Android进程包活

早前，我在知乎上回答了这样一个问题：[怎么让 Android 程序一直后台运行，像 QQ 一样不被杀死？](https://www.zhihu.com/question/29826231/answer/79475911" \t "_blank)。关于 Android 平台的进程保活这一块，想必是所有 Android 开发者瞩目的内容之一。你到网上搜 Android 进程保活，可以搜出各种各样神乎其技的做法，绝大多数都是极其不靠谱。前段时间，Github还出现了一个很火的“黑科技”进程保活库，声称可以做到进程永生不死。

怀着学习和膜拜的心情进去Github围观，结果发现很多人提了 Issue 说各种各样的机子无法成功保活。

看到这里，我瞬间就放心了。坦白的讲，我是真心不希望有这种黑科技存在的，它只会滋生更多的流氓应用，拖垮我大 Android 平台的流畅性。

扯了这么多，接下来就直接进入本文的正题，谈谈关于进程保活的知识。提前声明以下四点

* 本文是本人开发 Android 至今综合各方资料所得
* 不以节能来维持进程保活的手段，都是耍流氓
* 本文不是教你做永生不死的进程，如果指望实现进程永生不死，请忽略本文
* 当前业界的Android进程保活手段主要分为 黑、白、灰 三种，其大致的实现思路如下：

黑色保活：不同的app进程，用广播相互唤醒（包括利用系统提供的广播进行唤醒）

白色保活：启动前台Service

灰色保活：利用系统的漏洞启动前台Service

**黑色保活**

所谓黑色保活，就是利用不同的app进程使用广播来进行相互唤醒。举个3个比较常见的场景：

场景1：开机，网络切换、拍照、拍视频时候，利用系统产生的广播唤醒app

场景2：接入第三方SDK也会唤醒相应的app进程，如微信sdk会唤醒微信，支付宝sdk会唤醒支付宝。由此发散开去，就会直接触发了下面的 场景3

场景3：假如你手机里装了支付宝、淘宝、天猫、UC等阿里系的app，那么你打开任意一个阿里系的app后，有可能就顺便把其他阿里系的app给唤醒了。（只是拿阿里打个比方，其实BAT系都差不多）

没错，我们的Android手机就是一步一步的被上面这些场景给拖卡机的。

针对场景1，估计Google已经开始意识到这些问题，所以在最新的Android N取消了 ACTION\_NEW\_PICTURE（拍照），ACTION\_NEW\_VIDEO（拍视频），CONNECTIVITY\_ACTION（网络切换）等三种广播，无疑给了很多app沉重的打击。我猜他们的心情是下面这样的

而开机广播的话，记得有一些定制ROM的厂商早已经将其去掉。

针对场景2和场景3，因为调用SDK唤醒app进程属于正常行为，此处不讨论。但是在借助LBE分析app之间的唤醒路径的时候，发现了两个问题：

1. 很多推送SDK也存在唤醒app的功能
2. app之间的唤醒路径真是多，且错综复杂

我把自己使用的手机测试结果给大家围观一下（我的手机是小米4C，刷了原生的Android5.1系统，且已经获得Root权限才能查看这些唤醒路径）



*15组相互唤醒路径*



*全部唤醒路径*

我们直接点开 简书 的唤醒路径进行查看



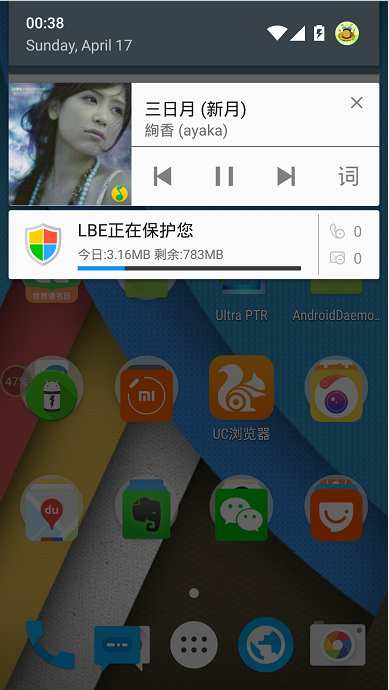
*简书唤醒路径*

可以看到以上3条唤醒路径，但是涵盖的唤醒应用总数却达到了23+43+28款，数目真心惊人。请注意，这只是我手机上一款app的唤醒路径而已，到了这里是不是有点细思极恐。

当然，这里依然存在一个疑问，就是LBE分析这些唤醒路径和互相唤醒的应用是基于什么思路，我们不得而知。所以我们也无法确定其分析结果是否准确，如果有LBE的童鞋看到此文章，不知可否告知一下思路呢？但是，手机打开一个app就唤醒一大批，我自己可是亲身体验到这种酸爽的......

**白色保活**

白色保活手段非常简单，就是调用系统api启动一个前台的Service进程，这样会在系统的通知栏生成一个Notification，用来让用户知道有这样一个app在运行着，哪怕当前的app退到了后台。如下方的LBE和QQ音乐这样：



**灰色保活**

灰色保活，这种保活手段是应用范围最广泛。它是利用系统的漏洞来启动一个前台的Service进程，与普通的启动方式区别在于，它不会在系统通知栏处出现一个Notification，看起来就如同运行着一个后台Service进程一样。这样做带来的好处就是，用户无法察觉到你运行着一个前台进程（因为看不到Notification）,但你的进程优先级又是高于普通后台进程的。那么如何利用系统的漏洞呢，大致的实现思路和代码如下：

* 思路一：API < 18，启动前台Service时直接传入new Notification()；
* 思路二：API >= 18，同时启动两个id相同的前台Service，然后再将后启动的Service做stop处理；

1. public class GrayService extends Service {
2. private final static int GRAY\_SERVICE\_ID = 1001;
3. @Override
4. public int onStartCommand(Intent intent, int flags, int startId) {
5. if (Build.VERSION.SDK\_INT < 18) {
6. startForeground(GRAY\_SERVICE\_ID, new Notification());//API < 18 ，此方法能有效隐藏Notification上的图标
7. } else {
8. Intent innerIntent = new Intent(this, GrayInnerService.class);
9. startService(innerIntent);
10. startForeground(GRAY\_SERVICE\_ID, new Notification());
11. }
12. return super.onStartCommand(intent, flags, startId);
13. }
14. ...
15. ...
16. /\*\*
17. \* 给 API >= 18 的平台上用的灰色保活手段
18. \*/
19. public static class GrayInnerService extends Service {
20. @Override
21. public int onStartCommand(Intent intent, int flags, int startId) {
22. startForeground(GRAY\_SERVICE\_ID, new Notification());
23. stopForeground(true);
24. stopSelf();
25. return super.onStartCommand(intent, flags, startId);
26. }
27. }
28. }

public class GrayService extends Service {

private final static int GRAY\_SERVICE\_ID = 1001;

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

if (Build.VERSION.SDK\_INT < 18) {

startForeground(GRAY\_SERVICE\_ID, new Notification());//API < 18 ，此方法能有效隐藏Notification上的图标

} else {

Intent innerIntent = new Intent(this, GrayInnerService.class);

startService(innerIntent);

startForeground(GRAY\_SERVICE\_ID, new Notification());

}

return super.onStartCommand(intent, flags, startId);

}

...

...

/\*\*

\* 给 API >= 18 的平台上用的灰色保活手段

\*/

public static class GrayInnerService extends Service {

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

startForeground(GRAY\_SERVICE\_ID, new Notification());

stopForeground(true);

stopSelf();

return super.onStartCommand(intent, flags, startId);

}

}

}

代码大致就是这样，能让你神不知鬼不觉的启动着一个前台Service。其实市面上很多app都用着这种灰色保活的手段，什么？你不信？好吧，我们来验证一下。流程很简单，打开一个app，看下系统通知栏有没有一个 Notification，如果没有，我们就进入手机的adb shell模式，然后输入下面的shell命令

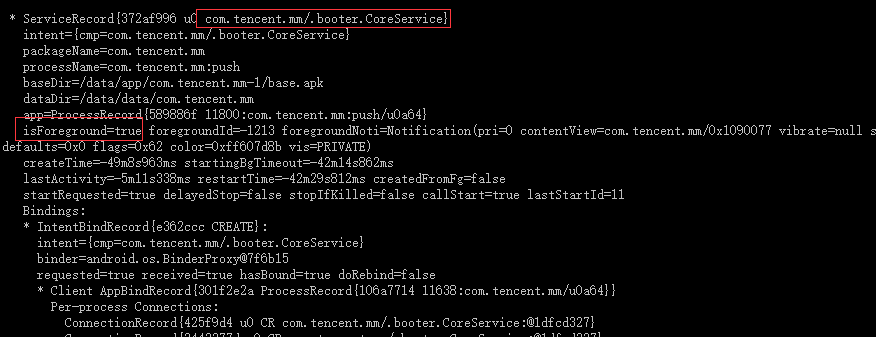
**[js]** [view plain](https://blog.csdn.net/yangxi_pekin/article/details/51498497) [copy](https://blog.csdn.net/yangxi_pekin/article/details/51498497) [print?](https://blog.csdn.net/yangxi_pekin/article/details/51498497)

1. dumpsys activity services PackageName

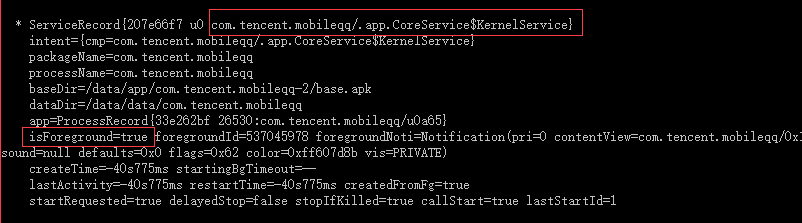
dumpsys activity services PackageName

打印出指定包名的所有进程中的Service信息，看下有没有 isForeground=true 的关键信息。如果通知栏没有看到属于app的 Notification 且又看到 isForeground=true 则说明了，此app利用了这种灰色保活的手段。

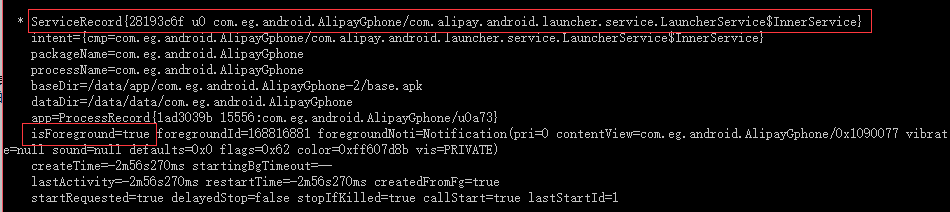
下面分别是我手机上微信、qq、支付宝、陌陌的测试结果，大家有兴趣也可以自己验证一下。



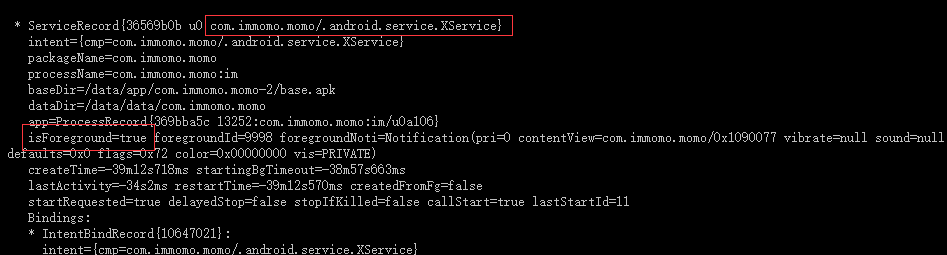
*微信*



*手Q*

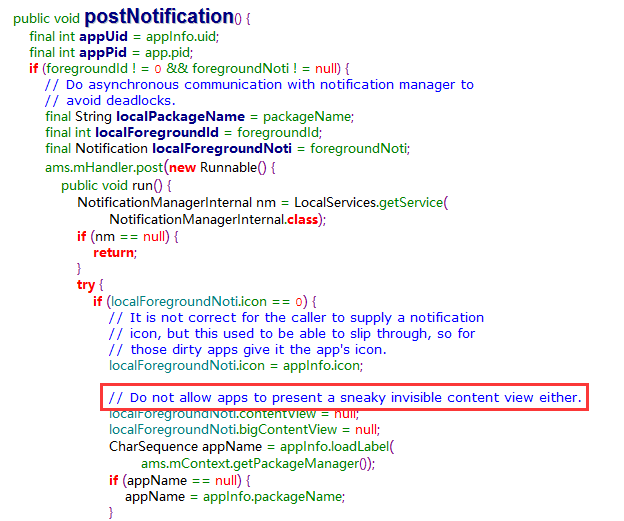


*支付宝*



*陌陌*

其实Google察觉到了此漏洞的存在，并逐步进行封堵。这就是为什么这种保活方式分 API >= 18 和 API < 18 两种情况，从Android5.0的ServiceRecord类的postNotification函数源代码中可以看到这样的一行注释



当某一天 API >= 18 的方案也失效的时候，我们就又要另谋出路了。需要注意的是，使用灰色保活并不代表着你的Service就永生不死了，只能说是提高了进程的优先级。如果你的app进程占用了大量的内存，按照回收进程的策略，同样会干掉你的app。感兴趣于灰色保活是如何利用系统漏洞不显示 Notification 的童鞋，可以研究一下系统的 ServiceRecord、NotificationManagerService 等相关源代码，因为不是本文的重点，所以不做详述。

**唠叨的分割线**

到这里基本就介绍完了 黑、白、灰 三种实现方式，仅仅从代码层面去讲保活是不够的，我希望能够通过系统的进程回收机制来理解保活，这样能够让我们更好的避免踩到进程被杀的坑。

**进程回收机制**

熟悉Android系统的童鞋都知道，系统出于体验和性能上的考虑，app在退到后台时系统并不会真正的kill掉这个进程，而是将其缓存起来。打开的应用越多，后台缓存的进程也越多。在系统内存不足的情况下，系统开始依据自身的一套进程回收机制来判断要kill掉哪些进程，以腾出内存来供给需要的app。这套杀进程回收内存的机制就叫 Low Memory Killer ，它是基于Linux内核的 OOM Killer（Out-Of-Memory killer）机制诞生。

了解完 Low Memory Killer，再科普一下oom\_adj。什么是oom\_adj？它是linux内核分配给每个系统进程的一个值，代表进程的优先级，进程回收机制就是根据这个优先级来决定是否进行回收。对于oom\_adj的作用，你只需要记住以下几点即可：

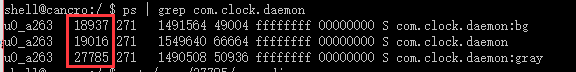
* 进程的oom\_adj越大，表示此进程优先级越低，越容易被杀回收；越小，表示进程优先级越高，越不容易被杀回收
* 普通app进程的oom\_adj>=0,系统进程的oom\_adj才可能<0

那么我们如何查看进程的oom\_adj值呢，需要用到下面的两个shell命令

**[js]** [view plain](https://blog.csdn.net/yangxi_pekin/article/details/51498497) [copy](https://blog.csdn.net/yangxi_pekin/article/details/51498497) [print?](https://blog.csdn.net/yangxi_pekin/article/details/51498497)

1. ps | grep PackageName //获取你指定的进程信息

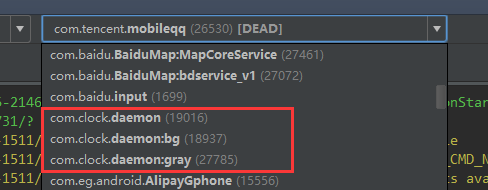
ps | grep PackageName //获取你指定的进程信息



这里是以我写的demo代码为例子，红色圈中部分别为下面三个进程的ID

UI进程：com.clock.daemon  
普通后台进程：com.clock.daemon:bg  
灰色保活进程：com.clock.daemon:gray

当然，这些进程的id也可以通过AndroidStudio获得

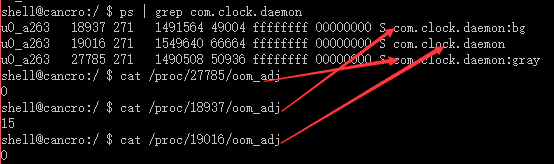


接着我们来再来获取三个进程的oom\_adj

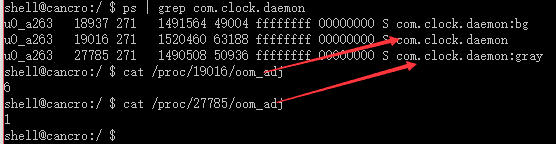
**[js]** [view plain](https://blog.csdn.net/yangxi_pekin/article/details/51498497) [copy](https://blog.csdn.net/yangxi_pekin/article/details/51498497) [print?](https://blog.csdn.net/yangxi_pekin/article/details/51498497)

1. cat /proc/进程ID/oom\_adj

cat /proc/进程ID/oom\_adj



从上图可以看到UI进程和灰色保活Service进程的oom\_adj=0，而普通后台进程oom\_adj=15。到这里估计你也能明白，为什么普通的后台进程容易被回收，而前台进程则不容易被回收了吧。但明白这个还不够，接着看下图



上面是我把app切换到后台，再进行一次oom\_adj的检验，你会发现UI进程的值从0变成了6,而灰色保活的Service进程则从0变成了1。这里可以观察到，app退到后台时，其所有的进程优先级都会降低。但是UI进程是降低最为明显的，因为它占用的内存资源最多，系统内存不足的时候肯定优先杀这些占用内存高的进程来腾出资源。所以，为了尽量避免后台UI进程被杀，需要尽可能的释放一些不用的资源，尤其是图片、音视频之类的。

从Android官方文档中，我们也能看到优先级从高到低列出了这些不同类型的进程：Foreground process、Visible process、Service process、Background process、Empty process。而这些进程的oom\_adj分别是多少，又是如何挂钩起来的呢？推荐大家阅读下面这篇文章：

<http://www.cnblogs.com/angeldevil/archive/2013/05/21/3090872.html>

**总结**

絮絮叨叨写完了这么多，最后来做个小小的总结。回归到开篇提到QQ进程不死的问题，我也曾认为存在这样一种技术。可惜我把手机root后，杀掉QQ进程之后就再也起不来了。有些手机厂商把这些知名的app放入了自己的白名单中，保证了进程不死来提高用户体验（如微信、QQ、陌陌都在小米的白名单中）。如果从白名单中移除，他们终究还是和普通app一样躲避不了被杀的命运，为了尽量避免被杀，还是老老实实去做好优化工作吧。

所以，进程保活的根本方案终究还是回到了性能优化上，进程永生不死终究是个彻头彻尾的伪命题！

文章到此结束，相关简单的实践代码请看

<https://github.com/D-clock/AndroidDaemonService>

文／D\_clock（简书作者）  
原文链接：<http://www.jianshu.com/p/63aafe3c12af>