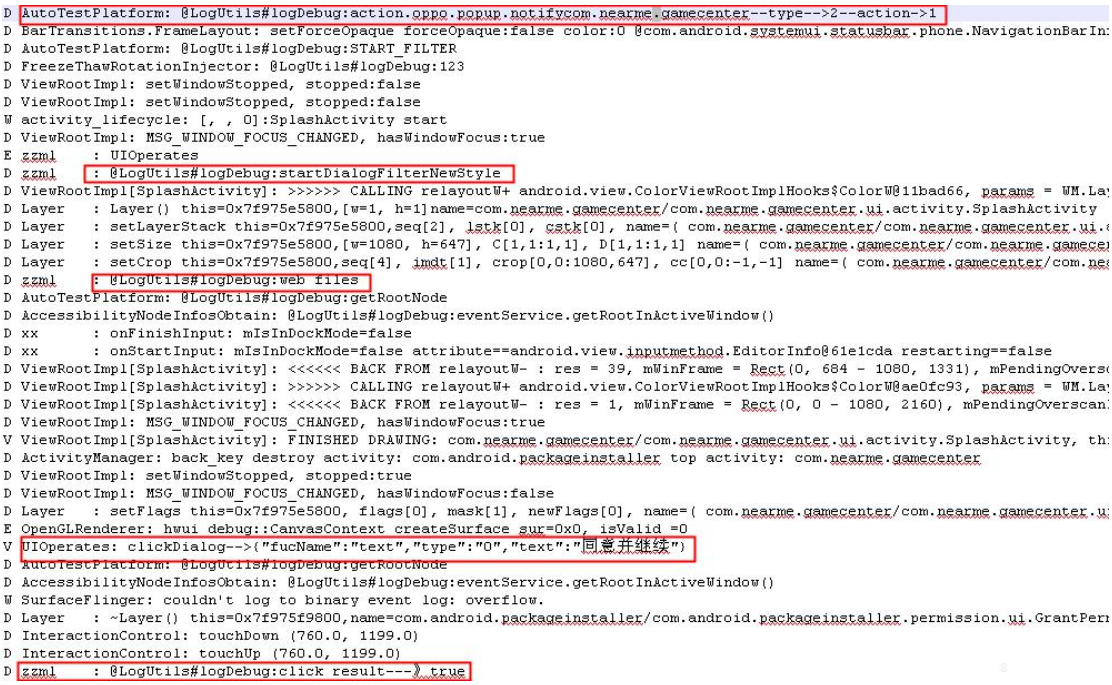
弹框代码是否合入校验

如果发现没有合入改代码，请联系我。目前在行的所有平台都有合入改代码。新平台需要推动开发合入才能使用该功能。

4、弹框处理 log 分析

通过 log 分析能看出当前处理的具体情况。抓取手机的 log, 简单的抓取 log 方式可以执行这个指令：adb logcat >d:\test.txt 连上 usb 情况下会把手机的当前的 log 保存在 D 盘的 text.txt 文件下。

1）处理成功关键 log 分析：



弹窗处理成功 log 展示

本次以声明与条款_同意并继续为例子。关键 log 包含 5 部分。依次用红色框圈了出来。

第一个框：代表当前接收到了系统发送的弹框广播，当前有弹出框出现。

第二个框：代表开启弹窗处理模式

第三个框：代表本次处理的关键字_处理步骤用的是云端下载的文件。

第四个框：代表找到了声明与条款的关键字，要点击“同意并继续”。

第五个框：代表点击找到了“同意并继续”，点击成功。

2）处理弹框失败的关键 log

```
D AutoTestPlatform: @LogUtils#logDebug:action.oppo.popup.notifycom.xunmeng.pinduoduo--type-->2--action->1
D AutoTestPlatform: @LogUtils#logDebug:START_FILTER
D FreezeThawRotationInjector: @LogUtils#logDebug:123
E zzml    : UIOperates
D zzml    : @LogUtils#logDebug:startDialogFilterNewStyle
D zzml    : @LogUtils#logDebug:web files
D AutoTestPlatform: @LogUtils#logDebug:getRootNode
D AccessibilityNodeInfosObtain: @LogUtils#logDebug:eventService.getRootInActiveWindow()
D ViewRootImpl: setWindowStopped, stopped:false
V ViewRootImpl: enableLogLight: enable = false
I Choreographer: Skipped 1 frames!  The application may be doing too much work on its main thread.
W System.err: java.lang.RuntimeException: Appkey is null or empty, Please check AndroidManifest.xml
W System.err:    at u.aly.z.b(SessionTracker.java:104)
W System.err:    at u.aly.z.a(SessionTracker.java:178)
W System.err:    at u.aly.z.c(SessionTracker.java:123)
W System.err:    at com.umeng.analytics.d.g(InternalAgent.java:195)
W System.err:    at com.umeng.analytics.d.a(InternalAgent.java:25)
W System.err:    at com.umeng.analytics.d$1.a(InternalAgent.java:116)
W System.err:    at com.umeng.analytics.g.run(SafeRunnable.java:8)
W System.err:    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1133)
W System.err:    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:607)
W System.err:    at java.lang.Thread.run(Thread.java:761)
D ViewRootImpl[MainFrameActivity]: >>>>>> CALLING relayoutW+ android.view.ColorViewRootImplHooks$ColorW@c566735, params = WM
D Layer   : Layer() this=0x7f94aaa400,[w=1080, h=2016]name=com.xunmeng.pinduoduo/com.xunmeng.pinduoduo.ui.activity.MainFrame
D Layer   : setLayerStack this=0x7f94aaa400,seq[2], lstk[0], cstk[0], name=( com.xunmeng.pinduoduo/com.xunmeng.pinduoduo.ui.
D Layer   : setCrop this=0x7f94aaa400,seq[3], imdt[1], crop[0,0:1080,2016], cc[0,0:-1,-1] name=( com.xunmeng.pinduoduo/com.x
D ViewRootImpl[MainFrameActivity]: <<<<<< BACK FROM relayoutW- : res = 7, mWinFrame = Rect(0, 0 - 1080, 2016), mPendingOvers
I Adreno  : QUALCOMM build                   : a777829, Ic67ba97793
I Adreno  : Build Date                       : 08/30/17
I Adreno  : OpenGL ES Shader Compiler Version: XE031.14.00.04
I Adreno  : Local Branch                     :
I Adreno  : Remote Branch                    :
I Adreno  : Remote Branch                    :
I Adreno  : Reconstruct Branch               :
I Adreno  : PFP: 0x005ff087, ME: 0x005ff063
I OpenGLRenderer: Initialized EGL, version 1.4
D OpenGLRenderer: Swap behavior 1
D ViewRootImpl[MainFrameActivity]: >>>>>> CALLING relayoutW+ android.view.ColorViewRootImplHooks$ColorW@3c23b88, params = WM
D Layer   : Layer() this=0x7f94a4bc00,[w=1080, h=2016]name=com.xunmeng.pinduoduo/com.xunmeng.pinduoduo.ui.activity.MainFrame
D Layer   : setLayerStack this=0x7f94a4bc00,seq[2], lstk[0], cstk[0], name=( com.xunmeng.pinduoduo/com.xunmeng.pinduoduo.ui.
D Layer   : setCrop this=0x7f94a4bc00,seq[3], imdt[1], crop[0,0:1080,2016], cc[0,0:-1,-1] name=( com.xunmeng.pinduoduo/com.x
D ViewRootImpl[MainFrameActivity]: <<<<<< BACK FROM relayoutW- : res = 7, mWinFrame = Rect(0, 0 - 1080, 2016), mPendingOvers
D ViewRootImpl: MSG_WINDOW_FOCUS_CHANGED, hasWindowFocus:true
D BarTransitions.FrameLayout: setForceOpaque forceOpaque:true color:ffffffff @com.android.systemui.statusbar.phone.Navigatic
D xx      : onFinishInput: mIsInDockMode=false
D xx      : onStartInput: mIsInDockMode=false attribute==android.view.inputmethod.EditorInfo@e0fbaf5 restarting==false
V ViewRootImpl[MainFrameActivity]: FINISHED DRAWING: com.xunmeng.pinduoduo/com.xunmeng.pinduoduo.ui.activity.MainFrameActivi
V ViewRootImpl[MainFrameActivity]: FINISHED DRAWING: com.xunmeng.pinduoduo/com.xunmeng.pinduoduo.ui.activity.MainFrameActivi
I ActivityManager: Displayed com.xunmeng.pinduoduo/.ui.activity.MainFrameActivity: +264ms
V ViewRootImpl[MainFrameActivity]: FINISHED DRAWING: com.xunmeng.pinduoduo/com.xunmeng.pinduoduo.ui.activity.MainFrameActivi
D zzml    : @LogUtils#logDebug:App---> com.xunmeng.pinduoduo/.ui.activity.MainFrameActivity    startTime---> +264ms
W Tinker.UpgradePatchRetry: onPatchRetryLoad retry info not exist, just return
E OpenGLRenderer: hwui_debug::CanvasContext createSurface sur=0x0, isValid =0
E OpenGLRenderer: hwui_debug::CanvasContext createSurface sur=0x0, isValid =0
D Layer   : setFlags this=0x7f94aaa400, flags[0], mask[1], newFlags[0], name=( com.xunmeng.pinduoduo/com.xunmeng.pinduoduo.u
D Layer   : setFlags this=0x7f94a4bc00, flags[0], mask[1], newFlags[2], name=( com.xunmeng.pinduoduo/com.xunmeng.pinduoduo.u
D zzml    : @LogUtils#logDebug:click result---》no key word found
```

本次以声明与条款_同意并继续为例子。关键 log 包含 5 部分。依次用红色框圈了出来。

第一个框：代表当前接收到了系统发送的弹框广播，当前有弹出框出现。

第二个框：代表开启弹窗处理模式

第三个框：代表本次处理的关键字_处理步骤用的是云端下载的文件。

第四个框：代表遍历之后没有找到异常关键字里面所有的关键字。处理失败。这里代表当前弹出框处理不了，需要在异常字典里添加对应的异常关键字_处理步骤。