Android恶意软件原理分析以及在鸿蒙系统上的复现

简介

基于恶意软件(https://github.com/fgkeepalive/AndroidKeepAlive)进行分析

设备&android sdk:

华为P40 androidQ(10.0 API level 29)

1. 基于文件锁的进程保活

1.1 简介

背景：实现一个杀不死的应用，用户体验为启动某个App（A），在后台清理后仍有属于A的进程在运行。

在Android中系统杀死进程一般通过ActivityManagerService中提供的两种方法

- killBackgroundProcess
- forceStopPackage

第一种方式太过柔和，第二种比较强力，因此实现保活就要看force-stop

在 android14.0 可以看到forceStopPackage的实现：

```
1  private void forceStopPackage(final String packageName, int userId, int
   userRunningFlags,
2              String reason) {
3          if
   (checkCallingPermission(android.Manifest.permission.FORCE_STOP_PACKAGES)
4                  != PackageManager.PERMISSION_GRANTED) {
5              String msg = "Permission Denial: forceStopPackage() from pid="
6                      + Binder.getCallingPid()
7                      + ", uid=" + Binder.getCallingUid()
8                      + " requires " +
   android.Manifest.permission.FORCE_STOP_PACKAGES;
9              Slog.w(TAG, msg);
10             throw new SecurityException(msg);
11         }
12         final int callingPid = Binder.getCallingPid();
13         userId = mUserController.handleIncomingUser(callingPid,
   Binder.getCallingUid(),
14                 userId, true, ALLOW_FULL_ONLY, "forceStopPackage", null);
15         final long callingId = Binder.clearCallingIdentity();
16         try {
17             IPackageManager pm = AppGlobals.getPackageManager();
18             synchronized (this) {
19                 int[] users = userId == UserHandle.USER_ALL
20                         ? mUserController.getUsers() : new int[] { userId
   };
```

```
21                 for (int user : users) {
22                     if
(getPackageManagerInternal().isPackageStateProtected(
23                             packageName, user)) {
24                         Slog.w(TAG, "Ignoring request to force stop
protected package "
25                                 + packageName + " u" + user);
26                         return;
27                     }
28                     int pkgUid = -1;
29                     try {
30                         pkgUid = pm.getPackageUid(packageName,
MATCH_DEBUG_TRIAGED_MISSING,
31                                 user);
32                     } catch (RemoteException e) {
33                     }
34                     if (pkgUid == -1) {
35                         Slog.w(TAG, "Invalid packageName: " + packageName);
36                         continue;
37                     }
38                     try {
39                         pm.setPackageStoppedState(packageName, true, user);
40                     } catch (RemoteException e) {
41                     } catch (IllegalArgumentException e) {
42                         Slog.w(TAG, "Failed trying to unstop package "
43                                 + packageName + ": " + e);
44                     }
45                     if (mUserController.isUserRunning(user,
userRunningFlags)) {
46                         forceStopPackageLocked(packageName,
47                                                 UserHandle.getAppId(pkgUid),
48                                                  false /* callerWillRestart
*/,
49                                                  false /* purgeCache */,
50                                                  true /* doIt */,
51                                                  false /* evenPersistent */,
52                                                  false /* uninstalling */,
53                                                  true /* packageStateStopped
*/,
54                                                  user,
55                                 reason == null ? ("from pid " + callingPid)
: reason);
56                         finishForceStopPackageLocked(packageName, pkgUid);
57                     }
58                 }
59             }
60         } finally {
61             Binder.restoreCallingIdentity(callingId);
62         }
63     }
```

在38行可以看到ActivityManageService会根据packageName和uid进一步执行，在14.0中加了部分检查，继续查看forceStopPackageLocked的实现：

```
1  @GuardedBy("this")
2      final boolean forceStopPackageLocked(String packageName, int appId,
3              boolean callerWillRestart, boolean purgeCache, boolean doit,
```

```
4             boolean evenPersistent, boolean uninstalling, boolean
packageStateStopped,
5             int userId, String reasonString, int reason) {
6         /*...
7             省略
8           ...*/
9         synchronized (mProcLock) {
10            // Notify first that the package is stopped, so its process
won't be restarted
11            // unexpectedly if there is an activity of the package without
attached process
12            // becomes visible when killing its other processes with
visible activities.
13            didSomething = mAtmInternal.onForceStopPackage(
14                    packageName, doit, evenPersistent, userId);
15            int subReason;
16            if (reason == ApplicationExitInfo.REASON_USER_REQUESTED) {
17                subReason = ApplicationExitInfo.SUBREASON_FORCE_STOP;
18            } else {
19                subReason = ApplicationExitInfo.SUBREASON_UNKNOWN;
20            }
21            didSomething |=
mProcessList.killPackageProcessesLSP(packageName, appId, userId,
22                    ProcessList.INVALID_ADJ, callerWillRestart, false /*
allowRestart */, doit,
23                    evenPersistent, true /* setRemoved */, uninstalling,
24                    reason,
25                    subReason,
26                    (packageName == null ? ("stop user " + userId) : ("stop
" + packageName))
27                    + " due to " + reasonString);
28        }
29         /*...
30            清理其他组件
31          ...*/
32    }
```

21行，根据包名、appid、uid继续执行，继续查看mProcessList.killPackageProcessesLSP

在3094行 执行removeProcessLocked

继续在3160行调用 ProcessRecord的killLocked方法

继续在1265行调用 killProcessGroupIfNecessaryLocked

继续在1299行根据uid、mid（pid）执行ProcessList.killProcessGroup

继续在1656行执行Process.killProcessGroup 是一个native方法 在此处找到.....

总之，force-stop通过循环40次每次间隔5ms，杀死进程组，只要应用进程fork够快，理论上可以留下属于应用的进程。

2. 透明背景弹窗

相关链接

https://juejin.cn/post/6951608145537925128#heading-8 悬浮窗

https://developer.android.com/guide/components/activities/background-starts?hl=zh-cn 针对后台启动activity的限制

重点分析

应用弹窗有两种情景：1.在应用界面上弹窗；2.不在应用界面上弹窗

对于第一种情景，很简单 不需要额外的权限，（在应用内展示什么界面由应用自己决定）

对于第二种情景，android添加了针对后台启动activity的限制，当手机中没有应用的界面时，应用的弹窗是不展示的，想要有透明弹窗只能通过申请权限的方式：这种方式在第5小节SYSTEM_OVERLAY_WINDOW权限部分将详细说明。

接下来只展示如何在应用界面上实现透明弹窗（弹窗周围透明），且点击周围无法取消。

目标场景

运行应用后，应用可以弹出覆盖全屏的弹窗（该过程可以循环进行），用户点击弹窗周围无法取消弹窗，点击返回键无法取消，手机界面只有顶部下拉栏和home键可用。

实现方式

可以通过设置背景属性实现

```
// 首先定义透明弹窗类
public class TransparentDialog extends Dialog {
    public TransparentDialog(@NonNull Context context) {
        super(context);
        // 设置窗口背景透明
        getWindow().setBackgroundDrawable(new
ColorDrawable(Color.TRANSPARENT));
    }
}
```

```
// 创建自定义 Dialog 对象
TransparentDialog dialog = new TransparentDialog(MainActivity.this);
// 设置 Dialog 的布局
dialog.setContentView(R.layout.dialog_layout);
// 显示 Dialog
dialog.show();
//设置透明背景黑暗程度为0，达到完全透明的效果
dialog.getWindow().setDimAmount(0f);
//设置点击弹窗周围弹窗不取消，达到类似霸屏的效果
dialog.setCancelable(false);
```

这一部分可以配合后面的单像素应用实现在手机界面的全屏透明背景弹窗，达到类似霸屏的效果（用户可见桌面，但上方覆盖透明背景弹窗，无法点击桌面）

3. 用户常规清理后台不可见任务——设置最近任务不显示

重点分析

对于普通用户，判断一个应用是否还在运行的方法通常是查看后台任务，但一般用户只会进入最近任务界面（点击最近任务列表按钮）。android10.0中可以通过清单文件配置，使应用不在最近任务列表展示，对于普通用户来说很难察觉应用停留在后台运行。

对于开发者查看应用是否在运行可以用adb shell的ps命令，也可以在手机中强制停止：设置->应用和服务->应用管理->找到目标应用->强行停止，使用该方法即可杀死应用。

在实际测试中发现，即使应用不在最近任务显示，但是如果清理后台有清理任务选项（有其他应用最近运行过），普通用户点击清理最近任务按钮，系统会一并将没有显示的后台任务也一并清除。

目标场景

用户运行应用后，想要清理应用后台时点击最近任务，界面不显示应用对应任务，做到用户无感知停留后台。

实现方式

可以在清单文件AndroidManifest.xml中设置activity属性达到任务不可见效果。

此时对于常规清理后台，是杀不死应用的。

```
<activity
    android:name=".MainActivity"
    android:exported="true"
    android:excludeFromRecents="true"> // 该行可设置应用在最近任务不显示
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

4. 1像素应用保活

相关链接

http://t.csdnimg.cn/6kOUz 使用1像素应用提升进程优先级进行保活

https://github.com/wangchunfei/KeepAlive 1像素应用保活配合监听熄屏广播

https://www.ihuntto.com/2019/03/24/Android%E8%BF%9B%E7%A8%8B%E4%BF%9D%E6%B4%BB%E4%B9%8B%E4%B8%80%E5%83%8F%E7%B4%A0%E6%82%AC%E6%B5%AE%E7%AA%97/ 1像素应用保活

1像素配合透明弹窗实现霸屏效果，1像素应用可配合

重点分析

系统杀死应用是根据应用优先级来进行。此优先级和内存回收机制相关

应用优先级（oom_adj）查询：在adb shell中 先使用ps命令查看进程的进程号pid， 之后使用cat /proc/查到的pid/ oom_adj，查看对应进程的优先级。

一个应用在前台时（界面可见），oom_adj为0，优先级最高，

退到后台后oom_adj有所升高变为4，优先级降低，

一段时间后（和手机有关，一般为十几秒）变为11，优先级进一步降低，

当oom_adj变为16后，应用即将被系统作为不活跃应用杀死。

因此为了实现应用的保活，可以通过降低应用的oom_adj值（提升优先级）。而一个应用在前台（界面可见）时oom_adj最小，但想做到用户无感知的保活，就不能有明显的界面，因此出现使用1像素应用保活的方法。

目标场景

1 用户锁屏时，应用通过注册动态广播接收锁屏动作，并创建1像素应用，此时用户看不到应用启动了一个界面，应用实际成为前台应用，oom_adj降低，优先级提高，达到保活的目的

2 配合上述透明背景弹窗，尽管此时弹窗仍是依附于activity界面，但activity界面只有1像素，用户运行应用会看到桌面上出现弹窗，点击弹窗周围不取消，桌面无法点击，达到霸屏效果

实现方式

1像素应用

```
// 在Activity的onCreate回调函数设置窗口参数
//
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Window window = getWindow();
    window.setGravity(Gravity.LEFT | Gravity.TOP);
    WindowManager.LayoutParams params = window.getAttributes();
    params.x = 0;
    params.y = 0;
    params.width = 1;
    params.height = 1;
    window.setAttributes(params);
}
```

透明弹窗

```
// 在Activity中使用Handler配合runnable执行定时任务
// 在上述1像素应用之上进行弹窗
Handler handler = new Handler();
handler.postDelayed(new Runnable() {
    @Override
    public void run() {
        count++;
        if (dialog != null) { dialog.dismiss(); }
        dialog = new TransparentDialog(MainActivity.this, count);
        dialog.show();
        handler.postDelayed(this, 1000);
    }
}, 1000);

// 透明弹窗类
class TransparentDialog extends Dialog {
    public TransparentDialog(@NonNull Context context, int count) {
        super(context);
        setContentView(R.layout.dialog_layout);
        // 设置窗口背景透明
```

```
        getWindow().setBackgroundDrawable(new
ColorDrawable(Color.TRANSPARENT));
        //设置透明背景黑暗程度为0，达到完全透明的效果
        getWindow().setDimAmount(0f);
        //设置点击弹窗周围弹窗不取消，达到类似霸屏的效果
        setCancelable(false);
        // 设置弹窗内容
        String str = (String) ((TextView)
findViewById(R.id.textView2)).getText();
        str = "count is : " + count + "\n" + str;
        ((TextView) findViewById(R.id.textView2)).setText(str);
        findViewById(R.id.button2).setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                dismiss();
            }
        });
    }
}
```

5. uses-permission 分析

5.0 简介

从敏感权限出发，分析恶意应用可能的实现途径

相关连接

https://developer.android.google.cn/reference/android/Manifest.permission android官网

https://www.cnblogs.com/shiwei-bai/p/4916794.html android权限总结

https://cloud.tencent.com/developer/article/1578473?shareByChannel=link android权限解析

重点解析

只有用户敏感权限才需要询问用户是否赋予，例如**android.permission.READ_EXTERNAL_STORAGE** 从外部存储读取信息，这类权限应用需引导用户赋予权限才能使用

而一些不太敏感的权限但同时对于保活有重要作用的权限，应用可以在清单文件中直接设置。

接下来对这些不敏感同时又对保活有重要作用的权限进以及部分敏感权限行分析

5.1 android.permission.POST_NOTIFICATIONS | android.permission.USE_FULL_SCREEN_INTENT

简介

通知权限在保活中有重要作用，应用可以通过通知拉起activity。

5.1.1 android.permission.POST_NOTIFICATIONS

该权限是发送通知权限，但是发送通知和显示通知有区别，想要应用的通知显示需要更多的权限

该权限在android13之后才有显著作用，在华为P40手机上，安装的应用默认都允许通知。

5.1.2 android.permission.USE_FULL_SCREEN_INTENT

相关链接

https://developer.android.com/develop/ui/views/notifications?hl=zh-cn

重点分析

该权限和通知显示位置有关，不需询问用户即可配置。配合

```
<uses-permission android:name="android.permission.USE_FULL_SCREEN_INTENT" />

NotificationCompat.Builder notification =
    new NotificationCompat.Builder(MainActivity.this, channelId)
    .setFullScreenIntent(pendingIntent, true)
    ....
```

使用可以将通知以悬浮窗的形式弹出

5.2 android.permission.WAKE_LOCK

简介

检测用户手机熄屏灭屏的权限，配置该权限不需要询问用户，配置后可配合WakeLock对屏幕活动进行检测

相关链接

http://t.csdnimg.cn/A7omT Android应用程序保持后台唤醒(使用WakeLock实现)

http://t.csdnimg.cn/CT1rr PowerManager之WakeLock

level

一共有五个级别，对应用可见的只有四个级别。

```
public static final int PARTIAL_WAKE_LOCK = 0x00000001;
public static final int SCREEN_DIM_WAKE_LOCK = 0x00000006;
public static final int SCREEN_BRIGHT_WAKE_LOCK = 0x0000000a;
public static final int FULL_WAKE_LOCK = 0x0000001a;
{@hide}
public static final int PROXIMITY_SCREEN_OFF_WAKE_LOCK = 0x00000020;
public static final int WAKE_LOCK_LEVEL_MASK = 0x0000ffff;
```

flag

```
//Turn the screen on when the wake lock is acquired.
public static final int ACQUIRE_CAUSES_WAKEUP = 0x10000000;
//When this wake lock is released, poke the user activity timer so the screen
stays on for a little longer.
public static final int ON_AFTER_RELEASE = 0x20000000;
```

重点分析

WakeLock的创建需要传入level和flag参数

主要涉及 level参数：**FULL_WAKE_LOCK**；flag参数：**ACQUIRE_CAUSES_WAKEUP**，

- **level 参数： FULL_WAKE_LOCK** (保持CPU 运转，保持屏幕**高亮**显示，键盘灯也保持**亮度**)

  在第二行设置锁类型 **FULL_WAKE_LOCK** 之后，即使用户设置无操作熄屏时长为15秒，应用仍可保持屏幕不熄灭。

```
1   PowerManager pm = (PowerManager)this.getSystemService(Context.POWER_SERVICE);
2   wakeLock =
3       pm.newWakeLock(PowerManager.FULL_WAKE_LOCK, "cby:PostLocationService");
4   wakeLock.acquire();
```

  即使应用切到后台，屏幕仍不熄灭。配合第3节最近任务不显示，可使用户无法察觉，实现屏幕常亮。

  这种方式最直观

- **flag 参数：ACQUIRE_CAUSES_WAKEUP** (强制使屏幕亮起，这种锁主要针对一些必须通知用户的操作)

  注意该参数的含义是：用该参数创建的锁，在锁被获取（即wakeLock.acquire()之后）时，强制使屏幕亮起。

  注意flag参数不能单独使用，需要配合level参数使用，这里选用FULL_WAKE_LOCK

  配合Handler和Runnable，定时获取锁释放锁，可达到**不间断的唤醒屏幕，用户体验为无法熄灭屏幕，即使按电源键，也会唤起屏幕**

```
1   private TextView numberTextView;
2   private int count = 0;
3   private Handler handler;
4   private Runnable runnable;
5   PowerManager.WakeLock wakeLock = null;
6
7   private void releaseWakeLock2() {
8       if (null != wakeLock) {
9           wakeLock.release();
10      }
11  }
12  private void acquireWakeLock2() {
13      if (null == wakeLock) {
14          PowerManager pm = (PowerManager) this.getSystemService(Context.POWER_SERVICE);
15          wakeLock =
16              pm.newWakeLock(
17              PowerManager.FULL_WAKE_LOCK | PowerManager.ACQUIRE_CAUSES_WAKEUP
18              , "cby:PostLocationService");
19          if (null != wakeLock) {
20              wakeLock.acquire(1000);
21          }
22      } else
23          wakeLock.acquire(1000);
24  }
25
26  private void startCounting() {
```

```
    runnable = new Runnable() {
        @Override
        public void run() {
            count++;
            numberTextView.setText(String.valueOf(count));
            Log.i("cby tag", "i am alive!" + count);
            releaseWakeLock2();
            handler.postDelayed(this, 300); // 300ms后再次执行
            acquireWakeLock2();
        }
    };
    handler.postDelayed(runnable, 3000); // 3s后第一次执行
}

```

在HarmonyOS上的实现

相关链接

...待完善

5.3 android.permission.REORDER_TASKS

重新排列任务顺序的权限，该权限不需要询问用户即可配置

相关链接

http://t.csdnimg.cn/QrxPg 使用 REORDER_TASKS 将后台应用切到前台

https://juejin.cn/post/6907817072185737224 android后台启动

重点解析

- 一个应用可以在不询问权限的情况下直接跳转到另一个应用

  传入应用包名即可实现跳转

```
/**
* 启动本地安装好的第三方 APP
* 注意：此种当时启动第三方 APP 时，如果第三方 APP 当时没有运行，则会启动它
* 如果被启动的 APP 本身已经在运行，则直接将它从后台切换到最前端
*
* @param packageNameTarget :App 包名、如
*                          微博 com.sina.weibo、
*                          飞猪 com.taobao.trip、
*                          QQ com.tencent.mobileqq、
*                          腾讯新闻 com.tencent.news
*/
private void startLocalApp(String packageNameTarget) {
    Log.i("Wmx logs::", "----------------------开始启动第三方 APP=" +
packageNameTarget);
    if (SystemHelper.appIsExist(MainActivity.this, packageNameTarget))
{
        PackageManager packageManager = getPackageManager();
        Intent intent =
packageManager.getLaunchIntentForPackage(packageNameTarget);
        intent.addCategory(Intent.CATEGORY_LAUNCHER);
```

```
18          intent.setFlags(Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED |
Intent.FLAG_ACTIVITY_NEW_TASK);
19
20          /**android.intent.action.MAIN：打开另一程序*/
21          intent.setAction("android.intent.action.MAIN");
22          /**
23          * FLAG_ACTIVITY_SINGLE_TOP:
24          * 如果当前栈顶的activity就是要启动的activity,则不会再启动一个新的
activity*/
25          //intent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
26          startActivity(intent);
27      } else {
28          Toast.makeText(getApplicationContext(), "被启动的 APP 未安装",
Toast.LENGTH_SHORT).show();
29      }
30  }
```

- 跳转之后，跳转前的任务仍在前台，可以通过注册**android.permission.REORDER_TASKS**权限将其从后台提到前台

  在android10之前，配置该权限并结合moveTaskToFront方法，可以将后台应用调到前台。

  在android10以及之后的版本中，moveTaskToFront方法已经失效了，但是可以通过**android.permission.USE_FULL_SCREEN_INTENT**权限配合**Notification**实现（此处将在后面详细解释）。
- 总的来说 **android.permission.REORDER_TASKS**在安卓10.0之后已经不太好用了

5.4 android.permission.WRITE_SYNC_SETTINGS

简介

修改系统设置的权限，如调整声音亮度。

相关链接

https://www.jianshu.com/p/0acb66694860

重点解析

在android 6.0及以后，WRITE_SETTINGS权限的保护等级已经由原来的dangerous升级为signature，这意味着我们的APP需要用系统签名或者成为系统预装软件才能够申请此权限，并且还需要提示用户跳转到修改系统的设置界面去授予此权限

5.5 android.permission.FOREGROUND_SERVICE

简介

前台服务权限，使用该权限启动前台服务必须在通知栏常驻通知。

此方式用户可感知（通知栏常驻通知），属于正常可控的应用保活方式，因此暂不做过多分析

相关链接

https://developer.android.com/develop/background-work/services?hl=zh-cn android文档—前台服务

5.6 android.permission.RECEIVE_BOOT_COMPLETED

简介

和自启动相关的权限。该权限实际是接收开机广播的权限，手机开机后系统会发送广播Boardcast。

相关链接

https://developer.android.com/develop/background-work/background-tasks/broadcasts?hl=zh-cn android文档

重点解析

android.permission.RECEIVE_BOOT_COMPLETED 该权限的含义为，允许接受开机时系统广播。

在android api level26之后 无法使用清单为隐式广播（不专门针对您的应用的广播）声明接收器，但一些不受该限制约束的隐式广播除外。在大多数情况下，可以改用预定作业。

在HuaweiP40上以及常见手机上均有手机管家类应用，对应用开机自启有格外限制，依靠开机广播拉起应用已经不太好实现了，并且在androidQ开始，对广播接收器的注册有了更加严格的限制，详见后面广播监听分析。

总之，单凭静态注册广播接收器的方式在androidQ之后已经不能拉起应用了。

5.7 android.permission.SYSTEM_OVERLAY_WINDOW | android.permission.SYSTEM_ALERT_WINDOW

和弹窗相关的权限，该权限在常见手机上需要引导用户进行配置：设置->应用管理->选择应用->允许应用在其他应用上层显示。两种权限类似。

相关链接

http://t.csdnimg.cn/Rg5Pu 如何从后台唤起应用到前台

http://t.csdnimg.cn/W9ubO 引导用户添加权限

http://t.csdnimg.cn/JxtiE 实现后台弹窗

https://developer.android.com/develop/ui/views/components/dialogs?hl=zh-cn#FullscreenDialog android官网—Dialog

重点解析

在引导用户设置权限（允许应用在其他应用上层展示）后，可配合上节透明弹窗，实现不断的从后台弹窗

首先创建service，在service中使用Handler配合Runnable开启定时任务，在其中循环弹窗

```
// 该示例展示如何让应用从后台弹窗
public class MyService extends Service {
    private int count = 0;
    private Handler handler;

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i("cby tag", "启动！");
```

```
        handler = new Handler();
        count = 0;
        // 1秒后再次执行
        Runnable runnable = new Runnable() {
            @Override
            public void run() {
                count++;
                Log.i("cby tag", "i am alive!" + count);
                handler.postDelayed(this, 3000); // 1秒后再次执行
                Dialog dialog = new
AlertDialog.Builder(MyService.this.getApplicationContext())
                        .setTitle("cby")
                        .setMessage("hhh cby coming")
                        .create();

 dialog.getWindow().setType(WindowManager.LayoutParams.TYPE_APPLICATION_OVE
RLAY);
                dialog.show();
            }
        };
        handler.postDelayed(runnable, 3000); // 1秒后再次执行
        return super.onStartCommand(intent, flags, startId);
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) { return null; }
}
```

6. action android:name="android.settings.INPUT_METHOD_SETTINGS"

...待完善...

7.透明(隐藏图标)

相关链接

https://github.com/RenZhongrui/android-learn/tree/master/013-android-alias%20-hide 透明图标Demo

http://t.csdnimg.cn/cmMVL从点击图标到界面流程

https://www.cnblogs.com/anywherego/p/18221943 android系统启动流程(博客园)

http://t.csdnimg.cn/6N8YH alias使用实例

https://developer.android.com/guide/topics/manifest/activity-alias-element android-alias官网

重点解析

- **图标是如何展示的**

  我们能看到桌面图标实际上是lancher服务在运行，luncher会扫描所有已安装的应用包，从其中清单文件查找满足如下条件的activity：

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

  因此仅更改filter中的内容，可以让应用不被lunchaer扫描，从而桌面不显示图标，例如在mainActivity中的filter中添加data字段，此时安装应用将没有图标（此处是没有图标，不同于下面的透明图标，此时无法通过点击启动应用，外部应用可以隐式启动，但intent要包含三个字段），例如将mainActivity中的filter增加一个data字段

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
    <data android:host="MainActivity" android:scheme="com.cby.functest3"
          tools:ignore="AppLinkUrlError" />
</intent-filter>
```

  此时安装应用桌面上没有图标，想要达到存在透明图标且可点击的效果请看下节。

- **透明效果的图标**
  - **activity-alias**

    用来动态设置应用的别名、图标。

    静态设置应用图标为透明会被系统替换，在HuaweiP40静态将清单文件中的application->icon修改为透明色 #00000000后，安装应用，系统将图标自行替换，不会显示透明效果。

    但**动态设置**透明图标不会被替换，方法为在清单文件注册activity-alias（应用别名），在MainActivity中动态修改

```
<!--清单文件中alias注册-->
<!--enable要设置为true, lunchMode根据需求选择,-->
<!--targetAct要设置为先于alias注册的act-->
<activity-alias
   android:name="alias1"
   android:enabled="true"
   android:icon="@drawable/alias1_icon"
   android:label=""
   android:launchMode="singleTask"
   android:targetActivity=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER"
/>
    </intent-filter>
</activity-alias>
```

```xml
<!--@drawable/alias1_icon 透明图标的xml-->
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
<solid android:color="#00000000"/>
</shape>
```

```
MainActivity.this
    .getPackageManager()
    .setComponentEnabledSetting(
    new ComponentName(MainActivity.this, "com.xxx.xxx.YouActName"),
    PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
    PackageManager.DONT_KILL_APP );

MainActivity.this
    .getPackageManager()
    .setComponentEnabledSetting(
    new ComponentName(MainActivity.this,
"com.xxx.xxx.YouAliasActName"),
    PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
    PackageManager.DONT_KILL_APP );

    // 第4行和第11行传入act名和aliasAct名
```

8. 广播监听

相关链接

http://t.csdnimg.cn/SLDKo 两个应用之间使用广播通信

http://t.csdnimg.cn/6vBwb android-broadcast详解

重点解析

在androidQ（10.0）之后，使用**静态方式**注册的广播接收器在应用没有运行时已经接收不到广播了，甚至有些系统广播在运行时也无法接收。

使用**动态方式**注册receiver可以正常接收，但是对于应用拉活没有什么用。即应用不运行，就接不到广播，通过接收广播拉活的方式自然也行不通了

相关实现

9. Instrumention