

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

相关链接

<https://weishu.me/2020/01/16/a-keep-alive-method-on-android/> 文件锁保活详解

<http://t.csdnimg.cn/lkC3T> linux系统上文件锁的使用

<http://t.csdnimg.cn/HIs7b> android中native开发

重点分析

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
4     (checkCallingPermission(android.Manifest.permission.FORCE_STOP_PACKAGES)
5         != PackageManager.PERMISSION_GRANTED) {
6         String msg = "Permission Denial: forceStopPackage() from pid="
7             + Binder.getCallingPid()
8             + ", uid=" + Binder.getCallingUid()
9             + " requires " +
10         android.Manifest.permission.FORCE_STOP_PACKAGES;
11         Slog.w(TAG, msg);
12         throw new SecurityException(msg);
13     }
14     final int callingPid = Binder.getCallingPid();
15     userId = mUserController.handleIncomingUser(callingPid,
16         Binder.getCallingUid(),
17         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
5      packageStateStopped,
6          int userId, String reasonString, int reason) {
7          /*...
8              省略
9              ...*/
10         synchronized (mProcLock) {
11             // Notify first that the package is stopped, so its process
12             won't be restarted
13             // unexpectedly if there is an activity of the package without
14             attached process
15             // becomes visible when killing its other processes with
16             visible activities.
17             didSomething = mAtmInternal.onForceStopPackage(
18                 packageName, doit, evenPersistent, userId);
19             int subReason;
20             if (reason == ApplicationExitInfo.REASON_USER_REQUESTED) {
21                 subReason = ApplicationExitInfo.SUBREASON_FORCE_STOP;
22             } else {
23                 subReason = ApplicationExitInfo.SUBREASON_UNKNOWN;
24             }
25             didSomething |=
26             mProcessList.killPackageProcessesLSP(packageName, appId, userId,
27                 ProcessList.INVALID_ADJ, callerWillRestart, false /*
28                 allowRestart */, doit,
29                 evenPersistent, true /* setRemoved */, uninstalling,
30                 reason,
31                 subReason,
32                 (packageName == null ? ("stop user " + userId) : ("stop
33                 " + packageName))
34                 + " due to " + reasonString);
35         }
36         /*...
37             清理其他组件
38             ...*/
39     }
```

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。

进程之间通过文件锁感知是否死亡

android10.0上的实现

思路

am杀死进程应当是有顺序的，因此用两组进程进行保活：P1和P2，Q1和Q2。

执行流程：

- fork P1，P1先拿文件锁K1
- fork P2，P2尝试拿锁K1，等待P1死亡
- fork Q1，等待10s后再请求锁K2，等待Q2死亡
- fork Q2，立即拿锁K2，

这样无论am从后往前杀死进程（先杀P1，则P2会唤醒新的进程），还是从后往前杀死进程（先杀Q2，则Q1会唤醒新的进程），均可唤醒新的进程。

代码

[CBY11/MaliceAndroidApp at forkTest_New \(github.com\)](#)

```
1 // 此处展示P1和P2进程
2 char *tmp_txt = (char *) malloc(20);
3 sprintf(tmp_txt, "%s/tmp%d.txt", cFilePath, i + 1 );
4 std::ofstream cbyfile(tmp_txt);
5 cbyfile.write("good", 4);
6 cbyfile.close();
7 int lock_fd = open(tmp_txt, O_CREAT | O_RDWR, 0666);
8 create_Process(lock_fd, tmp_txt, 1);
9 create_Process(lock_fd, tmp_txt, 0);
10
11 void create_Process(int lock_fd, char *file, int first) {
12     // 第3个参数标记是P1还是P2，并方便fork新进程后的管理
13     __android_log_print(ANDROID_LOG_DEBUG, TAG, "cp1 begin !");
14     int pid = fork();
15     if (pid == 0) {
16         close(lock_fd);
17         lock_fd = open(file, O_CREAT | O_RDWR, 0666);
18         if (flock(lock_fd, LOCK_EX) == 0) {
19             // log:"process %d(pid) get lock1", getpid()
20             if (!first) {
21                 // log:"process %d(pid) will create another process",
getpid()
22                 create_Process(lock_fd, file, 0);
23             }
24             while (1) {
25                 // log: "process %d(pid) start endless loop",getpid());
26                 usleep(2000);
27             }
28         }
29     } else {
30         // log:"fork return process %d created process %d",getpid(), pid);
31         return;
32     }
33 }
```

效果&问题

效果：使用adb命令kill指定进程后可以拉起新进程，

问题：

- 清理后台不能拉起新进程，无法保活
- 即便能够实现保活，目前没找到很好的拉活方案。由于fork出来的进程只会复制父进程中执行fork的线程，其他线程不会复制，导致fork出的进程和ui线程失去联系。

HarmonyOS上的实现与分析

思路

类似安卓上的实现，但是模拟器的行为和真机有些不同

- 在模拟器上，清理后台不会一并杀死native层fork出来的子进程
- 真机可杀死，和安卓效果类似

2. 透明背景弹窗Dialog

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。

android10.0上的实现

代码

[CBY11/MaliceAndroidApp_at_onepixelTransparent\(github.com\)](https://github.com/CBY11/MaliceAndroidApp_at_onepixelTransparent)

实现方式

可以通过设置背景属性实现

```
1 // 首先定义透明弹窗类
2 public class TransparentDialog extends Dialog {
3     public TransparentDialog(@NonNull Context context) {
4         super(context);
5         // 设置窗口背景透明
6         getWindow().setBackgroundDrawable(new
7             ColorDrawable(Color.TRANSPARENT));
8     }
9 }
```

```
1 // 创建自定义 Dialog 对象
2 TransparentDialog dialog = new TransparentDialog(MainActivity.this);
3 // 设置 Dialog 的布局
4 dialog setContentView(R.layout.dialog_layout);
5 // 显示 Dialog
6 dialog.show();
7 //设置透明背景黑暗程度为0，达到完全透明的效果
8 dialog.getWindow().setDimAmount(0f);
9 //设置点击弹窗周围弹窗不取消，达到类似霸屏的效果
10 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）

HarmonyOS上的实现

3. 用户常规清理后台不可见任务——设置最近任务不显示

相关链接

<http://t.csdnimg.cn/loTsu> 最近任务是如何展示的

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。

android10.0上的实现

代码

略

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。

```
1 <activity
2     android:name=".MainActivity"
3     android:exported="true"
4     android:excludeFromRecents="true"> // 该行可设置应用在最近任务不显示
5     <intent-filter>
6         <action android:name="android.intent.action.MAIN" />
7         <category android:name="android.intent.category.LAUNCHER" />
8     </intent-filter>
9 </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%B%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

android10.0上的实现

代码

[CBY11/MaliceAndroidApp at onepixelTransparent\(github.com\)](#)

实现方式

使用

1像素应用

```
1 // 在Activity的onCreate回调函数设置窗口参数
2 //
3 @Override
4 protected void onCreate(Bundle savedInstanceState) {
5     super.onCreate(savedInstanceState);
6     window window = getWindow();
7     window.setGravity(Gravity.LEFT | Gravity.TOP);
8     WindowManager.LayoutParams params = window.getAttributes();
9     params.x = 0;
10    params.y = 0;
11    params.width = 1;
12    params.height = 1;
13    window.setAttributes(params);
14 }
```

透明弹窗

```
1 // 在Activity中使用Handler配合Runnable执行定时任务
2 // 在上述1像素应用之上进行弹窗
3 Handler handler = new Handler();
4 handler.postDelayed(new Runnable() {
5     @Override
6     public void run() {
7         count++;
8         if (dialog != null) { dialog.dismiss(); }
9         dialog = new TransparentDialog(MainActivity.this, count);
10        dialog.show();
11        handler.postDelayed(this, 1000);
12    }
13 }, 1000);
14
15 // 透明弹窗类
16 class TransparentDialog extends Dialog {
17     public TransparentDialog(@NonNull Context context, int count) {
18         super(context);
19         setContentView(R.layout.dialog_layout);
20         // 设置窗口背景透明
21         getWindow().setBackgroundDrawable(new
22         ColorDrawable(Color.TRANSPARENT));
23         //设置透明背景黑暗程度为0，达到完全透明的效果
```



```

23 getWindow().setDimAmount(0f);
24 //设置点击弹窗周围弹窗不取消，达到类似霸屏的效果
25 setCancelable(false);
26 // 设置弹窗内容
27 String str = (String) ((TextView)
findViewById(R.id.textView2)).getText();
28 str = "count is : " + count + "\n" + str;
29 ((TextView) findViewById(R.id.textView2)).setText(str);
30 findViewById(R.id.button2).setOnClickListener(new
View.OnClickListener() {
31     @Override
32     public void onClick(View v) {
33         dismiss();
34     }
35 });
36 }
37 }

```

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 | 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。

单独的通知并没法进行恶意行为，但通过通知和前台服务的配合可以实现应用保活，同时，通知的样式也可能成为恶意应用实现入口，可以设置通知无法取消，使用取消回调函数进行拉活。

5.1.1 android.permission.POST_NOTIFICATIONS

相关链接

<http://t.csdnimg.cn/rzIBD> 通知的使用方式

重点解析

普通通知

默认效果：

- 通知只在锁屏以及下拉栏显示，没有横幅展示，查看设置发现默认不授予横幅展示的权限

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（Android10.0）手机上，安装的应用默认都允许通知，即便不声明该权限也可发送通知。

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

该权限用于将横幅展示的通知通过下拉的方式变成小窗应用，同5.1节需要开启横幅通知权限

```
1 <uses-permission android:name="android.permission.USE_FULL_SCREEN_INTENT" />
2
3 NotificationCompat.Builder notification =
4     new NotificationCompat.Builder(MainActivity.this, channelId)
5     .setFullScreenIntent(pendingIntent, true)
6     ....
```

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/A7omI> 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。

```

1 public static final int PARTIAL_WAKE_LOCK = 0x00000001;
2 public static final int SCREEN_DIM_WAKE_LOCK = 0x00000006;
3 public static final int SCREEN_BRIGHT_WAKE_LOCK = 0x0000000a;
4 public static final int FULL_WAKE_LOCK = 0x0000001a;
5 {@hide}
6 public static final int PROXIMITY_SCREEN_OFF_WAKE_LOCK = 0x00000020;
7 public static final int WAKE_LOCK_LEVEL_MASK = 0x0000ffff;

```

flag

```

1 //Turn the screen on when the wake lock is acquired.
2 public static final int ACQUIRE_CAUSES_WAKEUP = 0x10000000;
3 //When this wake lock is released, poke the user activity timer so the screen
  stays on for a little longer.
4 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)
15         this.getSystemService(Context.POWER_SERVICE);
16         wakeLock =
17             pm.newWakeLock(
18                 PowerManager.FULL_WAKE_LOCK |
19                 PowerManager.ACQUIRE_CAUSES_WAKEUP
20                 , "cby:PostLocationService");
21         if (null != wakeLock) {
22             wakeLock.acquire(1000);
23         }
24     } else
25         wakeLock.acquire(1000);
26 }
27
28 private void startCounting() {
29     runnable = new Runnable() {
30         @Override
31         public void run() {
32             count++;
33             numberTextView.setText(String.valueOf(count));
34             Log.i("cby tag", "i am alive!" + count);
35             releaseWakeLock2();
36             handler.postDelayed(this, 300); // 300ms后再次执行
37             acquireWakeLock2();
38         }
39     };
40     handler.postDelayed(runnable, 3000); // 3s后第一次执行

```

android10.0上的实现

代码

[CBY11/MaliceAndroidApp at wakelockTest \(github.com\)](https://github.com/CBY11/MaliceAndroidApp/blob/master/wakelockTest)

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

```
1  /**
2  * 启动本地安装好的第三方 APP
3  * 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
4  * 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
5  *
6  * @param packageNameTarget :App 包名、如
7  * 微博 com.sina.weibo、
8  * 飞猪 com.taobao.trip、
9  * QQ com.tencent.mobileqq、
10 * 腾讯新闻 com.tencent.news
11 */
12 private void startLocalApp(String packageNameTarget) {
13     Log.i("Wmx logs::", "-----开始启动第三方 APP=" +
        packageNameTarget);
14     if (SystemHelper.appIsExist(MainActivity.this, packageNameTarget))
15     {
16         PackageManager packageManager = getPackageManager();
17         Intent intent =
18         packageManager.getLaunchIntentForPackage(packageNameTarget);
19         intent.addCategory(Intent.CATEGORY_LAUNCHER);
20         intent.setFlags(Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED |
21         Intent.FLAG_ACTIVITY_NEW_TASK);
22
23         /**android.intent.action.MAIN: 打开另一程序*/
24         intent.setAction("android.intent.action.MAIN");
25         /**
26         * FLAG_ACTIVITY_SINGLE_TOP:
27         * 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
28         activity*/
29         //intent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
30         startActivity(intent);
31     } else {
32         Toast.makeText(getApplicationContext(), "被启动的 APP 未安装",
33         Toast.LENGTH_SHORT).show();
34     }
35 }
```

- 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。

- 总的来说 **android.permission.REORDER_TASKS**在安卓10.0之后已经不太好用了

android10.0上的实现

略

HarmonyOS上的实现

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，但可以尝试修改常驻通知的样式

相关链接

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

<http://t.csdnimg.cn/BD36V> 修改通知样式

重点分析

在activity中使用startForegroundService来启动前台服务，实际上，并不是该方法决定了启动的服务是前台服务，而是service中使用的startForeground使一个服务成为前台服务，下面给出构建顺序：

- **声明service**，在service启动后的onStartCommand回调方法中开启前台服务
- 在onStartCommand方法中**使用startForeground方法**，传入构造好的notification
- 由于service中使用了startForeground，因此其成为**前台服务**，在唤起该service时就要**使用startForegroundService方法**，而不是startService方法
- 同时，在activity中想要使用startForegroundService就要声明< **uses-permission android:name="android.permission.FOREGROUND_SERVICE" />**权限

在启动前台服务后，通知栏会有一个常驻通知，此时通知是可见的，但是可以在构建通知时指定通知的样式：

```
1 Notification notification = new NotificationCompat
2   .Builder(MyService.this, "cbyChannelID1")
3   .setContent(new RemoteViews(getPackageName(), R.layout.note_view)) // 指定下拉后透明
4   .setContentTitle("")
5   .setContentText("")
6   .setSmallIcon(R.drawable.ic_note) // 指定不下拉时的图标透明
7   .build();
```

```

8
9 // R.layout.note_view
10 <LinearLayout
11     xmlns:android="http://schemas.android.com/apk/res/android"
12     android:layout_width="0dp"
13     android:layout_height="0dp" // 仅设置宽高，通知会成为一条线
14     android:background="#00000000"> // 设置背景为透明，通知栏将看不到通知，也没有占
    位符
15 </LinearLayout>
16
17 //R.drawable.ic_note
18 <selector
19     xmlns:android="http://schemas.android.com/apk/res/android"
20     android:width="1dp"
21     android:height="1dp"
22     android:viewportwidth="1"
23     android:viewportheight="1"> // 设置后 不下拉通知栏时通知栏展示的图标为透明（占
    位）
24 </selector>

```

android10.0上的实现:

代码

[CBY11/MaliceAndroidApp at foregroundServiceTest \(github.com\)](https://github.com/CBY11/MaliceAndroidApp/blob/master/app/src/main/java/com/cby11/malice/foregroundservice/ForegroundServiceTest.java)

HarmonyOS上的实现

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。

android10.0上的实现

略

HarmonyOS上的实现

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

简介

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。

在权限管理页，

- 当在清单文件声明 **SYSTEM_OVERLAY_WINDOW** 权限，对应权限管理中的...(似乎没有对应的权限)
- 当在清单文件声明 **SYSTEM_ALERT_WINDOW** 权限，对应权限管理中的**悬浮窗权限**和是否允许在**其他应用上层展示**，这两个权限是相同的，只要允许一个就能弹窗成功

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

```
1  // 判断是否有悬浮窗权限
2  public boolean isOverlayPermission(Context context) {
3      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
4          return Settings.canDrawOverlays(context);
5      } else {
6          try {
7              Class clazz = Settings.class;
8              Method method =
9                  clazz.getDeclaredMethod("canDrawOverlays", Context.class);
10             return (Boolean) method.invoke(null, context);
11         } catch (Exception e) { e.printStackTrace(); } }
12     return false;
13 }
14
15 // 跳转系统设置-悬浮窗页面
16 public void jumpToPermission() {
17     Intent intent =
18         new Intent(Settings.ACTION_MANAGE_OVERLAY_PERMISSION,
19             Uri.parse("package:" +
20 MainActivity.this.getPackageName()));
21     intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
22     startActivity(intent);
23 }
```



```

23
24
25 @Override
26 public int onStartCommand(Intent intent, int flags, int startId) {
27     Handler handler = new Handler();
28     handler.postDelayed(new Runnable() {
29         @Override
30         public void run() {
31             Dialog dialog = new TransparentDialog(MyService.this, cnt);
32             dialog.getWindow()
33                 .setType(WindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY);
34             dialog.show();
35             handler.postDelayed(this, 1000);
36         }
37     }, 1000);
38     return super.onStartCommand(intent, flags, startId);
39 }
40 // 关于TYPE_APPLICATION_OVERLAY的描述是这样的:
41 // window type: Application overlay windows are displayed above all
42 // activity windows
43 // (types between FIRST_APPLICATION_WINDOW and LAST_APPLICATION_WINDOW)
44 // but below critical system windows like the status bar or IME.

```

android10.0上的实现

代码

[CBY11/MaliceAndroidApp at overlayWindowTest \(github.com\)](https://github.com/CBY11/MaliceAndroidApp/blob/master/app/src/main/java/com/cby11/malice/overlayWindowTest/overlayWindowTest.java)

问题

此方式属于系统提供的合理的弹窗方式，

- 想要弹窗就要申请弹窗权限
- 同时当弹窗存在于手机界面时，通知栏会提示"当前应用正在其他应用上层展示"，可考虑删除此通知达到用户无感知。

HarmonyOS上的实现

6. action

android:name="android.settings.INPUT_METHOD_SETTINGS"

...待完善...

简介

输入法类应用？

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/anywhere/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:

```
1 <intent-filter>
2     <action android:name="android.intent.action.MAIN" />
3     <category android:name="android.intent.category.LAUNCHER" />
4 </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

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

此时安装应用桌面上没有图标(也没有占位符)，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

```
1 <!--清单文件中alias注册-->
2 <!--enable要设置为true，lunchMode根据需求选择，-->
3 <!--targetAct要设置为先于alias注册的act-->
4 <activity-alias
5     android:name="alias1"
```

```

6     android:enabled="true"
7     android:icon="@drawable/alias1_icon"
8     android:label=""
9     android:launchMode="singleTask"
10    android:targetActivity=".MainActivity">
11        <intent-filter>
12            <action android:name="android.intent.action.MAIN" />
13            <category android:name="android.intent.category.LAUNCHER"
14        />
15    </intent-filter>
16 </activity-alias>

```

```

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

```

```

1 MainActivity.this
2     .getPackageManager()
3     .setComponentEnabledSetting(
4         new ComponentName(MainActivity.this, "com.xxx.xxx.YouActName"),
5         PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
6         PackageManager.DONT_KILL_APP );
7
8 MainActivity.this
9     .getPackageManager()
10    .setComponentEnabledSetting(
11        new ComponentName(MainActivity.this,
12        "com.xxx.xxx.YouAliasActName"),
13        PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
14        PackageManager.DONT_KILL_APP );
15
16    // 第4行和第11行传入act名(原组件名)和aliasAct名(新组件名)

```

android10.0上的实现

代码

[CBY11/MaliceAndroidApp at aliasTest \(github.com\)](https://github.com/CBY11/MaliceAndroidApp/blob/master/app/src/main/res/xml/alias_test.xml)

HarmonyOS上的实现

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. Instrumentation

10 窗口穿透应用

- z轴排序覆盖权限弹窗
- window创建特殊类型覆盖锁屏
- 传入特殊id控制native绕过权限检查控制其他应用window
- 主要是窗口覆盖的应用

11. 输入法应用保活

12. 权限残留
