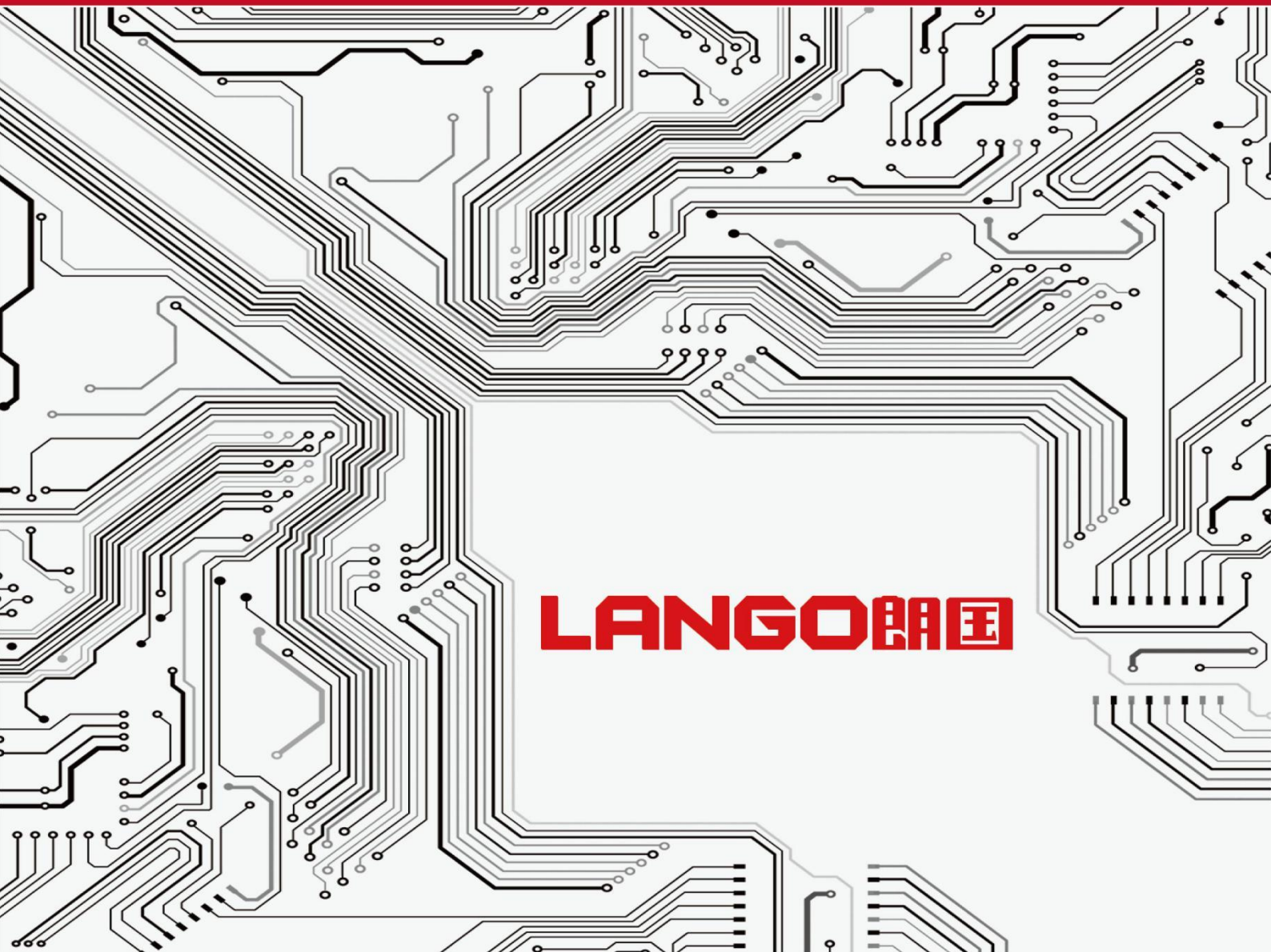
广州朗国电子科技有限公司

Guangzhou LANGO Electronic Technology Co.,Ltd.

朗国白板加速方案使用说明



注意

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

Rights Reserved

NOTE

This document may be updated irregularly for upgrade version or other reasons. Unless some additional agreements, there is no guarantee for any explicitly and implicitly made by statements, information or suggestions in this document.

文档对象

本文档主要适用于：

FAE，RD，APP 开发小组成员

修订记录

修订日期	修订人	邮箱	文档版本	修订说明
20211201			1.0.0	
20220525	吴志豪	zhihao.wu@lango-tech.com	2.0.0	解决透明度重叠，其它 app 区域无法书写的问题
20221112	吴志豪	zhihao.wu@lango-tech.com	2.0.1	文档更新，对新接口加以说明
20230601	吴志豪	zhihao.wu@lango-tech.com	2.0.2	文档更新，标记部分过期接口，添加 clearAcclayer 新接口

目录

一、方案描述

白板是将传统的触摸点击操作，转换成书写体验的一类应用场景。其间涉及大量的图形操作，对程序的性能有极高的要求。为了提高体验，我们引入了基于硬件的白板加速方案。该方案通过直接操作显存空间，由白板通过 JNI 接口，自动控制屏幕刷新。通过加速库的刷新和书写加速，可以更好提升白板的响应速度，为用户提供更流畅的体验。

二、加速框架

该方案由两部分组成：底层的加速库以及上层的 JNI 接口。

1．加速库

加速库与平台方案强相关，无法脱离具体的硬件平台使用。不同的平台，加速方案不同。加速库会预置在系统固件中，用户不需关注其打包细节，直接通过上层 JNI 接口就可以调用。

在一些平台（如 311D2），用户在安装调试自己的 app 时，会出现加速库找不到的情况。可以通过串口修改终端中/system/etc/public.libraries.txt 文件，将 libAccelerateDraw.so 添加到文本中后重启解决。

```
11-02 10:44:50.754  2804  2804 E AndroidRuntime: FATAL EXCEPTION: main
11-02 10:44:50.754  2804  2804 E AndroidRuntime: Process: com.xbh.simplewhiteboarddemo, PID: 2804
11-02 10:44:50.754  2804  2804 E AndroidRuntime: java.lang.UnsatisfiedLinkError: dlopen failed: library "/system/lib/libAccelerateDraw.so" needed or dlopened by "/apex/com
.android.art/lib/libnativeloader.so" is not accessible for the namespace "classloader-namespace"
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at java.lang.Runtime.loadLibrary0(Runtime.java:1087)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at java.lang.Runtime.loadLibrary0(Runtime.java:1008)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at java.lang.System.loadLibrary(System.java:1664)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at com.xbh.whiteboard.AccelerateDraw.<clinit>(AccelerateDraw.java:17)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at com.xbh.whiteboard.AccelerateDraw.getInstance(AccelerateDraw.java:13)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at com.xbh.simplewhiteboarddemo.MainActivity.<init>(MainActivity.java:37)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at java.lang.Class.newInstance(Native Method)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at android.app.AppComponentFactory.instantiateActivity(AppComponentFactory.java:95)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at android.app.Instrumentation.newActivity(Instrumentation.java:1253)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3335)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3595)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:85)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionExecutor.java:135)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.java:95)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at android.app.ActivityThread$H.handleMessage(ActivityThread.java:2066)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at android.os.Handler.dispatchMessage(Handler.java:106)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at android.os.Looper.loop(Looper.java:223)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at android.app.ActivityThread.main(ActivityThread.java:7664)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at java.lang.reflect.Method.invoke(Native Method)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:592)
11-02 10:44:50.754  2804  2804 E AndroidRuntime:        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:952)
11-02 10:44:50.757   615  2828 I DropBoxManagerService: add tag=data_app_crash isTagEnabled=true flags=0x2
11-02 10:44:50.757   615   944 W ActivityTaskManager:   Force finishing activity com.xbh.simplewhiteboarddemo/.MainActivity
11-02 10:44:50.789  2804  2804 I Process : Sending signal. PID: 2804 SIG: 9
11-02 10:44:50.807   615   987 I ActivityManager: Process com.xbh.simplewhiteboarddemo (pid 2804) has died: fg  TOP
```

2．JNI 接口

JNI 接口是加速库为 Java 侧调用提供的模块入口。它主要包括三大功能：加速库资源初始化和回收，屏幕局部刷新，触摸事件加速。

开始加速前，需要将屏幕对应的分辨率传递给加速库，完成逻辑的初始化。之后，开启加速库，加速库会对屏幕内容进行冻结，系统不再拥有对屏幕的刷新权限。屏幕内容的刷新，交由加速库控制。直到停止加速后，系统才能恢复屏幕刷新。

用户需要谨慎使用加速逻辑。确保加速过程中，不会异常崩溃，也不会刷新逻辑问题。否则，如果在加速过程中，逻辑异常，而又未能及时对屏幕解锁，将会导致触摸无响应，内容不刷新的卡死假象。（注：此现象只在 311d2 11 上会有，其它平台加速时不会对屏幕冻结，其它应用可以正常刷新）

三、接口说明：

JNI 接口包路径为：com.xbh.whiteboard.AccelerateDraw

接口	函数名	备注
初始化	public native void accelerateInit(int w, int h);	其中 w，h 代表为屏幕分辨率对应的宽高
资源回收	public native void accelerateDeInit();	回收加速库资源，一般是应用退出时调用
开启加速	public native void startAccelerateDraw();	开始加速
停止加速	public native void stopAccelerateDraw();	停止加速
局部刷新	public native void refreshAccelerateDrawV2(int screenX, int screenY, int w, int h, Bitmap bitmap, int bitmapX, int bitmapY,boolean isNotFilter);	通知加速库，把 bitmap 的 指定区域（src）贴到屏幕的指定区域(dst),其中 src=(bitmapX,bitmapY,bitmapX+w,bitmapY+h); dst=(x,y,x+w,y+h) isNotFilter 是否不过滤透明度为 0 的像素，默认为 false
局部刷新	public native void refreshAccelerateDraw(int screenX, int screenY, int w, int h, Bitmap bitmap);	局部刷新 src=(x,y,x+w,y+h); dst=(x,y,x+w,y+h)

清空加速层数据	public native void clearAccLayer()	停止加速后，还需要调用此方法来清空加速层的数据。此时加速层完全不可见
清空加速层指定区域数据	public native void clearAccLayerRect(int x, int y, int w, int h)	clearAccLayer 是清除加速层所有像素数据，clearAccLayerRect 是清除加速层指定区域像素数据。

过期接口是为了兼容之前加速库版本，在后续新版本中，功能会无效，后续会清除，请不应该使用。

过期接口	函数名	备注（为后面可能会删除）

刷新 surface	public native void refreshSurface(Surface surface, Bitmap bitmap);	过期，不推荐使用，效果等同于：
		Canvas canvas = mSurfaceHolder.lockCanvas();
		canvas.drawBitmap(bitmap,null, null, null);
		mSurfaceHolder.unlockCanvasAndPost(canvas);

添加防覆盖内容	public native void getActivityBitmap(Bitmap bitmap);	过期，不推荐使用，无需此方法，refreshAccelerateDraw 中使用背景透明图。 将菜单栏等不愿被擦除的区域预存在加速库中，避免板擦界面异常
显示触摸轨迹球	public native void showTrackBall(boolean flag);	是否显示手指触摸轨迹球特效。true 开启，false 关闭。
开启触摸事件加速	public native void setStartEventPull(boolean flag)	是否开启触摸加速。true 开启，false 关闭。
触摸事件加速事件回调	public void callback(int id, float x, float y, float width, float height, float stroke, int status, int type)	触摸事件加速开启后，触摸事件回调入口

四、Demo 实例:

1．权限

加速库需要操作显存空间,需要为白板 APP 赋予 system 权限。在 APP 工程的 AndroidManifest.xml 中添加 android:sharedUserId="android.uid.system"说明。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.xbh.simplewhiteboarddemo"
    android:sharedUserId="android.uid.system">
```

2．加载加速库

加速库以 so 库的形式，打包在系统固件中。java 侧代码在使用之前，需要对其先进行加载。

```
static {
    System.loadLibrary( libname: "AccelerateDraw");
}
```

3．加速库初始化与资源回收

加速库是针对总个屏幕的刷新区域进行管理。对其传递的区域和坐标，需要转换成全屏对应的坐标系。如果加速库初始化宽高和分辨率不一致，会导致书写时花屏，或者坐标偏移。

```
@Override
public void surfaceCreated(SurfaceHolder holder) {
    Log.d(TAG, msg: "surfaceCreated");
    mAcd.accelerateInit(Util.SCREEN_WIDTH, Util.SCREEN_HEIGHT);
    initEventCallBack(Util.INPUTEVENT_DRAW);
}

@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    Log.d(TAG, msg: "surfaceDestroyed");
    cleanSurfaceView();
    mAcd.accelerateDeInit();
}
```

4．加速开启与屏幕解锁

一般在接收到触摸 down 事件时，开始加速，对 android 屏幕进行锁定；在抬手时，停止加速，对 android 屏幕进行解锁。

开始加速：

```
//开始加速
private void touchDown(int curPointerId, float x, float y) {
    IDrawer drawer = toCreateDrawer(curPointerId);
    mAcd.startAccelerateDraw();
    drawer.touchDown(x, y);
}
```

停止加速：

```
private void touchUp(int curPointerId, float x, float y, boolean releaseAll) {
    synchronized (PEN_LOCKER) {
        paintUp(curPointerId, x, y);
        //刷新总个界面
        if (releaseAll) {
            preEraserPointId = -1;
            clearAllDrawer();
            requestCacheDraw();
            mAcd.stopAccelerateDraw();
        }
    }
}
```

5. 渲染与刷新

加速库不关心书写的内容，它只负责刷新。用户需要自己完成书写内容的渲染动作。android 中 canvas 提供了丰富的绘制接口，用户根据自身产品定义，将触摸事件转换成书写内容。

```
@Override
public void draw(DrawSurfaceView drawSurfaceView) {
    if (drawSurfaceView == null) {
        return;
    }

    Canvas drawCanvas    = drawSurfaceView.getDrawCanvas();
    if(drawCanvas == null) {
        return;
    }

    drawCanvas.drawPath(mCurrPath, mPaint);
    drawSurfaceView.refreshCache(getCurrRect());
}
```

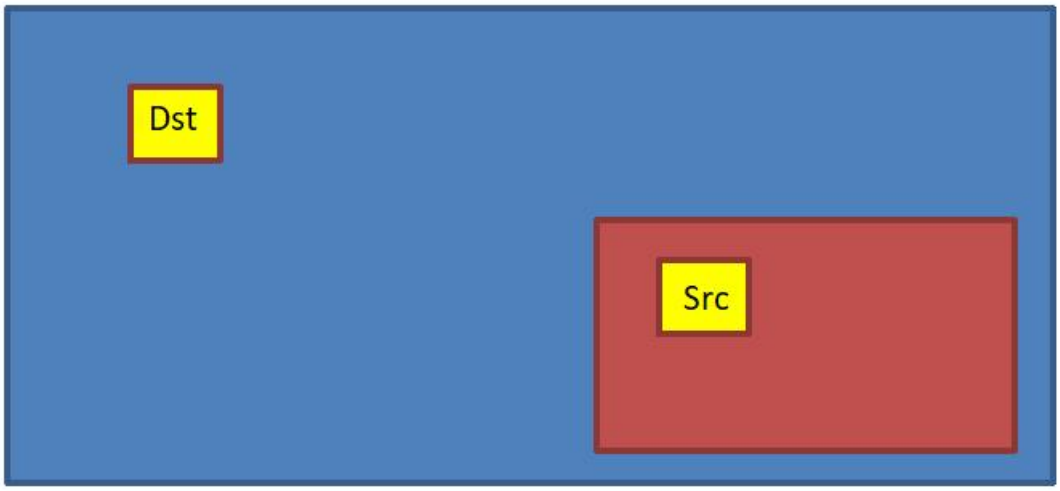
书写结束后，将需要刷新的内容转换成 Bitmap 位图，交由加速库处理。加速库会根据区域，将其刷新到屏幕上。

局部刷新：

```
//刷新书写内容
public void refreshCache(Rect rect) {
    if (rect != null) {
        /*针对多窗适配*/
        int left = mViewRect.left + rect.left;
        int top = mViewRect.top + rect.top;

        //把 书写图层 传入加速库
        mAcd.refreshAccelerateDrawV2(left, top, rect.width(), rect.height(),
                mDrawBitmap, rect.left, rect.top,  isFilter: false);
    }
}
```

使用 refreshAccelerateDrawV2 接口，扩展性更强，可以把 bitmap 的指定区域，贴到屏幕上的指定区域，如下图，蓝色区域是全屏幕，红色区域是 bitmap，把 bitmap 中的黄色区域（src）贴到屏幕左上角黄色区域（dst），区域的宽高是一致的，由 w,h 决定。isNotFilter 参数，是否不过滤透明度为 0 的像素，默认为 false，也就是透明像素会被过滤。（例：在书写模式中，bitmap 为透明背景，bitmap 上只有笔迹）



全局刷新：

```
//刷新总个屏幕
public void requestCacheDraw() {
    synchronized (mSurfaceHolder) {

        //绘制之前，将书写图层叠加到 成熟图层
        mCacheCanvas.drawBitmap(mDrawBitmap, src: null, mScreenRect, paint: null);
        mDrawBitmap.eraseColor(Color.TRANSPARENT);

        Canvas canvas = mSurfaceHolder.lockCanvas();
        if (canvas == null) return;
        canvas.drawBitmap(mBgBitmap, src: null, mScreenRect, paint: null);
        canvas.drawBitmap(mCacheBitmap, src: null, mScreenRect, paint: null);

        mSurfaceHolder.unlockCanvasAndPost(canvas);
    }
}
```

6．事件加速

朗国的系统中已经对触摸事件的分发做了优化，使触摸事件的分发能更快到达 app 层，无需 app 做任何适配。

五、其他

为了更好的将加速与书写逻辑解耦，加速库将渲染和刷新进行了分离。加速库只负责屏幕刷新，不在乎刷新的内容。用户只需要将待刷新的内容转换成 Bitmap 位图，然后通知加速库刷新该位图中的哪个区域就行。